University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Data Science	Spring semester, 2023. Open access
Writer: Piotr Koloszyc Rongbing Li	Piotr Koloszyc, Rongbing Li (Writer's signature)
Faculty supervisor: Mina Farmanbar, Muhammad Sulaiman	
Thesis title: Comparative Analysis of Sampling Methods for Imbalanced Classification	
Credits (ECTS): 30	
Key words: Machine learning Imbalanced data Binary classification Sampling methods SMOTE MaMiPot Generative Adversarial Network	Pages: 55 + enclosure: 10 Stavanger, 15.06.2023

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Comparative Analysis of Sampling Methods for Imbalanced Classification

Master's Thesis in Computer Science

by

Piotr Kazimierz Koloszyc and

Rongbing Li

Supervisors

Mina Farmanbar

Muhammad Sulaiman

June 15, 2023

"Programming is a nice break from thinking."

Leslie Lamport

Abstract

Assigning class labels to instances is a key component of the machine learning technique known as classification predictive modeling. While concentrating largely on balanced classification problems, which are thought to be the easiest type, the prevalent models and assessment metrics used in classification learning assume an equal distribution of data across class labels.

Many machine learning algorithms fail when the distribution of instances among classes is unbalanced, and the assessment measures used, including classification accuracy, become dangerously misleading. Numerous real-world issues, including as fraud detection, churn prediction, medical diagnosis, and many more, frequently include imbalanced class distributions. In fact, it is frequently more frequent to find unbalanced courses than balanced ones, emphasizing how important it is to solve this problem.

This thesis primarily investigates innovative strategies for managing imbalanced data. One of the approaches examined is the utilization of the Majority and Minority repositioning Technique (MaMiPot) algorithms in combination with different variations of SMOTE and the application of K-means clustering before repositioning. Another method emphasized in this research is the implementation of Generative Adversarial Networks (GAN), a neural network-based technique designed for addressing imbalanced data issues. The evaluation of these approaches was performed on 25 imbalanced datasets obtained from the KEEL repository, encompassing various levels of class imbalance ratios spanning from 5.14 to 129.44.

To assess the performance of the proposed method in mitigating the class imbalance problem, several evaluation metrics were utilized. These metrics include F-score, G-mean, and AUC, which provide valuable insights into the effectiveness of the approach in improving classification results and addressing the challenges posed by imbalanced datasets.

Acknowledgements

We would like to extend our heartfelt appreciation to our supervisors, Professor Mina Farmanbarand and Muhammad Sulaiman, for their invaluable guidance and constructive feedback throughout the course of this thesis. Their unwavering support, expert advice, and continuous encouragement have played a pivotal role in the successful completion of this research. Their profound understanding of thesis research, coupled with their valuable insights and suggestions, have significantly contributed to our academic growth and assisted us in overcoming the obstacles we encountered.

Piotr and Rongbing are also deeply grateful to their parents and friends, whose unwavering belief in our abilities and unwavering support have been a constant source of motivation. Their encouragement to excel and better ourselves in every aspect of life has been instrumental in our journey.

Piotr Kazimierz Koloszyc and Rongbing Li

Stavanger, June 2023.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Unbalanced datasets [1] , or datasets where the distribution of the target class labels is not uniform, are frequently encountered in classification tasks. When training a machine learning model, data imbalance might be problematic since the model tends to be biased towards predicting the majority class because it is trained mostly on that class.

Therefore, before moving further with the modeling pipeline, it is imperative to solve the problem of class imbalance. Sampling techniques are extensively utilized for managing imbalanced data due to their wide adoption and effectiveness. Sampling techniques modify the dataset to enhance the balance of classes. Different sampling strategies are employed to alter the class ratio within an imbalanced modeling dataset, which may be generally divided into three groups.

1. Oversampling strategies: These strategies create fictitious minority class samples. Oversampling methods like Random Oversampling, ADASYN, and SMOTE are often employed.

2. Undersamplig strategies: These strategies entail fewer samples from the majority class. Edited Nearest Neighbour, Random Undersampling, and Tomek links removal are among popular undersampling strategies.

3. Combination of oversampling and undresampling: SMOTE can be combined with various undersampling techniques like ENN (Edited Nearest Neighbors) and Tomek links removal to increase its effectiveness in dealing with unbalanced classes. SMOTE with ENN and SMOTE with Tomek links removal are two famous instances of combining undersampling methods with SMOTE.

However, underampling methods cause a large loss of data points from the majority class, which might have an impact on the model's overall performance. On the other hand, when several synthetic samples are created within the minority class as a result of oversampling procedures, overfitting might result.

SMOTE is a well-known and often applied oversampling technique among data scientists. Within clusters created by the existing minority class samples, it creates fictional minority data points.

SMOTE has been criticized for a number of drawbacks [2]. One of the key problems is that, particularly in dense clusters of minority samples, it might produce synthetic samples that are overly similar to the ones that already exist. Overfitting and worse generalization performance may result from this. Additionally, SMOTE generates synthetic samples in the wrong regions of the feature space since it does not take into account the relative placements and distances of minority samples when choosing nearest neighbors. Additionally, some SMOTE variants may oversample the minority class in a non-selective or global manner, which may prevent some minority samples from benefiting from oversampling.

1.2 Approach and Contributions

The primary objective of this thesis is to assess both conventional and contemporary methods for managing imbalanced data.

One of the more recent methods is the Majority and Minority repositioning Technique (MaMiPot) [3], which focuses on repositioning samples to enable the classifier to learn the decision regions of the minority class. By adjusting the positions of majority class samples, the classifier is encouraged to pay more attention to the minority class. This repositioning scheme can be utilized as a preprocessing step before oversampling with other methods.

In this thesis we make a significant contribution by extending the MaMiPot method by incorporating different SMOTE variants and applying K-means clustering before repositioning. Additionally, we assess a deep learning technique by employing the Generative Adversarial Network (GAN) [4]. GAN utilizes neural networks to produce synthetic minority samples. We evaluate this approach on 25 imbalanced datasets with imbalance ratios ranging from 5.14 to 129.44, sourced from the KEEL repository [5]. The evaluation metrics used include F-score, G-mean, and AUC, which provide insights into the performance of the proposed method in addressing the class imbalance problem.

1.3 Outline

The remainder of this paper is structured as follows.

In Section 2, we provide the necessary background information, as well as a description of several popular sampling methods commonly used in machine learning analysis. Additionally, we discuss the potential flaws associated with these methods, including their limitations and potential biases.

Section 3 is specifically dedicated to outlining the implementation of all the sampling methods developed for this thesis. It includes a detailed explanation of the structure and functioning of these methods.

In Section 4, we describe our experimental setup, including the data sources and methodologies used to evaluate the effectiveness of all data sampling techniques used in this thesis. We provide a detailed overview of the statistical tests and measures employed to analyze our results.

Section 5 presents the results of our experiments, including a comprehensive set of empirical tests that compare the performance of all sampling methods. We provide a detailed analysis of the results, discussing the strengths and weaknesses of each approach and drawing conclusions about the effectiveness of different sampling methods.

Chapter 2

Background

2.1 Class imbalance problem

Class imbalance [1] is a common problem in machine learning where the distribution of classes in the training data is skewed. Specifically, one class (referred to as the minority class) is represented by a much smaller number of samples than another class (referred to as the majority class).

The causes of class imbalance can vary depending on the domain and the task. In some cases, the minority class might simply be rare, making it difficult to collect enough samples to balance the dataset. In other cases, the data collection process itself might introduce bias towards the majority class. For example, in credit card fraud detection, the majority of transactions are legitimate, making it difficult to collect enough fraudulent transactions to balance the dataset. Moreover, fraudulent transactions are usually much more difficult to detect, which could introduce additional bias towards the majority class.

The impact of class imbalance can be severe, particularly in applications where the cost of misclassifying the minority class is high. For example, in medical diagnosis, misclassifying a patient with a rare disease as healthy could have serious consequences. In fraud detection, misclassifying a fraudulent transaction as legitimate could result in significant financial losses. In such cases, a machine learning model that is biased towards the majority class can be unacceptable.

2.2 Classification algorithms

Classification algorithms are employed to assign labels to provided data. In order to establish decision boundaries between classes, classifiers need to undergo training using labeled training data. In this thesis, we utilize a range of diverse classifiers to evaluate the effectiveness of our sampling methods.

Logistic regression [6] is a classification technique that involves attempting to match a sigmoid function to the given samples. The sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Although logistic regression is a straightforward method, it may encounter challenges when dealing with datasets that contain a large number of features.

K-Nearest Neighbour [7] is a classification algorithm that makes predictions by measuring the distance between samples. The choice of distance metric can vary depending on the type of dataset being classified. By employing the distance metric, the classifier identifies the k nearest neighbours to the new sample being classified. The predicted label is determined by the most commonly occurring value among these k nearest neighbours.

Gaussian Naive Bayes [8] is a specific type of Naive Bayes classifier that is used for handling continuous data. The general concept of Naive Bayes involves employing Bayes' theorem to classify samples. Bayes' theorem can be expressed as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.2}$$

The posterior probability of assessment A being true given assessment B (P(A|B)) is equal to the product of the likelihood probability of assessment A being true given assessment B (P(B|A)) and the prior probability of assessment A (P(A)), divided by the evidence probability of assessment B (P(B)).

In this context, we calculate the posterior probability for each class and assign a new sample to the class with the highest posterior probability.

The term "naive" in Naive Bayes stems from the assumption that all features in the dataset are independent of each other. Additionally, Gaussian Naive Bayes assumes that the data within each feature follows a normal distribution. It's important to note that if these assumptions are not met, the performance of the Gaussian Naive Bayes classifier may not be optimal.

Support Vector Machine (SVM) [9] is a classification algorithm that aims to identify the optimal separating hyperplane between classes. In a two-dimensional dataset, this hyperplane would correspond to a line. During the training process, the SVM algorithm explores various lines to find the one that best separates the classes.

To determine the ideal line, SVM identifies the points closest to the line from each class, referred to as support vectors. Subsequently, the algorithm calculates the margin, which represents the distance between the line and the support vectors. The line with the highest margin is considered the best fit. SVM's objective is to discover a hyperplane that maximizes this margin, effectively providing the greatest separation between the classes.

Decision tree [10] is an algorithm used for classifying data by dividing a dataset into different segments based on the values of its features. This process rearranges the dataset into a tree-like structure.

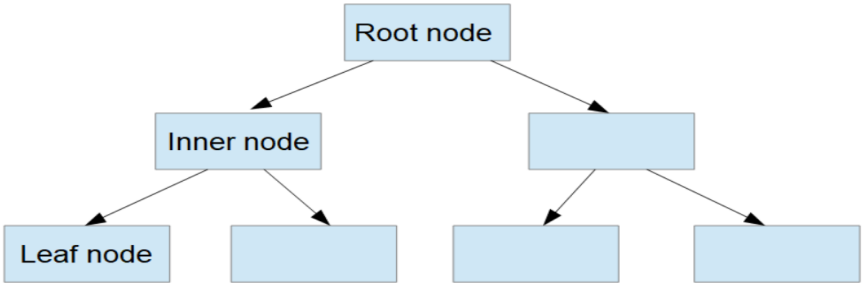

Figure 2.1: Decision tree

The construction process of decision trees is depicted in Figure 2.1. The initial node in the decision tree is referred to as the root node. As the decision tree algorithm progresses, the final nodes eventually become the leaf nodes. The decision tree construction terminates either when a predetermined number of splits is reached or when there is no further data available for splitting.

To construct a decision tree, we need to make decisions about how to perform the splits. These splits are based on the values of the features. The primary objective is to divide the tree in such a way that each node contains only one class, ensuring purity. When training a decision tree, various loss functions like entropy or Gini impurity are utilized to determine the purity of a node before and after a split. The dataset is split based on the feature that yields the lowest loss value. Ideally, the leaf nodes of the decision tree would consist of only one class. However, if that is not the case, the label is determined by the most frequent class present in the leaf node.

Random Forest [11] is an ensemble technique that classifies data by constructing multiple decision trees. It utilizes bagging, a method that generates several subsets of the dataset with replacement, to train the decision trees. For each subset, a decision tree is built. Once all the decision trees are trained, the final prediction is computed by averaging the predictions from each individual decision tree.

Adaptive Boosting (AdaBoost) [12] is an ensemble method that shares similarities with random forest. However, there are distinct differences in how AdaBoost operates. Instead of constructing decision trees for each subset of data like random forest, AdaBoost creates decision stumps. A decision stump is a small decision tree comprising a single root node and multiple leaf nodes.

AdaBoost is trained using the boosting method, which differs from bagging used in random forest. Boosting involves sequentially creating and training each decision stump. After a decision stump is trained, the misclassified samples are assigned higher sample weights. This means that when the next decision stump is trained, it pays more attention to those misclassified samples. The classification error is calculated for each decision stump, and stumps with lower error contribute more significantly to the final prediction.

Multi Layer Perceptron (MLP) [13] is a type of neural network model that is composed of multiple layers. Each layer contains a collection of neurons. The initial layer in the MLP is referred to as the input layer, while the final layer is known as the output layer. In between the input and output layers, there can be any number of hidden layers. The input layer is designed to match the size of the features present in the given samples.

The structure of the MLP is shown in Figure 2.2. Each layer employs its own activation function to modify the data. Every neuron alters the sum of the inputs based on the activation function of its corresponding layer. Each neuron possesses its own weight, which determines the degree of influence the input has on the output of the subsequent layer. The output layer computes the probabilities of a sample belonging to each class. Consequently, the number of neurons in the output layer matches the number of classes in the dataset. The MLP is trained by forwarding a sample through the neural network. The resulting predicted label is compared to the label in the training data. Based on this comparison, the loss is calculated. The loss is then propagated backward through the network, and the weights are updated.

2.3 Undersampling

Undersampling the majority class involves removing instances from the majority class to balance the number of instances in each class. The idea is to focus on the minority

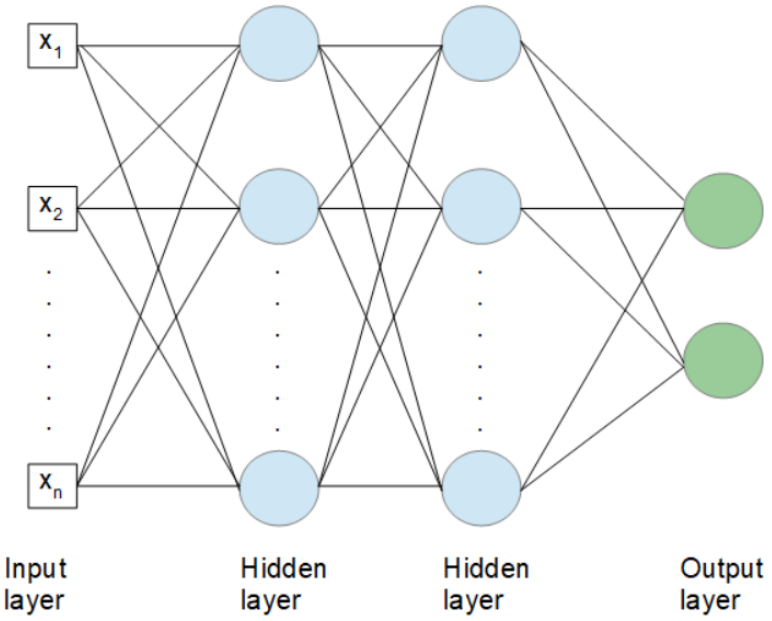

Figure 2.2: Multi Layer Perceptron

class by reducing the influence of the majority class. This can be done using techniques such as random undersampling or Tomek links removal. Undersampling can be useful when the majority class is significantly larger than the minority class, as it can reduce the computational cost of training the classifier and improve the performance on the minority class. However, undersampling can lead to loss of information and may not be effective if the majority class contains important instances.

NearMiss [14] Alternative approaches, such the "near neighbor" methodology and its modifications, have been suggested to solve the worry of potential information loss. The close neighbor family's core algorithms function as follows: The first step is to compute the distances between every instance of the majority class and every instance of the minority class. Then, the k instances of the majority class that are closest to the minority class are chosen. The "nearest" technique will choose k*n examples from the majority class when there are n instances in the minority class.

Based on their least average distances to the three nearest cases in the minority class, the "NearMiss-1" technique locates samples from the majority class. In contrast, "NearMiss-2" chooses the minority class samples that are closest to it. For each sample in the minority class, "NearMiss-3" selects a predetermined number of the closest samples from the majority class.

Edited Nearest Neighbor Rule (ENN) In order to exclude instances from the majority class whose class labels differ from the majority of their three nearest neighbours,

Wilson [15] devised the ENN in 1972. The goal of this strategy is to locate and eliminate instances of the majority class that are found on or near class borders. By identifying these examples using the closest neighbour (NN) concept, the ENN method seeks to improve the classification accuracy of minority instances rather than majority ones. ENN aims to enrich the dataset and maybe enhance the performance of classification models on minority class cases by selectively deleting borderline majority class examples.

Tomek Links Similarly, Tomek [16] presented a successful technique that concentrates on locating samples close to the class borders. The idea of Tomek links is the foundation of this approach.

Tomek links are groups of samples in a dataset that are close to one another and from distinct classes, but are not considered to be nearest neighbors. A (E_i, E_j) pair is said to be a Tomek link if there isn't an example E_l such that $d(E_i, E_l) < d(E_i, E_j)$ or $d(E_j, E_l) < d(E_i, E_j)$, given two instances E_i and E_j belonging to separate classes, and $d(E_i, E_j)$ is the distance between E_i and E_j. When two instances connect via a Tomek link, one of the examples is either noise or both examples are on the edge.

Tomek links removal can be applied as a data cleansing technique or as an under-sampling technique. Only instances from the majority class are dropped when the undersampling technique is applied. Examples of both types are eliminated when the procedure is applied to clean up the data. Tomek links removal can enhance the performance of machine learning models by reducing noisy or borderline samples, particularly in the case of unbalanced datasets.

One Sided Selection (OSS) [17] is a method created especially to address the issues brought on by unbalanced training sets in machine learning. Its main goal is to correct the imbalance by choosing examples from the majority class that are most similar to those from the minority class. By doing this, the method hopes to lessen the impact noisy or unreliable majority class examples have on how the learning algorithm behaves.

The method finds the cases of the majority class that resemble the minority class examples the most. Different distance metrics or similarity measurements can be used to determine how similar things are. A new, more balanced training set is created by combining the selected subset of cases from the majority class with all of the examples from the minority class. This balanced training set offers a more representative and insightful training sample, which can aid in improving the performance of classifiers trained on unbalanced data.

Neighbourhood Cleaning Rule(NCR) [18] is a method for dealing with the problems presented by noisy and borderline cases in unbalanced data by utilizing the one-sided

selection (OSS) concept. By balancing the unbalanced class distribution through data reduction, NCR seeks to improve the identification of challenging tiny classes.

The first step of the procedure is to locate each example's closest neighbors in the minority class. It then looks at these closest neighbors to see if they misclassified any samples from the majority class. The related majority class examples are dropped from the dataset if such misclassifications take place. Through the elimination of noisy and borderline cases from the majority class, this selective removal technique contributes to improving the overall quality of the training data.

2.4 Oversampling

Random over-sampling [19] attempts to solve class imbalance by randomly reproducing minority class examples. It's crucial to remember that this strategy can have a downside. Overfitting has been linked to an increased chance of random oversampling. Because a biased dataset might result from repeating instances of minority classes, the learning system can become unduly sensitive to such cases. As a result, the model can have trouble applying successfully to fresh, unknown data. It is advised to use methods like cross-validation or regularization in addition to random oversampling to reduce the chance of overfitting. Alternative oversampling techniques, such as SMOTE [20], can add more variability and possibly lower the danger of overfitting.

2.4.1 SMOTE

SMOTE [20] presented the Synthetic Minority Over-sampling Technique (SMOTE) to solve the over-fitting problem. SMOTE is frequently referred to as a cutting-edge technique and has been successful in a number of applications. By using the commonalities of existing minority occurrences' features, this method produces synthetic data.

SMOTE initially determines the K-nearest neighbors of each minority instance before generating a synthetic instance. Then, it chooses one of these neighbors at random, and between that neighbor and the original instance, it computes linear interpolations. A new minority instance is produced as a result of this operation nearby the current data points.

In order to address class inequality and broaden the variety of the minority class, SMOTE generates synthetic instances based on feature space similarities. It is well known that using this method will enable classifiers trained on unbalanced datasets to perform better.

SMOTE creates synthetic examples along the line segments connecting minority class examples in feature space. By doing so, SMOTE avoids creating redundant examples and helps to maintain the distribution of the minority class in the original dataset.

Overall, SMOTE is a simple but effective technique for dealing with imbalanced datasets in machine learning. It can help to improve the performance of classifiers on the minority class by creating synthetic examples that reflect the distribution of the minority class in the original dataset.

The SMOTE algorithm, while widely used for imbalanced data classification tasks, has several shortcomings that can impact its effectiveness. Various researchers [3] have discussed these shortcomings:

1. Overfitting and poor generalization performance due to synthetic samples generated in dense clusters that may be near replicas of the seed sample [21].
2. Synthetic samples being located in incorrect regions due to the relative positions and distances of minority samples not being considered in k-nearest-based neighbor selection [21].
3. Synthetic samples being located in the space of the majority class when the nearest neighbor of a seed is in a different cluster [21].
4. Distortion of the true distribution of minority samples due to linear interpolation-based sample generation, leading to incorrect variance estimates and additional challenges for classifiers during testing [22, 23].
5. Ignoring variations in the density of samples in different regions due to using a fixed k value in k-nearest neighbor selection [24].
6. Treating all minority samples equally, even those that may be located close to the decision boundary and are harder to classify, which can lead to synthetic samples being generated in regions where they may not be as useful for improving classification performance [21, 24].

Researchers have proposed various modifications to SMOTE to address these limitations. These modifications highlight ongoing efforts to improve upon SMOTE and develop more effective approaches for imbalanced data classification.

2.4.2 SMOTE variants

ADASYN [25] is an adaptive learning method for issues with unbalanced data categorization is called ADASYN. In order to lessen the bias caused by the unbalanced

data distribution, it creates synthetic data samples for the minority class based on the original data distribution. To achieve this goal, the method employs an adaptive learning approach and modifies weights in response to data distributions. ADASYN enhances learning in two ways by producing synthetic data for minority class instances that are more challenging to learn: lowering the bias created by class imbalance and adaptively pushing the classification decision boundary toward challenging cases.

K-means SMOTE [26] is proposed to overcome some of the limitations of SMOTE. One limitation of SMOTE is that it can generate noisy samples if synthetic samples are generated too close to existing minority class instances.

The suggested approach operates in three steps:

1. Clustering: Using k-means clustering, the input space is divided into k groups.
2. Filtering: The amount of synthetic samples to be generated is distributed depending on the sparsity of minority samples inside each cluster, and clusters with a high proportion of minority class samples are chosen for oversampling.
3. Oversampling: SMOTE is used to obtain the desired ratio of minority and majority instances in each chosen cluster.

The technique avoids creating noisy samples while aiming to remove both between-class imbalances and within-class imbalances

Borderline-SMOTE [27] aims to exclusively oversample the minority cases that are close to the boundary.

According to Han et al. [27], to identify the minority samples near the threshold, Borderline-SMOTE identifies the minority samples that are near the borderline by using a distance measure. Then, synthetic samples are generated along the line segments joining these examples. There are two versions of Borderline-SMOTE: Borderline-SMOTE1 and Borderline-SMOTE2.

- Borderline-SMOTE1 generates synthetic samples for each minority example that has at least one majority neighbor and at least one minority neighbor within its k nearest neighbors. The synthetic samples are generated along the line segment joining each minority example with its nearest minority neighbor.
- Borderline-SMOTE2 generates synthetic samples for each minority example that has at least one majority neighbor within its k nearest neighbors but no minority neighbors within its k nearest neighbors. The synthetic samples are generated

along the line segment joining each minority example with its nearest majority neighbor.

Borderline-SMOTE tries to enhance classification performance while avoiding overfitting brought on by oversampling too many minority cases by concentrating solely on the minority examples close to the borderline.

Density-Based Synthetic Minority Over-sampling TEchnique (DBSMOTE) [28] creates synthetic instances by finding the shortest route between each positive instance and the center of a minority-class cluster. The objective of this method is to produce synthetic instances that are concentrated in the vicinity of the centroid, while being sparse in regions far away from the centroid. DBSMOTE uses DBSCAN to find clusters with arbitrary geometries before oversampling inside clusters, particularly near the centroid. As a result, synthetic instances frequently do not exist in classes with a majority.

DBSMOTE has several benefits over other over-sampling strategies, including its capacity to produce synthetic instances in the most suitable locations, which improves accuracy, F-value, and AUC when using different classifiers. In contrast to other strategies that rely on certain assumptions about the data distribution, DBSMOTE is more versatile since it can handle datasets with various cluster shapes and sizes.

Combined Cleaning and Resampling (CCR) [29] is a cutting-edge solution to the problems presented by unbalanced datasets in pattern recognition. The CCR method deliberately generates fake data points surrounding risky samples in order to oversample the minority class. It uses two main concepts: first, to remove majority instances that are too near to minority examples from the decision border; and second, to deliberately oversample with a greater number of synthetic data points produced around risky samples.

The CCR algorithm's key benefit is that it incorporates cleaning and resampling methods into a single framework, which can enhance classification performance while lowering computing complexity. Additionally, it can deal with skewed distributions, noisy data, and not enough observations, all of which make the classification process more challenging. It is crucial to remember that the efficiency of this technique may vary depending on the dataset being examined.

Majority Weighted Minority Oversampling Technique (MWMOTE) [2] According to their Euclidean distance from the closest majority class samples, MWMOTE determines the hard-to-learn informative minority class samples and provides weights to them. Using a clustering method, it creates synthetic samples from the weighted informative minority class data, making sure that every sample produced is contained inside a minority class cluster.

MWMOTE offers a number of benefits over currently used oversampling techniques. First, it prevents the creation of incorrect and unneeded synthetic samples, which can make learning tasks challenging, and instead creates valuable synthetic minority class examples. To further enhance its performance, it can be used with various undersampling and boundary estimation techniques. Third, depending on the particular issue at hand, MWMOTE involves a variety of characteristics that may be tuned for the greatest results.

Overall, MWMOTE generates valuable synthetic minority class samples while avoiding overfitting and other problems associated with conventional oversampling techniques, making it an excellent strategy for enhancing classifier performance on unbalanced data sets.

SMOTEFUNA [30] Synthetic Minority Over-Sampling Technique based on the Farthest Neighbor Algorithm (SMOTEFUNA) is a brand-new, parameter-free approach for applying synthetic oversampling to unbalanced datasets. SMOTEFUNA creates a cuboid-shaped ovesampling space between two most distant minority samples. By using this strategy, the produced samples are guaranteed to be less sensitive to outliers and more representative of the minority class.

On 33 benchmark machine learning datasets utilizing two distinct classifiers, SMOTEFUNA surpasses existing oversampling methods including SMOTE, ADASYN, and SWIM in terms of ROC and PR curves as well as AUC and AUPR values. Principal component analysis (PCA) projection findings show that SMOTEFUNA inhibits the development of synthetic instances that overlap with the majority class and compels the data to follow the distribution of the minority class. SMOTEFUNA's main benefit is that it can provide high-quality synthetic samples that enhance classification performance on unbalanced datasets and is parameter-free, computationally effective, and economical.

Sampling With the Majority (SWIM) [31] makes use of the knowledge provided by the majority class data to produce synthetic minority class cases, in contrast to conventional oversampling techniques that concentrate on inflating the rare class by creating synthetic data. Due to this, SWIM is able to provide synthetic data in a way that results in a more broad decision boundary without going too far into the majority class. The fundamental benefit of SWIM over current approaches for severe imbalance is that it offers more performance improvements in terms of the geometric mean (g-mean) than other cutting-edge oversampling methods on very unbalanced domains.

SMOTE-Tomek SMOTE (Synthetic Minority Over-sampling Technique) and Tomek links removal are two existing techniques that are combined in SMOTE-Tomek[32].

While Tomek links removal technique locates pairs of samples that are similar to one another but belong to different classes and delete the sample from the majority class from

each pair, SMOTE interpolates between existing samples to create synthetic samples for the minority class. SMOTE-Tomek attempts to oversample the minority class and eliminate noisy majority class samples by combining these two techniques.

For data sets with a small number of positive examples, SMOTE-Tomek produced extremely good results in comparison to other oversampling techniques examined in the study. Compared to other oversampling techniques, it has benefits including enhanced classification performance and decreased overfitting.

SMOTE-Tomek is an effective method for handling imbalanced datasets because it addresses both the problem of class imbalance and noisy samples in a single process. By generating synthetic samples for the minority class and removing noisy samples from both classes, SMOTE-Tomek can improve classification accuracy and reliability in many real-world applications.

SMOTE-ENN The Synthetic Minority Over-sampling Technique (SMOTE) and Edited Nearest Neighbor (ENN) algorithms are combined in the oversampling technique known as SMOTE-ENN [32]. While ENN eliminates noisy and borderline cases from the majority class, SMOTE develops synthetic samples by interpolating between instances of the minority class.

Step 1: SMOTE The first step of SMOTE-ENN is to use the Synthetic Minority Over-sampling Technique (SMOTE) to generate synthetic examples of the minority class. This oversampling method creates new examples by interpolating between existing examples. The basic idea behind SMOTE is to randomly select an example from the minority class and then select one or more of its nearest neighbors. New examples are then created by interpolating between the selected example and its neighbors.

Step 2: ENN The second step of SMOTE-ENN is to use Edited Nearest Neighbors (ENN) to remove any noisy or misclassified examples from both the minority and majority classes. This is an under-sampling method that removes examples that are considered noisy or misclassified based on their nearest neighbors. The basic idea behind ENN is to find the k-nearest neighbors for each sample in the training set and then remove any sample whose class label does not agree with the majority class label among its k-nearest neighbors.

SMOTE-ENN offers the benefit of not only raising the number of minority class instances but also decreasing the number of noisy majority class instances as compared to other oversampling techniques. This aids in enhancing the generalization capabilities of classifiers developed using unbalanced data sets.

By combining these two methods, SMOTE-ENN helps to balance overfitting and underfitting issues associated with imbalanced datasets. The synthetic samples generated by SMOTE help to increase the size of the minority class, while ENN helps to remove any noisy or misclassified examples from both classes.

According to Batista et al. [32] SMOTE-ENN was found to be one of the most effective methods for improving classification performance on imbalanced data sets with a small number of positive instances.

SMOTE-IPF [33] solves the issue of noisy and borderline instances in unbalanced classification. It combines two methods: IPF, which recognizes and eliminates noisy samples, and SMOTE, which evens out the distribution of the class and fills in the interior of minority class sub-parts. SMOTE-IPF functions by combining the SMOTE and IPF procedures. SMOTE is used first to even out the distribution of classes and fill in the interior of minority class sub-parts. IPF is then used to find and eliminate noisy samples. Due to the minority class's under-representation in the dataset and potential for being mistaken for noisy samples, using IPF before SMOTE may result in the removal of all examples from that class.

By interpolating between existing samples, SMOTE creates synthetic examples of the minority class. This evens out the distribution of classes and fills in the interior of minority class subgroups.

The noise filter known as IPF (Iterative Partitioning Filter) locates and eliminates noisy cases from the dataset. It does this by classifying instances with comparable traits and labeling deviations as noise.

SMOTE-IPF has a number of benefits over other oversampling strategies. First, it can handle samples that are noisy and on the edge, which are frequent in datasets with imbalances. Second, it doesn't require prior knowledge of the data distribution and may be used with any kind of unbalanced dataset. Thirdly, by regularizing the class borders, it increases classification accuracy. Last but not least, it is adaptable enough to replace SMOTE with any of its variations while still functioning.

2.5 MaMiPot

The SMOTE algorithm and its variants are commonly used for imbalance learning, but they have some shortcomings. One of the main issues [3] is that they tend to create synthetic samples that are very close to existing minority instances, which can lead to overfitting and poor generalization performance. This is because SMOTE generates

synthetic samples by interpolating between existing minority instances, which can result in a high degree of similarity between the original and synthetic samples.

The MAjority and MInority rePOsitioning Technique (MaMiPot) [3] addresses these shortcomings by proposing a new approach that repositions majority samples in smaller steps, rather than just oversampling the minority class. This helps to avoid distorting the distribution of minority samples while still achieving better balance in the dataset. MaMiPot can be used as a preprocessing step before oversampling with other methods, providing several advantages such as a smaller balancing factor and similar imbalance for training and test data.

2.6 Generative Adversarial Network

Generative Adversarial Network (GAN)[4] is a machine learning model that generates synthetic instances of training data. GAN consists of two neural networks a generator and a discriminator. The purpose of the generator is to generate new synthetic samples that are indistinguishable from real data , while the discriminator aims to differentiate between real training data and the data created by the generator. Throughout the training process, the generator improves its ability to generate more realistic data, while the discriminator enhances its ability to accurately discriminate between real and fake data As the training progresses, the generator becomes capable of producing data that is highly similar to real data, making it difficult for the discriminator to distinguish between the two. This is the point at which the GAN reaches equilibrium.

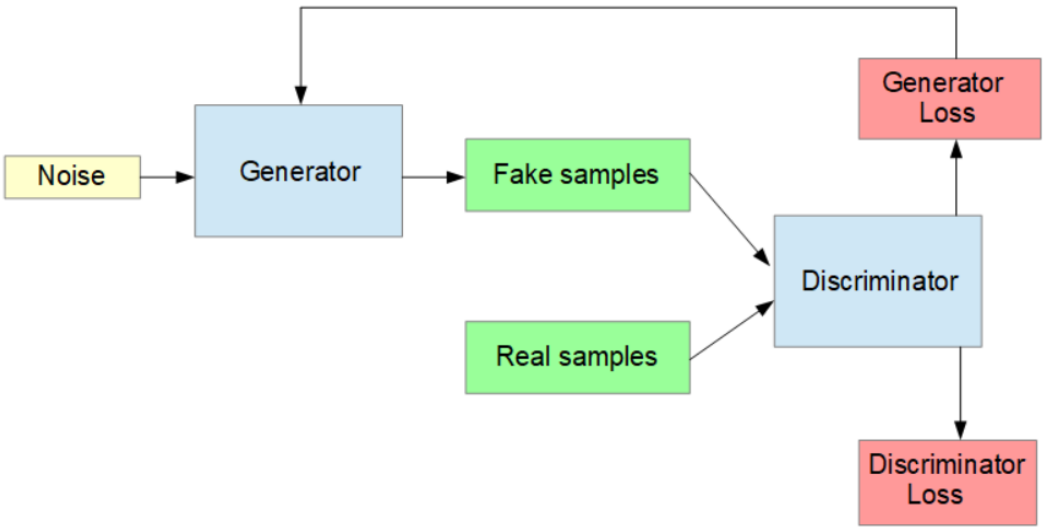

Figure 2.3: GAN structure

The structure of a GAN can be observed in Figure 2.3. Generator and discriminator are connected to each other to form a single, interconnected neural network. The generator's output size matches the input size of the discriminator. The discriminator is associated with two distinct loss functions. One loss function is used to train the discriminator itself, while the other is used to train the generator.

2.7 Performance metrics for imbalanced data

Ghaderi Zefrehi and Altınçay [3] explains that accuracy is inappropriate for evaluating imbalanced datasets. In an imbalanced dataset, the majority class will dominate the accuracy measure, while the minority class will be largely ignored. This can lead to misleading results and poor performance evaluation of classifiers. Therefore, alternative metrics such as F-score, G-mean and AUC evaluate classifier performance on imbalanced datasets. These metrics provide a more comprehensive classifier performance evaluation by considering minority and majority classes. The F-score is defined as the harmonic mean of precision and recall, while G-mean is defined as the geometric mean of sensitivity (recall) and specificity. AUC measures a classifier's ability to distinguish between positive and negative samples across all possible threshold settings.

The confusion matrix is shown in Figure 2.4 where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. True positives are instances that are correctly classified as positive, while true negatives are instances that are correctly classified as negative. False positives and false negatives are instances that are incorrectly classified.

	Predicted positive	Predicted negative
Actual positive	TP (true positive)	FN (false negative)
Actual negative	FP (false positive)	TN (true negative)

Figure 2.4: Confusion matrix in binary classification [34]

$$TNR = \frac{TN}{TN + FP} \tag{2.3}$$

$$Recall(TPR) = \frac{TP}{TP + FN} \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$FPR = \frac{FP}{TN + FP} \tag{2.6}$$

The accuracy[35] of a classifier is calculated by dividing the number of correctly classified instances (both positive and negative) by the total number of instances in the dataset. The formula for accuracy is:

$$Accuracy = \frac{TP + TN}{P + TN + FP + FN} = \frac{TPR + TNR}{2} \tag{2.7}$$

The accuracy formula measures a classifier's overall performance on a given dataset.

2.7.1 F-score

The F-score [36], also known as the F1-score, is a widely used metric in binary classification problems that combines precision and recall into a single score. Precision measures the proportion of true positive predictions among all positive predictions made by the classifier, while recall measures the proportion of true positive predictions among all actual positive samples in the dataset.

The F-score is calculated as the harmonic mean of precision and recall:

$$F1 - score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2.8}$$

The F-score ranges from 0 to 1, with higher values indicating better performance. A perfect classifier would have an F-score of 1.

The advantage of using F-score over other metrics such as accuracy or precision-recall curves is that it considers both false positives and false negatives, which can be especially important when the classes are imbalanced. For example, in a medical diagnosis problem where detecting true positives (patients with a disease) is more important than avoiding false positives (healthy patients diagnosed with a disease), using only accuracy or precision may not be sufficient to evaluate the performance of a classifier.

2.7.2 G-mean

The geometric mean [37] (G-mean) is a performance metric that combines the true positive rate (TPR) and true negative rate (TNR) of a binary classifier. It is calculated by taking the square root of the product of TPR and TNR. The formula for G-mean is:

$$G\text{-}mean = \sqrt{TPR \times TNR} \tag{2.9}$$

where TPR is the number of true positive predictions divided by the total number of actual positive samples, and TNR is the number of true negative predictions divided by the total number of actual negative samples.

The advantage of using G-mean over other metrics such as accuracy or precision-recall curves is that it considers the balance between TPR and TNR, which can be especially important when the classes are imbalanced. For example, if there are many more negative samples than positive samples, a classifier that always predicts negative will have high accuracy but low sensitivity. In this case, G-mean can provide a more balanced evaluation of the classifier's performance.

In addition to its use in binary classification problems, G-mean has also been extended to multi-class classification problems using various approaches such as one-vs-all and pairwise comparisons.

2.7.3 AUC

AUC [37] (area under the ROC curve) is a metric used to evaluate the performance of binary classifiers. It measures how well a classifier can distinguish between positive and negative samples across all possible decision thresholds. The ROC curve in figure 2.5 is a plot of true positive rate (TPR) against false positive rate (FPR) for different decision thresholds of the classifier. TPR is the ratio of true positives to all actual positives, while FPR is the ratio of false positives to all actual negatives.

The AUC represents the area under the ROC curve, which ranges from 0 to 1. A value of 1 indicates perfect classification, while a value of 0.5 indicates random guessing.

The formula for AUC involves integrating over all possible decision thresholds:

$$AUC = \int_0^1 TPR(FPR^{-1}(t))dt \tag{2.10}$$

where TPR is the true positive rate and FPR^{-1} is the inverse function of false positive rate.

AUC has several advantages over other metrics such as accuracy or precision-recall curves.

Firstly, AUC is not affected by class imbalance and misclassification costs. In many real-world classification problems, the number of positive samples is much smaller than

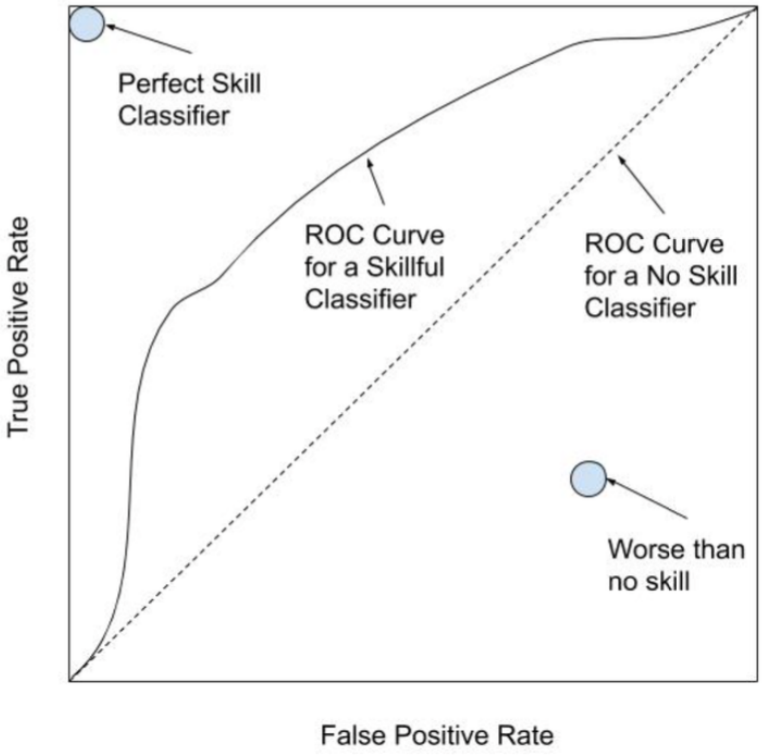

Figure 2.5: ROC Curve [38]

the number of negative samples. This class imbalance can lead to biased classifiers that perform well on negative samples but poorly on positive samples. AUC is not affected by class imbalance because it measures the ability of a classifier to distinguish between positive and negative samples across all possible decision thresholds.

Secondly, AUC provides a single scalar value that summarizes overall performance across all possible decision thresholds. This makes it easier to compare the performance of different classifiers or to evaluate the performance of a single classifier under different operating conditions.

In contrast, accuracy and precision-recall curves depend on the choice of decision threshold and may not provide a complete picture of classifier performance. For example, a classifier with high accuracy may have poor performance on positive samples if it has a high false negative rate. Similarly, precision-recall curves may be misleading if there are few positive samples or if the cost of false positives and false negatives is not balanced.

Overall, AUC is a robust metric for evaluating binary classifiers that overcome some of the limitations of other metrics such as accuracy or precision-recall curves.

Chapter 3

Approach

3.1 Overview

The aim of this thesis is to evaluate various data sampling techniques for imbalanced learning. To achieve this goal, we explore both established traditional approaches and newer methods. To implement the well-known undersampling and oversampling techniques, we utilize two Python libraries: *imblearn* and *smote-variants*. *Imblearn* is a python package offering a range of basic undersampling and oversampling methods. The library *smote-variants* implements 85 different variants of the SMOTE algorithm [39]. In addition, we utilize the *Scikit-learn* library to preprocess and manage our data. *Scikit-learn* is a python module that performs various machine learning operations. Additionally, we develop novel methods from scratch to address more recent challenges not covered by the aforementioned libraries.

3.2 Dataset

By using the imblearn library we create a two dimensional imbalanced dataset in order to test our methods.

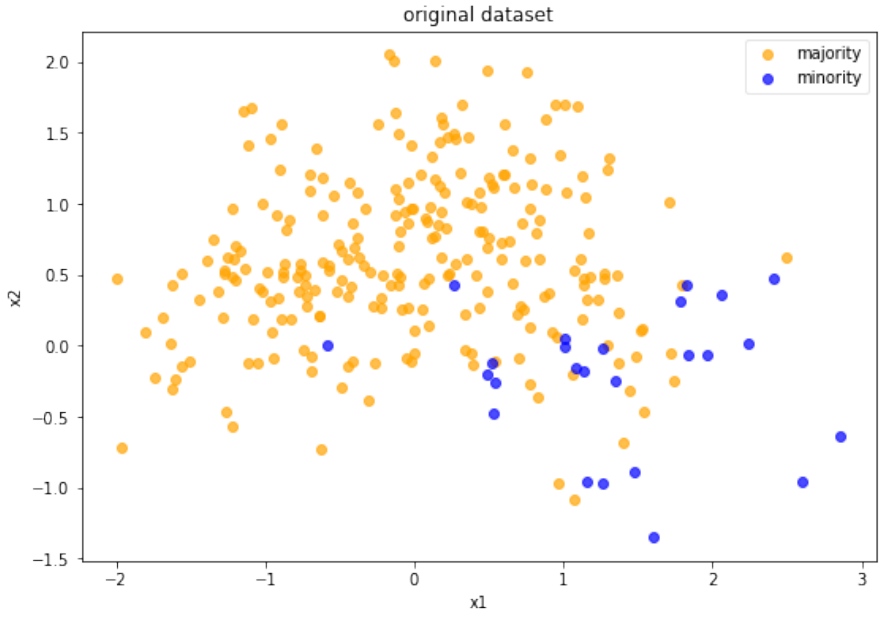

Figure 3.1: Distribution of the imbalanced dataset

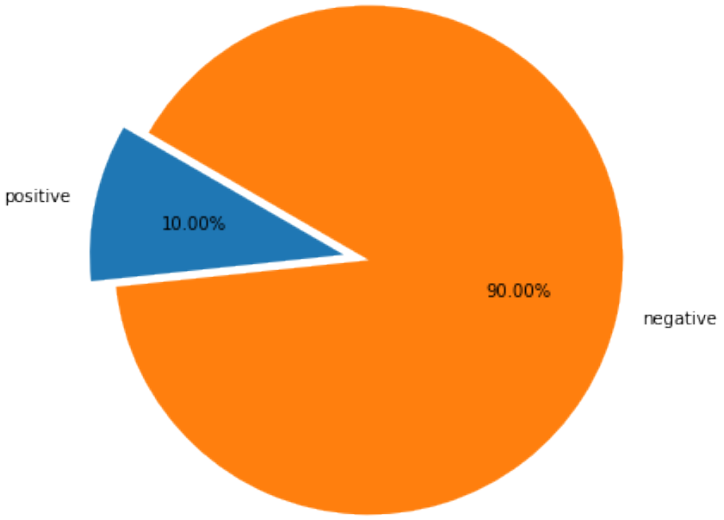

Figure 3.2: Class imbalance

Figure 3.1 depicts the sample distribution within our dataset, whereas Figure 3.2 show the class imbalance.

3.3 SWIM - Sampling with the majority class

3.3.1 Method Description

SWIM [31] is a technique for addressing severe class imbalance through oversampling. The primary goal of the SWIM algorithm is to position synthetic minority samples on the density contours that align with the original minority samples in relation to the majority class. To accomplish this, the Mahalanobis distance is computed. The Mahalanobis distance measures the distance between a sample and the centroid of a sample group. The concept behind SWIM is to position synthetic minority samples at an equivalent distance from the centroid of the majority class as the original minority samples.

The formula for Mahalanobis distance is as follows:

$$D_M(x) = (x - \mu)^T \Sigma^{-1} (x - \mu) \tag{3.1}$$

Here, Σ represents the estimated covariance matrix between all features, and μ denotes the mean of all samples. The Mahalanobis distance measures the distance in a correlated space. If the space is uncorrelated, meaning there is no covariance between features, the Mahalanobis distance simplifies to the Euclidean distance, which is a more straightforward calculation.

Pseudo code 3.1 shows the construction of the SWIM method. In line 6, we verify the rank of the majority samples matrix to determine the number of linearly independent (i.e., uncorrelated) columns. If the rank is lower than the total number of columns, we select the indices of all linearly dependent columns. We identify linearly dependent columns by utilizing QR decomposition, which expresses a matrix as the product of an orthogonal vector and an upper triangular matrix. If the upper triangular matrix contains zero values, it signifies linear dependency. In such cases, we remove the linearly dependent features and generate new samples within the subspace of the original data.

```
1         A = majority samples
2         B = minority samples
3         rank = matrix_rank(A)
4         dim = n_columns(A)
5
6         if rank < dim:
7             # perform QR decomposition
8             independent_columns = get_independent_columns(A)
9             A = A[independent_columns]
10            B = B[independent_columns]
11
12        mu_a = mean(A)
13        A_centered = (A - mu_a)
14        B_centered = (B - mu_a)
15
16        sigma = covariance_matrix(A_centered)
17
18        if determinant(sigma) <= 0:
19            # Covariance matrix is singular
20            return
21
22        sigma_scaled = square_root(inverse(sigma))
23
24        if sigma_scaled is complex == True:
25            # Complex numbers in sigma_scaled
26            return
27
28        # Whitening transformation
29        Bw = B_centered * sigma_scaled
30
31        for i in range(0, number of synthetic samples):
32            x = Bw[random sample] # random minority sample
33            s = [] #empty list
34            for feature in num_features:
35               uf =  x[feature] +  alpha * sd_feature
36               lf =  x[feature] -  alpha * sd_feature
37               s.append(random_float(lf, uf))
38
39            s_norm = s * (euclidean_norm(x)/euclidean_norm(s))
40            s_new = inv(sigma_scaled) *  s_norm # new sample
```

Pseudo Code 3.1: SWIM method

In lines 13 and 14, we center the majority and minority samples around the mean of the majority. This centering aids in calculating distances between the mean of the majority class and the minority samples.

From lines 16 to 29, we perform a whitening transformation on the minority class. The whitening transformation modifies the data to eliminate covariance between features. In the whitened space, we can utilize the Euclidean distance to measure distances between samples. It's important to note that this approach may not be suitable for all datasets. In some cases, the covariance matrix can be singular, meaning it cannot be inverted. Additionally, after taking the square root of the inverted covariance matrix, complex numbers may arise in the scaled covariance matrix. In such instances, the transformation is halted, and the original dataset is returned.

Between lines 31 and 40, we generate synthetic minority samples. Firstly, a random minority sample is selected. Then, lower and upper bounds for each feature of the synthetic samples are computed. The feature bounds are determined as follows:

$$l_f = x_f - \alpha * sd_f \tag{3.2}$$

$$u_f = x_f + \alpha * sd_f \tag{3.3}$$

Here, sd_f represents the standard deviation for each feature in the transformed minority set B_w, x_f is a feature of a random minority sample and α is a user-defined parameter that defines the spread around a density contour where the synthetic samples will be placed. A higher value of α allows the synthetic minority samples to be positioned in a wider area around the corresponding density contour. On lines 39 and 40, we transform the new sample back to its original space.

3.3.2 Result

We evaluate the performance of the SWIM method on our dataset to verify if it generates samples accurately.

Figure 3.3 displays our dataset after applying the SWIM oversampling method with a value of α set to 0.25. To provide a point of comparison, let's contrast this with the conventional SMOTE oversampling approach.

When comparing Figure 3.3 and Figure 3.6, the distinctions between these two methods become apparent. It is evident that SWIM positions synthetic minority samples along the contour lines surrounding the majority centroid. This results in more distinct decision regions for both the minority and majority classes. In contrast, SMOTE distributes new minority samples across the decision region of the majority class.

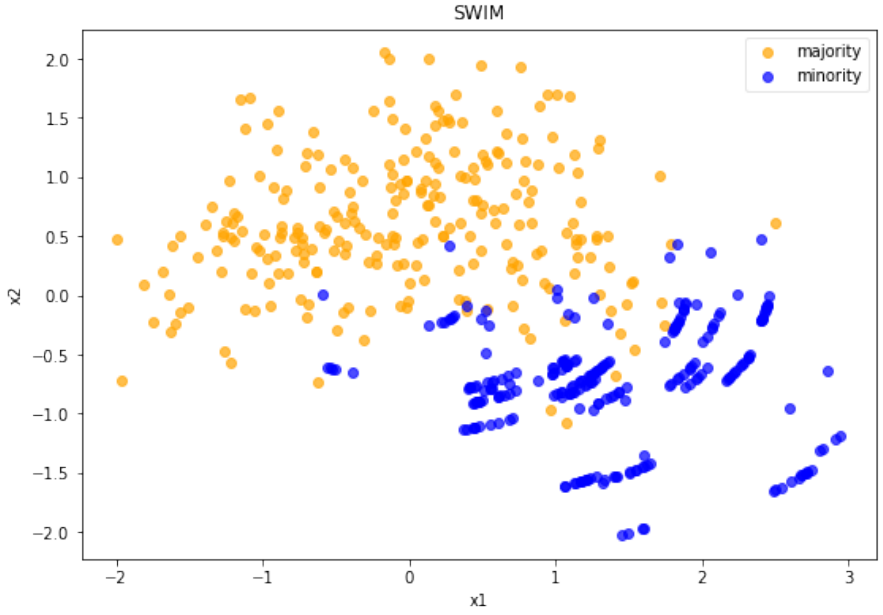

Figure 3.3: SWIM oversampling

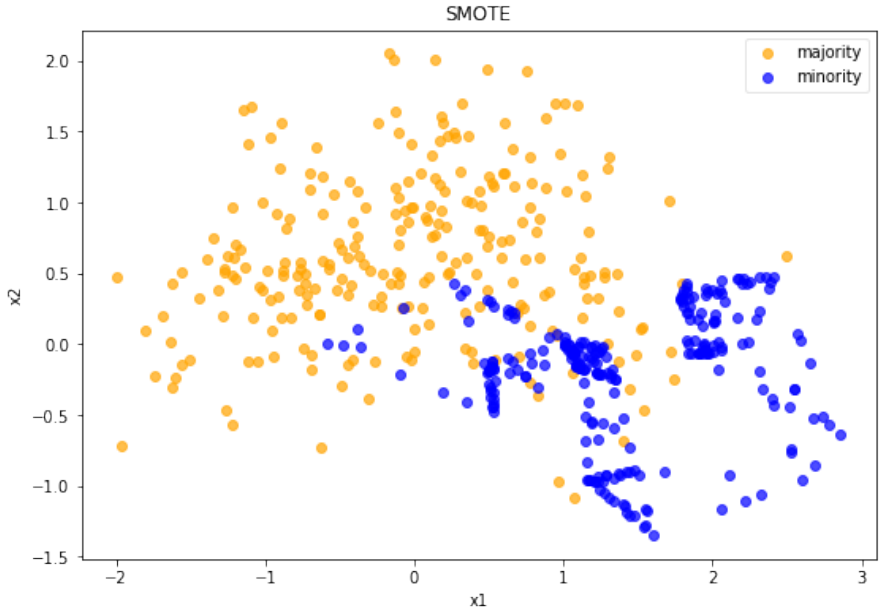

Figure 3.4: SMOTE oversampling

3.4 SMOTEFUNA

3.4.1 Method Description

SMOTEFUNA [30], an acronym for Synthetic Minority OverSampling Technique Based on Furthest Neighbour, differs from SMOTE in the way it generates minority samples. While SMOTE produces minority samples within a convex hull formed by existing minority samples, SMOTEFUNA creates an oversampling space in the shape of a hypercuboid.

This approach allows for diversified placement of synthetic samples, reducing the risk of overfitting the minority class.

```
1        majority = majority samples
2        minority = minority samples
3        synthetic_data = []
4        synthetic_labels = []
5        while i < num_samples:
6            s1 = minority[random sample]
7            s2 = get_furthest_neighbour(s1)
8            s_new = zeroes(num_features)
9            for j in range(num_features):
10               start = min(s1[j], s2[j])
11               stop = max(s1[j], s2[j])
12               s_new[j] = random(start, stop)
13           theta = get_closest_neighbour(minority, s_new)
14           beta = get_closest_neighbour(majority, s_new)
15           if theta <= beta:
16               synthetic_data.append(s_new)
17               synthetic_labels.append(minority_class)
18               i += 1
```

Pseudo Code 3.2: SMOTEFUNA

The pseudo code 3.2 outlines the operational procedure of the SMOTEFUNA function. Commencing from line 6, a random sample is selected from the minority class. Subsequently, the furthest neighbor of the chosen sample within the minority class is identified. By examining lines 10 and 11, the smallest and largest feature values between the two samples are determined, establishing the boundaries of the oversampling space. Following this, a new sample's feature values are randomly generated within the range defined by the largest and smallest feature values of the distanced samples. On lines 13 and 14, the distances of the synthetic sample to the nearest neighbor in both the minority and majority classes are calculated. If the nearest neighbor from the majority class is closer than the one from the minority class, the new minority sample is not added to the dataset. This measure is implemented to prevent oversampling within the majority region.

3.4.2 Result

We evaluate the performance of the SMOTEFUNA algorithm on our dataset to verify its accuracy and effectiveness.

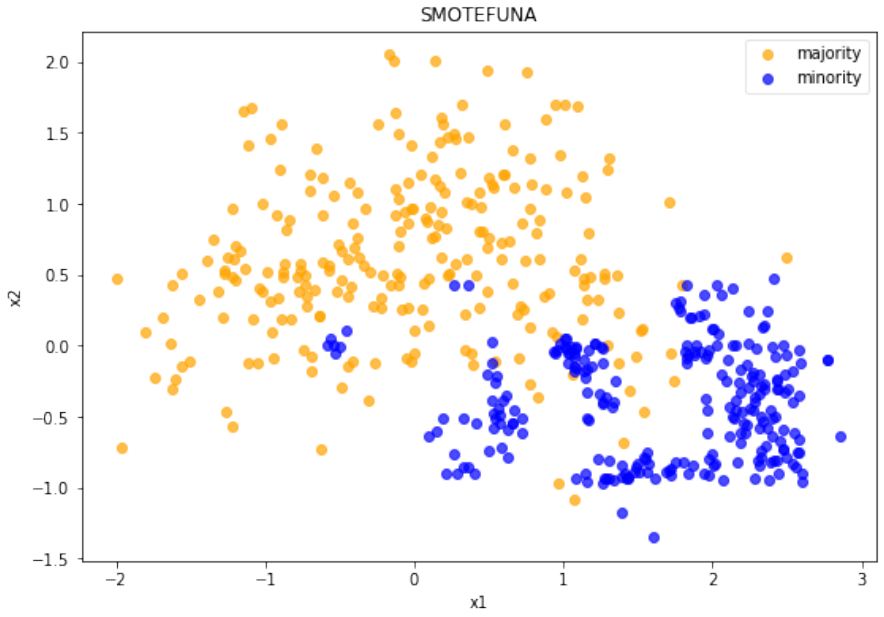

Figure 3.5: SMOTEFUNA oversampling

A scatter plot in Figure 3.5 is clearly demonstrating the visual representation of the outcome of the SMOTEFUNA oversampling technique. Upon comparing it with Figure 3.6, it becomes apparent that the synthetic minority samples are distributed across a wider range while still maintaining a separation from the majority region's boundaries.

3.5 MaMiPot

3.5.1 Method Description

MaMiPot, which stands for MAjority and MInority rePOsitioning Technique, presents an alternative approach to address the class imbalance issue. Instead of oversampling the minority samples, MaMiPot focuses on relocating both majority and minority samples to effectively extend the minority region. To determine which samples should undergo repositioning, a classifier is trained, and any samples falsely classified as either minority (positive) or majority (negative) are identified. False negative samples are subsequently shifted towards the minority centroid, while false positive samples are relocated towards the majority centroid. Additionally, the MaMiPot algorithm necessitates a metric to assess the impact of repositioning false negative and false positive samples on the overall classification performance.

```
        classifier.fit(data, labels)
        threshold = calculate_threshold()
        pred = classifier.predict(data, threshold)
        metric_old = metric(tr_labels, pred)
        # n = negative sample
        # p = postive sample

        stn = true negative samples # where pred = n and data = n
        stp = true positive samples # where pred = p and data = p
        sfn = false negative samples # where pred = n and data = p
        sfp = false positive samples # where pred = p and data = n

        mu_p = get_centroid(stp)
        mu_n = get_centroid(stn)

        iter = 0

        while iter < 3:
            iter += 1

            if stp.num_samples > gamma and sfn.num_samples > 0:
                for x in sfn:
                    x_new = alfa_p * mu_p + (1 - alfa_p) * x
                    sfn_new.append(x_new)

            if stn.num_samples > gamma and sfp.num_samples > 0:
                for y in sfp:
                    y_new = alfa_n * mu_n + (1 - self.alfa_n) * y
                    sfp_new.append(y_new)
            data_new = concatenate(stp, stn, sfp_new, sfn_new)
            classifier.fit(data_new, labels)
            pred = classifier.predict(data, threshold)
            metric_new = metric(tr_labels, pred)

            if metric_new > metric_old:
                metric_old = metric_new
                data = data_new
                iter = 0

        if beta > 0:
            data_res, labels_res = smote.sample(proportion=beta)
            return tr_data_res, tr_labels_res
        else:
            return data, labels
```

Pseudo Code 3.3: MaMiPot

Pseudo code 3.3 illustrates the process of MaMiPot. Initially, a classifier is trained using the given dataset. Subsequently, the optimal threshold is determined to maximize the decision boundary between the minority and majority classes. By default, classification algorithms use a threshold value of 0.5, where samples are classified as positive if the probability of being positive exceeds 0.5, and as negative if the probability is equal to or lower than 0.5. However, the default threshold may not always be suitable, and it is preferable to adjust it to maximize the performance metric used to evaluate the classifier. In their MaMiPot implementation, Ghaderi Zefrehi and Altınçay [3] utilize the F1-score as the performance metric, and the provided threshold that maximizes the F1-score is as follows:

$$\tau = \frac{1}{2}(\frac{P}{P+N} + 0.5) \tag{3.4}$$

In this equation, P denotes the count of positive samples, while N represents the count of negative samples.

Afterwards, based on the classification algorithm, the data is divided into four subsets: stn for true negative samples, stp for true positive samples, sfp for false positive samples, and sfn for false negative samples. Centroids are then calculated for the true positive and true negative samples by computing the mean values. The false positive and false negative samples are subsequently moved towards their respective centroids. The number of true positives and true negatives is checked against a user-defined threshold value γ on lines 22 and 27. If the count of correctly classified samples is too low, it becomes difficult to calculate a meaningful centroid for them. The formulas for moving the samples are provided in lines 24 and 29, where α_p and α_n represent repositioning factors for false negative and false positive samples, respectively. These repositioning factors indicate the proportion by which the samples should be moved towards their centroids.

Next, a new dataset is created consisting of the true positive and negative samples, along with the repositioned false negative and positive samples. The classifier is trained on this new dataset, and a new metric is calculated. If the new metric surpasses the old metric, the original data is replaced with the new data. This process is repeated until the new metric fails to exceed the old metric for three consecutive iterations.

Following the repositioning step, we have the option to oversample the minority region. As shown in lines 40 to 44, we can employ SMOTE or one of its variations for the oversampling process. The parameter β which is determined by the user, dictates the ratio of synthetic minority samples relative to the number of majority samples.

3.5.2 Result

We evaluate the performance of the MaMiPot approach on our dataset. We utilize the logistic regression classifier and the F1-score metric to carry out the MaMiPot method.

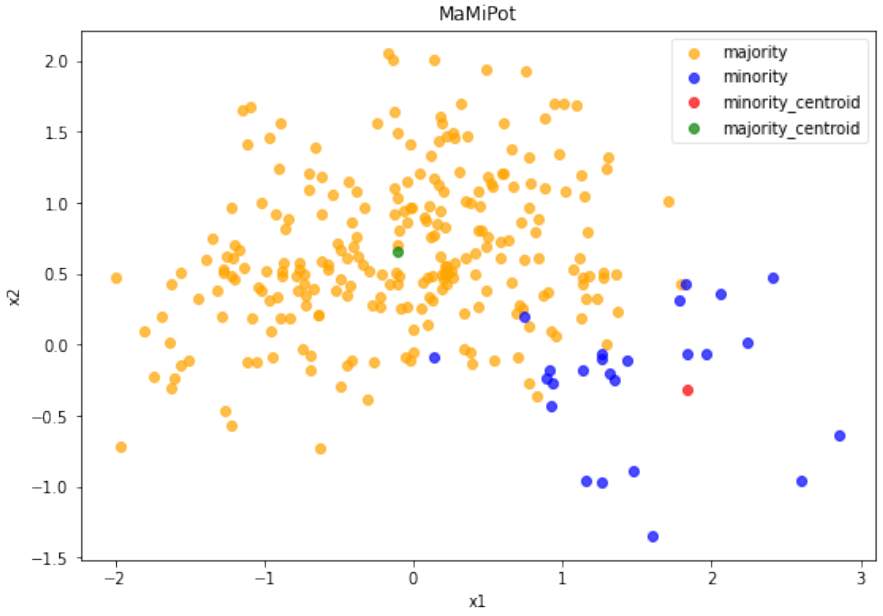

Figure 3.6: MaMiPot

It is evident that the minority and majority samples have reduced overlap compared to their previous state. The majority samples have been repositioned at a rate of 0.9, while the minority samples have been repositioned at a rate of 0.1. The repositioning of the minority samples has been kept relatively conservative to maintain the distribution of the minority class.

Next we check how MaMiPot with combination with SMOTE works.

MaMiPot in combination with SMOTE demonstrates the expected performance. The oversampling process employs a β parameter of 0.25. It is observed that SMOTE generates synthetic samples that overlap with the majority class. Hence, it might be beneficial to combine MaMiPot with a technique that takes into account both the majority and minority regions more effectively.

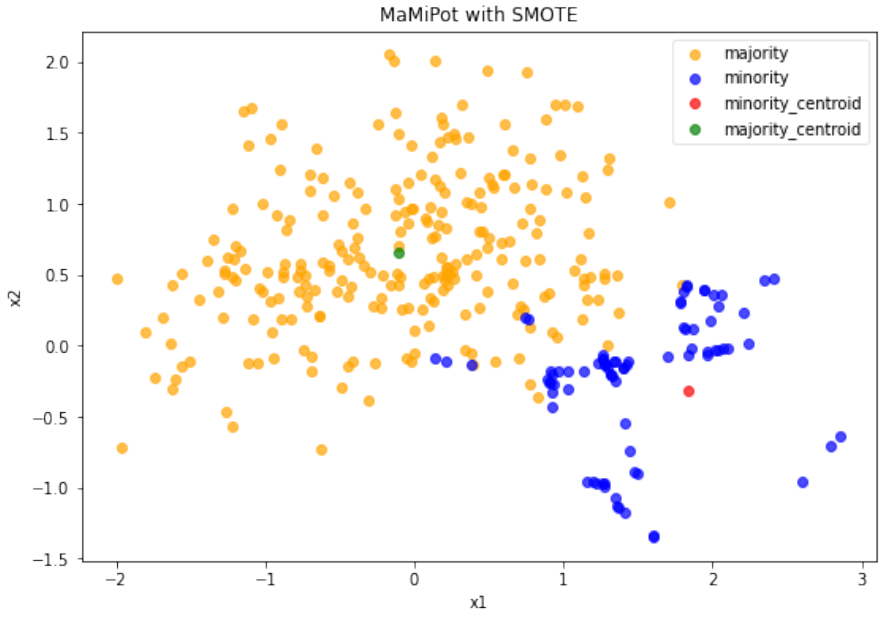

Figure 3.7: MaMiPot with SMOTE

3.6 K-means MaMiPot

3.6.1 Method Description

K-means MaMiPot, a modified version of MaMiPot developed for this thesis, incorporates k-means clustering into the process. This approach involves dividing the majority and minority classes into multiple clusters and subsequently relocating the false negative and false positive samples towards their nearest cluster. The objective is to distribute the repositioned samples across different locations rather than concentrating them in a single area. By implementing this approach, there is a possibility of enhancing the classification outcomes as it would mitigate the risk of the classifier overfitting the data.

Pseudo code 3.4 provides an implementation overview of KMeans-MaMiPot. In essence, this method shares similarities with plain MaMiPot, but there is a notable distinction in how centroids are calculated. Specifically, on lines 13 to 17, two k-means clustering algorithms are trained—one on the true positive samples and the other on the true negative samples. Users can define the values of p_clusters and n_clusters to specify the number of clusters for the positive and negative classes, respectively. False negative samples are assigned to clusters of true positive samples, while false positive samples are assigned to clusters of true negative samples. Subsequently, on lines 25 to 35, the false negative and false positive samples are repositioned towards the centroids of their respective clusters.

```
    classifier.fit(data, labels)
    threshold = calculate_threshold()
    pred = classifier.predict(data, threshold)
    metric_old = metric(tr_labels, pred)
    # n = negative sample
    # p = postive sample

    stn = true negative samples # where pred = n and data = n
    stp = true positive samples # where pred = p and data = p
    sfn = false negative samples # where pred = n and data = p
    sfp = false positive samples # where pred = p and data = n

    kmeans_tp = KMeans(num_clusters=p_clusters).fit(stp)
    clusters_fn = kmeans_tp.predict(sfn)

    kmeans_tn = KMeans(num_clusters=n_clusters).fit(stn)
    clusters_fp = kmeans_tn.predict(sfp)

    iter = 0
    while iter < 3:
        iter += 1
        if stp.shape[0] > (gamma + p_clusters) and sfn.num_samples > 0:
            for index, x in sfn:
                mu_p = kmeans_tp.cluster_centers[clusters_fn[index]]
                x_new = alfa_p * mu_p + (1 - alfa_p) * x
                sfn_new.append(x_new)

        if stn.shape[0] > (gamma + n_clusters) and sfp.num_samples > 0:
            for inde, y in sfp:
                mu_n = kmeans_tn.cluster_centers[clusters_fp[index]]
                y_new = alfa_n * mu_n + (1 - self.alfa_n) * y
                sfp_new.append(y_new)
```

Pseudo Code 3.4: KM-MaMiPot

3.6.2 Result

We evaluate the performance of KMeans-MaMiPot on our dataset using a logistic regression classifier and the F1 metric. For this evaluation, we configure the majority class to have three clusters and the minority class to have a single cluster.

The outcome depicted in Figure 3.8 exhibits similarities to the outcome of plain MaMiPot. The primary distinction between the regular MaMiPot and KMeans-MaMiPot becomes apparent when we observe that instead of one centroid, we now have two centroids for

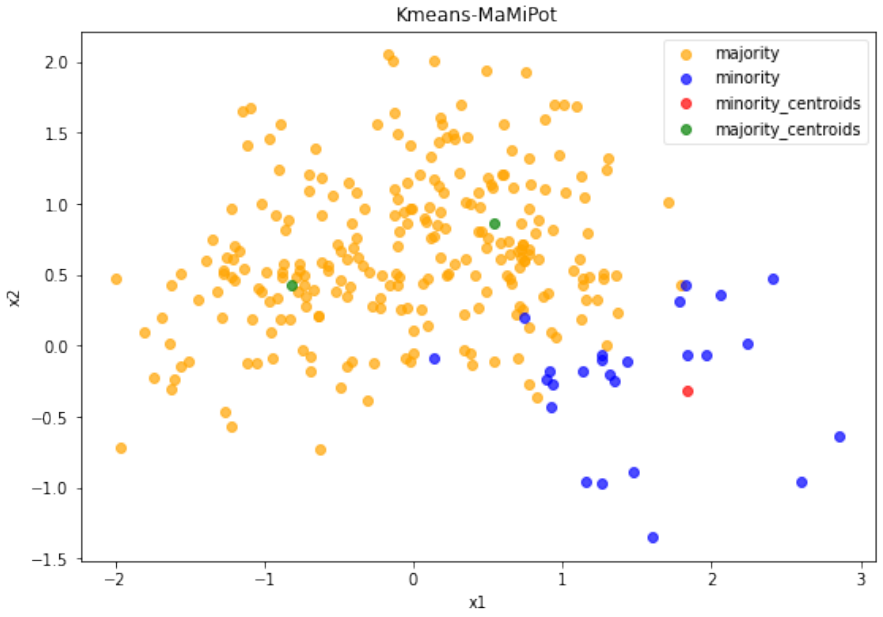

Figure 3.8: KMeans-MaMiPot

the majority class. Consequently, this leads to a broader distribution of majority samples across the majority region.

3.7 GAN

GAN [4], which stands for Generative Adversarial Network, is composed of two neural networks: a generator and a discriminator. The generator's role is to generate synthetic samples that closely resemble real samples, while the discriminator's task is to distinguish between real and synthetic samples. During training, the discriminator learns to improve its ability to discriminate between genuine and fake data, while the generator learns to produce more realistic data until both networks reach a state of equilibrium.

While GANs are commonly used for generating synthetic image or auditory data, in this thesis, we aim to apply GANs for oversampling smaller tabular datasets. Our implementation of GAN is based on the CIGAN approach proposed by Huang and Ma [40]. To implement the GAN, we utilize the *TensorFlow* and *Keras* libraries in *Python*, which provide robust support for machine learning and deep neural networks.

Before we commence the implementation of our GAN, it is essential to preprocess our data appropriately. The primary distinction lies in the fact that, instead of standardizing the data, we utilize a quantile transformer to ensure the data adheres to a uniform distribution. This is performed to enable the generation of synthetic samples in datasets with varying distributions. Subsequently, we employ the MinMaxScaler to scale the data,

ensuring that all values fall within the range of -1 to 1. The rationale behind these transformations will become evident as we delve into the construction of the generator.

```
1    kernel_initializer = initializers.random_normal(stddev=0.01)
2
3    generator = keras.models.Sequential()
4    self.generator.add(keras.layers.Dense(128, input_shape=input_size,
     kernel_initializer=kernel_initializer))
5    generator.add(keras.layers.LeakyReLU(alpha=0.7))
6    generator.add(keras.layers.BatchNormalization(momentum=0.9))
7    discriminator.add(keras.layers.Dropout(0.2))
8    generator.add(keras.layers.Dense(256))
9    generator.add(keras.layers.LeakyReLU(alpha=0.7))
10   generator.add(keras.layers.BatchNormalization(momentum=0.9))
11   discriminator.add(keras.layers.Dropout(0.2))
12   generator.add(keras.layers.Dense(512))
13   generator.add(keras.layers.LeakyReLU(alpha=0.7))
14   generator.add(keras.layers.BatchNormalization(momentum=0.9))
15   discriminator.add(keras.layers.Dropout(0.2))
16   generator.add(keras.layers.Dense(sample_size, activation='tanh'))
```

Pseudo Code 3.5: GAN generator

Pseudo code 3.5 outlines the construction of the generator. To begin, we establish the initial weights of the generator by generating random numbers from a normal distribution. The generator is defined as a sequential model, signifying that the layers are connected in a sequential manner. It consists of an input layer, three hidden layers, and an output layer. The input layer receives randomly generated noise values, which are sampled from a uniform distribution. Each hidden layer incorporates a *LeakyReLU* activation function, which determines the output of each node. *LeakyReLU* is a commonly used activation function. Additionally, the output of each hidden layer is normalized, enabling the model to converge more rapidly with a higher learning rate. Each hidden layer is also equipped with a dropout layer. The purpose of the dropout layer is to randomly disregard a portion of the layer's outputs, thereby mitigating the risk of overfitting. Each layer is specified as a dense layer, meaning that every node in a given layer is connected to all nodes in the subsequent layer. The output layer employs a *tanh* activation function, which produces values within the range of -1 to 1. This aligns with the scaling of our data during the preprocessing phase.

The discriminator is built in a similar manner to the generator. Its input is derived from the output of the generator. Like the generator, the discriminator comprises three hidden layers that employ the *LeakyReLU* activation function. Each hidden layer is equipped with a dropout layer and a normalization layer. The output layer of the discriminator

employs the *sigmoid* activation function. The *sigmoid* function produces values ranging from 0 to 1. Consequently, the discriminator outputs the probability of the generated sample being genuine or not.

The generator and discriminator are combined into a single GAN neural network. The binary crossentropy function is utilized to compute the training loss, and the weights of the GAN are updated using the Adam optimization algorithm in accordance with the loss.

Prior to training the GAN, the dataset is partitioned into several batches, with the number of batches determined by the user. The GAN will be trained sequentially on each batch of the dataset. The user also specifies the number of epochs for training. An epoch refers to a complete iteration of the neural network training on the entire dataset. The training loss is computed after each epoch.

During the training procedure, we initially focus on training the discriminator. Our first step is to generate a set of synthetic samples using the generator. These synthetic samples are created to match the size of the real sample batch. The fake samples are labeled as 0 (False), while the real samples are labeled as 1 (True). The discriminator is then trained on both batches of samples, allowing it to distinguish between real and synthetic samples..

Following that, we proceed with training the generator. It is important to emphasize that during the generator training, the discriminator is designated as non-trainable. This implies that the discriminator's weights remain unchanged throughout the generator's training process. The generator is trained using a batch of random noise samples, where all the labels are set to 1 to indicate that the generated samples are considered genuine. The discriminator will recognize that a significant portion of the samples are not authentic and will penalize these predictions. Consequently, this will impact the way the generator's weights are updated.

As each epoch progresses, the generated samples should become increasingly comparable to real minority samples.

Once the network has completed its training, the generator should possess the ability to generate precise synthetic minority samples.

3.7.1 Result

We evaluate the effectiveness of the GAN model on our dataset, where we configure the number of epochs to 50 and the batch size to 5.

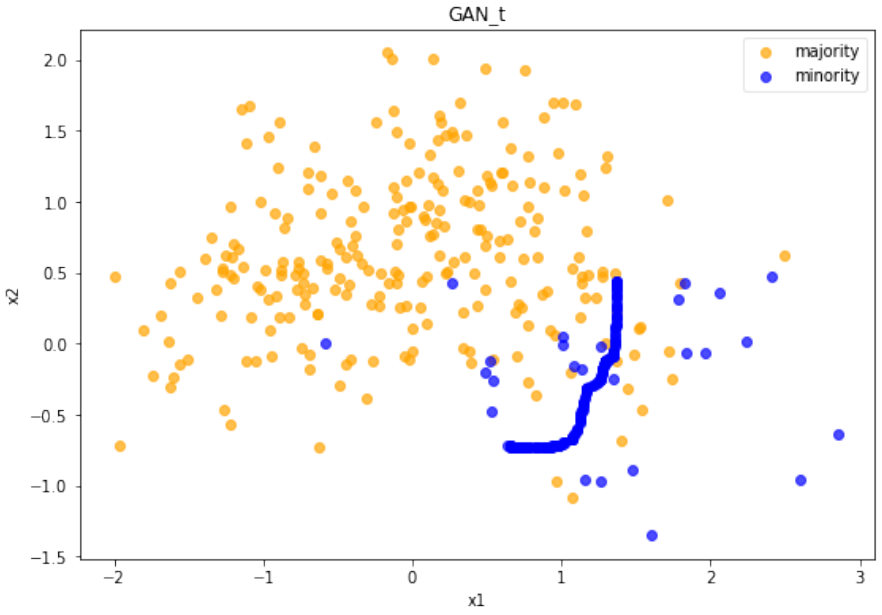

Figure 3.9: GAN

The depicted graph in Figure 3.9 illustrates the data sampling process of the GAN in a two-dimensional space. By employing the *tanh* function, the GAN generates synthetic minority samples that align in a manner resembling a hyperbolic line. This alignment has the potential to cause overfitting during the classification phase. Typically, GAN models are most effective when applied to large, high-dimensional datasets, making their utilization on a small two-dimensional dataset potentially excessive. Therefore, our implementation of the GAN is expected to yield more favorable results when employed on a dataset with a greater number of features.

Chapter 4

Experimental Evaluation

4.1 Overview

The main objective of this research is to evaluate and compare various sampling methods that have been previously published in the literature alongside newly proposed methods. Handling class imbalance leads to more precise and significant evaluation metrics that reflect the true performance of the model in each class. Traditional evaluation metrics such as accuracy can be misleading when dealing with imbalanced data since they can be heavily biased towards the majority class. Metrics such as precision, recall, and F1 score are better suited for evaluating models trained on imbalanced data.

The solution approach follows a workflow, illustrated in the figure 4.1:

Data Preprocessing: The initial stage involves preprocessing the data to ensure its suitability for subsequent steps. This includes tasks like label encoding, feature scaling, and splitting data for five-fold cross validation.

Resampling: In this step, we employ various techniques such as undersampling, oversampling, MaMiPot, and GAN to balance our dataset, addressing any class imbalance issues.

Model Training: Once the data is prepared, we proceed to train the models using diverse machine learning algorithms. These algorithms enable the models to learn patterns and relationships within the data.

Model Evaluation: After training the models, it is crucial to assess their performance. Evaluation metrics like F1-score, G-mean, and AUC are utilized to measure the models' effectiveness. To obtain reliable performance estimates, we employ three iterations of

5-fold cross validation, wherein we will fit and evaluate 50 models on the dataset during each iteration.

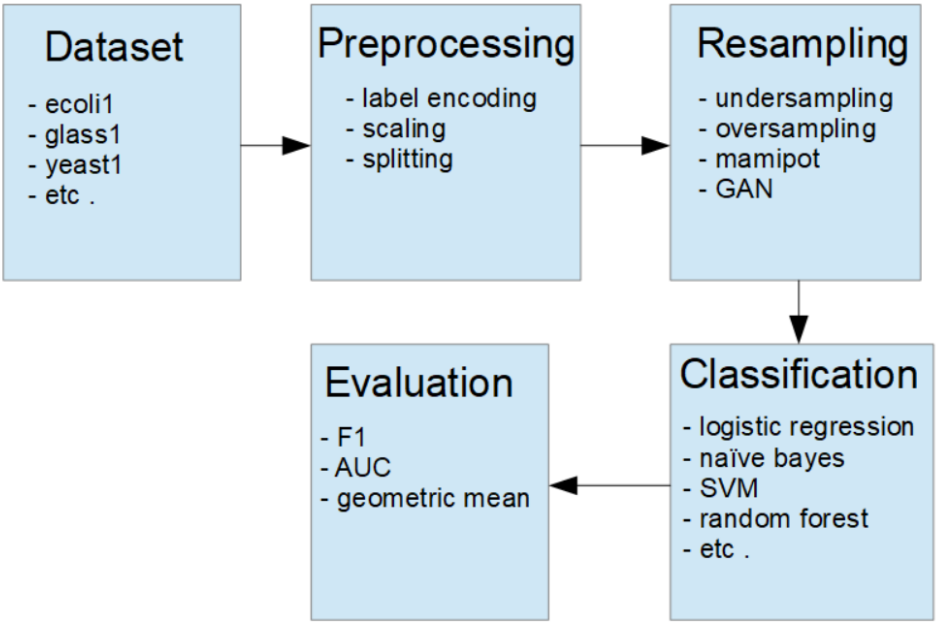

Figure 4.1: Workflow

4.2 Experimental Setup

Datasets

In our study, we utilized 25 binary datasets that exhibited imbalanced class distributions, with imbalance ratios ranging from 5.14 to 129.44, as described in table 4.1. These datasets were obtained from the KEEL repository [5], which can be accessed via URL http://sci2s.ugr.es/keel/imbalanced.php. Table 4.1 provides a summary of the datasets, including their names and number of instances, number of attributes and the imbalance ratio.

Table 4.1: A summary of binary datasets available in the KEEL repository [5]

Dataset	#Instances	#Attributes	Imbalance ratio
new-thyroid1	215	5	5.14
ecoli2	336	7	5.46
glass6	214	9	6.38
yeast3	1484	8	8.1
ecoli3	336	7	8.6
yeast-2_vs_4	514	8	9.08
yeast-0-3-5-9_vs_7-8	506	8	9.12
yeast-0-2-5-7-9_vs_3-6-8	1004	8	9.14
ecoli-0-1_vs_2-3-5	244	7	9.17
yeast-0-5-6-7-9_vs_4	528	8	9.35
vowel0	988	13	9.98
glass2	214	9	11.59
ecoli-0-1-4-6_vs_5	280	6	13
yeast-1_vs_7	259	7	14.3
glass4	214	9	15.47
ecoli4	336	7	15.8
page-blocks-1-3_vs_4	471	10	15.86
abalone9-18	731	8	16.4
glass-0-1-6_vs_5	184	9	19.44
glass5	214	9	22.79
yeast-2_vs_8	482	8	23.1
yeast4	1484	8	28.1
ecoli-0-1-3-7_vs_2-6	281	7	39.14
yeast6	1484	8	41.4
abalone19	4174	8	129.44

Preprocessing

During this specific stage, we begin by applying label encoding to the categorical target feature, transforming it into numerical representation for usage in model fitting and assessment. The *Scikit-learn* [41] module's *LabelEncoder* method, which transforms categorical variables into ordinal numbers, is used during the transformation process.

Subsequently, the data is normalized as the next step. Normalization is crucial to eliminate any biases caused by features being in disparate ranges, which could potentially confuse certain classifiers such as logistic regression.

We utilize the *StandardScaler* class from *Scikit-learn* to normalize our data. The *StandardScaler* removes the mean and scales the data to have a unit variance due to the substantial fluctuations observed in the feature variances. This is achieved by calculating the mean and standard deviation for each feature separately. The resulting standard score of a sample is calculated by subtracting the sample from the mean of its corresponding feature and dividing the result by the standard deviation [41, 42].

Next we divide the dataset into five segments of equal size. This is done for performing five-fold cross validation. It is essential to acknowledge that due to the dataset's imbalance, a stratified split is necessary. By employing a stratified split, we ensure that the class labels' proportions are equal in each split.

This preprocessing procedure is applied to all sampling methods, except for GAN, which, as described in section 3, employs its own distinct preprocessing scheme.

Resampling

In this stage of the research, we apply the methods discussed in Chapters 2 and 3 to address the class imbalance issue within the dataset. These methods are specifically designed to resample the data and create a more equitable distribution across the different classes, thus enabling effective classification modeling.

By applying these methods, we aim to achieve a balanced distribution of samples across the classes, paving the way for effective classification modeling. This stage of the research plays a critical role in mitigating the challenges posed by imbalanced datasets and improving the overall performance and generalization of the classification models.

Each resampling method is employed to modify the data in such a way that an equal number of minority and majority samples are present, except for the cases of MaMiPot and Kmeans-MaMiPot combined with SMOTE variants. In these instances, we oversample the minority class by 50%.

Classification

On the training dataset, we used five-fold stratified cross-validation to evaluate the performance of eight different machine learning models:

- logistic regression (LR) [6]
- K-Nearest Neighbors (KNN) [7]
- Support Vector Machine (SVM) [9]
- Naive Bayes (NB) [8]
- Decision Tree (DT) [10]
- Random Forest (RF) [11]
- AdaBoost [12]

- Multilayer Perceptron (MLP) [13]

All the classifiers mentioned above are obtained from the *Scikit-Learn* library. During training, we utilize the default parameters for each classifier. The process of five-fold cross-validation involves conducting training, testing, and evaluation five times, using a different data segment as the test set each time. The final metric is determined by calculating the average of the five metrics obtained from each fold. This approach ensures a more accurate and precise result.

Evaluation

To evaluate the effectiveness of the models, we computed each assessment metric mentioned in Chapter 2. Specifically, we used the *Scikit-learn* toolkit to compute F1, G-mean and AUC scores. We then created ranking tables for each of these three measures.

4.3 Experimental Results

In this section, we showcase the outcomes of our analysis. The average results across all datasets for each classifier and sampling method are displayed in Figures 4.2, 4.3 and 4.4. The transition of colors from green to red signifies a progression from high values to low values.

	LR	KNN	SVM	NB	DT	RF	AdaBoost	MLP
None	0.572883	0.639915	0.579638	0.386697	0.595341	0.345217	0.621284	0.624000
RUS	0.518032	0.543659	0.560111	0.403589	0.480647	0.505635	0.489071	0.481468
ROS	0.570028	0.628531	0.638743	0.368683	0.594616	0.603180	0.643089	0.632183
NearMiss	0.493280	0.551618	0.414050	0.273056	0.384307	0.448023	0.421459	0.371282
ENN	0.617123	0.669231	0.639361	0.454169	0.625654	0.432445	0.629779	0.657139
T-links	0.573781	0.656838	0.599056	0.409246	0.618348	0.347663	0.637997	0.631697
OSS	0.573781	0.656838	0.599056	0.409246	0.618348	0.342465	0.635098	0.631165
NCR	0.611195	0.680637	0.649411	0.457983	0.630272	0.415596	0.634616	0.663370
SMOTE	0.590437	0.611208	0.647042	0.403366	0.587673	0.606393	0.629929	0.632492
ADASYN	0.543611	0.591170	0.594775	0.388790	0.591607	0.573745	0.606727	0.618670
KM-SMOTE	0.614284	0.653193	0.625002	0.430907	0.589852	0.511340	0.619676	0.635850
BD-SMOTE1	0.615384	0.669391	0.658198	0.406600	0.594493	0.592700	0.648308	0.646109
BD-SMOTE2	0.571301	0.646394	0.627922	0.362493	0.593037	0.542805	0.643535	0.623198
DBSMOTE	0.590490	0.653496	0.631330	0.415074	0.595299	0.556714	0.635613	0.616652
CCR	0.556771	0.619832	0.635139	0.454491	0.632754	0.590748	0.630104	0.599808
MWMOTE	0.548932	0.616009	0.621163	0.368118	0.603183	0.579574	0.620937	0.626830
SMOTEFUNA	0.587919	0.679115	0.631549	0.391154	0.608000	0.573673	0.624773	0.635348
SWIM	0.601867	0.626124	0.653899	0.435712	0.605667	0.548849	0.640025	0.631007
SMOTE-TM	0.591901	0.615642	0.648738	0.404348	0.586632	0.616441	0.625028	0.634988
SMOTE-ENN	0.571224	0.639816	0.580163	0.386900	0.595273	0.357285	0.621834	0.623770
SMOTE-IPF	0.591782	0.617713	0.647310	0.405054	0.584142	0.613286	0.627335	0.621404
MaMiPot	0.604344	0.643016	0.586737	0.388914	0.595341	0.349426	0.495671	0.624000
MaMiPot + SMOTE	0.636164	0.644030	0.681274	0.393506	0.615418	0.646677	0.470775	0.630996
MaMiPot + ADASYN	0.581705	0.597113	0.618019	0.380158	0.591717	0.579282	0.476403	0.626203
MaMiPot + KM-SMOTE	0.627257	0.653483	0.620769	0.408563	0.590181	0.526973	0.478615	0.633402
MaMiPot + BD-SMOTE1	0.639022	0.677140	0.671942	0.383588	0.592318	0.571194	0.479375	0.638137
MaMiPot + BD-SMOTE2	0.610191	0.656436	0.639389	0.355878	0.598206	0.542266	0.489729	0.619389
MaMiPot + DBSMOTE	0.640381	0.654114	0.642309	0.415864	0.592594	0.548049	0.482706	0.630704
MaMiPot + CCR	0.615918	0.643481	0.660584	0.446487	0.632428	0.586832	0.494780	0.625259
MaMiPot + MWMOTE	0.613355	0.628752	0.634737	0.370690	0.610301	0.594246	0.478569	0.628930
MaMiPot + SM-TM	0.638225	0.645781	0.677885	0.398009	0.599878	0.653068	0.477856	0.623332
MaMiPot + SM-ENN	0.628496	0.631272	0.670875	0.433383	0.636600	0.655001	0.478475	0.639242
MaMiPot + SM-IPF	0.638723	0.646013	0.679049	0.398908	0.608920	0.644630	0.479125	0.616978
MaMiPot + SWIM	0.630978	0.646514	0.679817	0.435153	0.614928	0.519485	0.491452	0.607039
MaMiPot + SMOTEFUNA	0.612408	0.682809	0.640213	0.399158	0.605306	0.546522	0.454357	0.624959
KM-MaMiPot	0.604344	0.644391	0.589323	0.395538	0.595341	0.348170	0.502330	0.624000
KM-MaMiPot + SMOTE	0.639992	0.646593	0.687622	0.403147	0.600913	0.653286	0.479669	0.627292
KM-MaMiPot + ADASYN	0.580807	0.598250	0.614353	0.392442	0.586760	0.586976	0.479480	0.632425
KM-MaMiPot + KM-SMOTE	0.627335	0.658769	0.639021	0.403133	0.586603	0.526011	0.486552	0.635164
KM-MaMiPot + BD-SMOTE1	0.639973	0.678145	0.667846	0.393621	0.596049	0.587930	0.489534	0.634564
KM-MaMiPot + BD-SMOTE2	0.611466	0.647947	0.635418	0.357320	0.603793	0.544624	0.498293	0.606119
KM-MaMiPot + DBSMOTE	0.638224	0.653976	0.638062	0.414272	0.590701	0.555027	0.491875	0.622432
KM-MaMiPot + CCR	0.620508	0.647519	0.671951	0.426703	0.634180	0.574209	0.499470	0.619690
KM-MaMiPot + MWMOTE	0.610912	0.626645	0.635793	0.377148	0.613053	0.603203	0.494049	0.627633
KM-MaMiPot + SM-TM	0.634036	0.634824	0.692305	0.405734	0.603533	0.649547	0.481107	0.611022
KM-MaMiPot + SM-ENN	0.635000	0.629751	0.667637	0.416373	0.626064	0.658253	0.481174	0.657135
KM-MaMiPot + SM-IPF	0.637610	0.645118	0.680526	0.400905	0.610694	0.649500	0.484832	0.628668
KM-MaMiPot + SWIM	0.635649	0.652499	0.689397	0.430447	0.617201	0.511987	0.513936	0.623146
KM-MaMiPot + SMOTEFUNA	0.610668	0.678566	0.643964	0.399699	0.620083	0.557990	0.472555	0.627059
GAN	0.489638	0.624762	0.588276	0.397432	0.590392	0.329104	0.674448	0.633411

Figure 4.2: Avarage F1-score across all datasets

	LR	KNN	SVM	NB	DT	RF	AdaBoost	MLP
None	0.642970	0.699791	0.620525	0.569870	0.725314	0.373028	0.709828	0.753155
RUS	0.832947	0.839861	0.842849	0.625793	0.818460	0.835281	0.837051	0.822767
ROS	0.855210	0.835330	0.840817	0.562018	0.721250	0.836672	0.777060	0.755481
NearMiss	0.788878	0.801052	0.738603	0.568564	0.734289	0.765297	0.745441	0.695620
ENN	0.711526	0.748905	0.702526	0.632016	0.788010	0.474090	0.754543	0.832042
T-links	0.645552	0.719457	0.644087	0.581349	0.756659	0.377382	0.725342	0.768258
OSS	0.645552	0.719457	0.644087	0.581349	0.756659	0.370107	0.722795	0.768141
NCR	0.700555	0.758591	0.715270	0.634289	0.799846	0.454616	0.763427	0.825749
SMOTE	0.859009	0.850451	0.832860	0.624236	0.756748	0.843245	0.797458	0.776014
ADASYN	0.843136	0.850549	0.814445	0.593249	0.758259	0.834766	0.788203	0.759793
KM-SMOTE	0.719147	0.737252	0.681697	0.630811	0.727209	0.600127	0.712399	0.760330
BD-SMOTE1	0.827137	0.835377	0.794447	0.635449	0.730435	0.754573	0.779199	0.774469
BD-SMOTE2	0.823144	0.813635	0.787563	0.582233	0.738367	0.730961	0.785741	0.764484
DBSMOTE	0.801109	0.758734	0.746822	0.648088	0.721932	0.688739	0.736437	0.735594
CCR	0.855519	0.822329	0.849647	0.744275	0.788435	0.758533	0.734558	0.782126
MWMOTE	0.849438	0.835150	0.823708	0.603947	0.734251	0.809086	0.764515	0.753193
SMOTEFUNA	0.751475	0.808950	0.732585	0.606269	0.761456	0.707398	0.747441	0.775308
SWIM	0.853126	0.837726	0.835304	0.709828	0.744690	0.650820	0.733269	0.776997
SMOTE-TM	0.856424	0.847090	0.831538	0.623092	0.762354	0.849015	0.795116	0.776863
SMOTE-ENN	0.646332	0.700618	0.621835	0.567561	0.726042	0.387159	0.710767	0.753288
SMOTE-IPF	0.860771	0.852241	0.833252	0.620951	0.752283	0.851994	0.789421	0.769283
MaMiPot	0.689041	0.705019	0.630008	0.587426	0.725314	0.380335	0.732840	0.753155
MaMiPot + SMOTE	0.822551	0.841028	0.819030	0.611340	0.771782	0.765289	0.712215	0.764929
MaMiPot + ADASYN	0.831432	0.845227	0.808574	0.585078	0.753786	0.805354	0.718666	0.766004
MaMiPot + KM-SMOTE	0.739805	0.734195	0.673788	0.619957	0.726841	0.603215	0.717867	0.756835
MaMiPot + BD-SMOTE1	0.801545	0.829547	0.781279	0.617029	0.725623	0.678259	0.719595	0.767379
MaMiPot + BD-SMOTE2	0.800759	0.813909	0.774727	0.580803	0.742528	0.658144	0.728691	0.759313
MaMiPot + DBSMOTE	0.803264	0.757881	0.741430	0.664031	0.722992	0.630693	0.717409	0.751704
MaMiPot + CCR	0.817330	0.805027	0.810145	0.725344	0.783522	0.679739	0.746760	0.799589
MaMiPot + MWMOTE	0.823676	0.828174	0.795135	0.608744	0.750749	0.718777	0.714556	0.759599
MaMiPot + SM-TM	0.822436	0.845327	0.804903	0.620257	0.753705	0.773683	0.716343	0.763031
MaMiPot + SM-ENN	0.831186	0.846165	0.823693	0.650557	0.828471	0.795997	0.719127	0.816229
MaMiPot + SM-IPF	0.823730	0.848943	0.807042	0.618382	0.772500	0.762295	0.714146	0.749054
MaMiPot + SWIM	0.805396	0.831312	0.806888	0.705618	0.752806	0.590621	0.728911	0.751259
MaMiPot + SMOTEFUNA	0.741293	0.797975	0.727352	0.617291	0.760355	0.636195	0.700959	0.769044
KM-MaMiPot	0.689041	0.707689	0.633549	0.610514	0.725314	0.376834	0.741152	0.753155
KM-MaMiPot + SMOTE	0.823318	0.845460	0.813077	0.629136	0.755016	0.771983	0.726226	0.763925
KM-MaMiPot + ADASYN	0.829732	0.849233	0.811967	0.607618	0.756603	0.821086	0.721835	0.773683
KM-MaMiPot + KM-SMOTE	0.740886	0.743292	0.697664	0.621463	0.723659	0.601696	0.729111	0.755127
KM-MaMiPot + BD-SMOTE1	0.801277	0.825001	0.779328	0.628918	0.731796	0.700544	0.728288	0.760378
KM-MaMiPot + BD-SMOTE2	0.802765	0.803617	0.769830	0.592628	0.747407	0.660796	0.742814	0.744996
KM-MaMiPot + DBSMOTE	0.802494	0.759400	0.743946	0.662077	0.722580	0.638758	0.729331	0.743772
KM-MaMiPot + CCR	0.818650	0.809068	0.814462	0.718287	0.787444	0.670941	0.748468	0.800181
KM-MaMiPot + MWMOTE	0.823661	0.830460	0.800283	0.615702	0.756482	0.735539	0.732028	0.752583
KM-MaMiPot + SM-TM	0.816801	0.836446	0.822670	0.636093	0.766113	0.771322	0.724213	0.754263
KM-MaMiPot + SM-ENN	0.833765	0.845542	0.823915	0.640922	0.809971	0.800581	0.724829	0.829256
KM-MaMiPot + SM-IPF	0.821159	0.839975	0.810039	0.633382	0.769492	0.770862	0.729094	0.767556
KM-MaMiPot + SWIM	0.809576	0.831753	0.813526	0.692393	0.753919	0.586920	0.752942	0.770255
KM-MaMiPot + SMOTEFUNA	0.745197	0.801588	0.728157	0.624318	0.767149	0.648040	0.714064	0.768536
GAN	0.697319	0.782483	0.704384	0.595998	0.755109	0.404012	0.784216	0.769515

Figure 4.3: Avarage G-mean across all datasets

	LR	KNN	SVM	NB	DT	RF	AdaBoost	MLP
None	0.750635	0.792567	0.757556	0.699592	0.790414	0.641895	0.792201	0.804595
RUS	0.842460	0.850786	0.853461	0.733123	0.824678	0.844131	0.841739	0.829775
ROS	0.864790	0.855291	0.859614	0.701388	0.784805	0.855439	0.826409	0.810122
NearMiss	0.801043	0.816790	0.754408	0.637967	0.754095	0.779174	0.762405	0.723086
ENN	0.787966	0.815122	0.795764	0.730903	0.824557	0.685905	0.811443	0.854603
T-links	0.753955	0.801996	0.768100	0.706641	0.806581	0.644201	0.801655	0.814199
OSS	0.753955	0.801996	0.768100	0.706641	0.806581	0.643537	0.799481	0.814069
NCR	0.783367	0.822182	0.801657	0.733700	0.831576	0.676852	0.817530	0.853379
SMOTE	0.869093	0.862911	0.853031	0.721439	0.806019	0.858598	0.834342	0.817893
ADASYN	0.853066	0.862152	0.837619	0.710866	0.805302	0.849236	0.825713	0.809539
KM-SMOTE	0.799663	0.817755	0.791312	0.720401	0.787984	0.739869	0.794213	0.810143
BD-SMOTE1	0.850474	0.855313	0.829945	0.723758	0.793847	0.807649	0.825631	0.820109
BD-SMOTE2	0.838574	0.842549	0.827683	0.699612	0.799324	0.793790	0.826556	0.812972
DBSMOTE	0.830008	0.816486	0.804816	0.729282	0.788957	0.777107	0.806011	0.801220
CCR	0.865580	0.848799	0.867126	0.793352	0.831140	0.819495	0.802895	0.823936
MWMOTE	0.858144	0.853751	0.844474	0.713207	0.794685	0.836004	0.819903	0.809112
SMOTEFUNA	0.804153	0.842081	0.798707	0.707652	0.809434	0.781411	0.805967	0.824015
SWIM	0.864707	0.854392	0.854730	0.768183	0.802353	0.765506	0.799739	0.820595
SMOTE-TM	0.866559	0.862789	0.851882	0.721258	0.809190	0.862449	0.831482	0.819691
SMOTE-ENN	0.753677	0.793274	0.758786	0.697701	0.791018	0.652860	0.793119	0.804719
SMOTE-IPF	0.870394	0.864505	0.853593	0.721067	0.804037	0.864201	0.829761	0.813872
MaMiPot	0.773794	0.796128	0.761381	0.706679	0.790414	0.643577	0.783542	0.804595
MaMiPot + SMOTE	0.845816	0.859833	0.845684	0.714007	0.814497	0.820316	0.769693	0.814815
MaMiPot + ADASYN	0.848805	0.860683	0.836284	0.706041	0.805864	0.836565	0.772546	0.814557
MaMiPot + KM-SMOTE	0.805398	0.814783	0.783304	0.713094	0.788084	0.739906	0.771925	0.806949
MaMiPot + BD-SMOTE1	0.835218	0.854087	0.826595	0.714091	0.791044	0.774981	0.775543	0.815159
MaMiPot + BD-SMOTE2	0.829287	0.842920	0.822751	0.700211	0.800601	0.763277	0.780620	0.810357
MaMiPot + DBSMOTE	0.837005	0.817422	0.805204	0.739092	0.786674	0.752058	0.774015	0.804984
MaMiPot + CCR	0.845751	0.841807	0.846886	0.783860	0.826965	0.781767	0.792350	0.833972
MaMiPot + MWMOTE	0.847951	0.849447	0.832692	0.715312	0.801176	0.795762	0.771997	0.811300
MaMiPot + SM-TM	0.845581	0.863434	0.840915	0.717504	0.805791	0.822494	0.772714	0.810706
MaMiPot + SM-ENN	0.851518	0.862843	0.852713	0.735622	0.849793	0.835960	0.775350	0.844539
MaMiPot + SM-IPF	0.847282	0.864716	0.842736	0.719427	0.817242	0.817737	0.770850	0.807334
MaMiPot + SWIM	0.834879	0.851882	0.838170	0.762622	0.806198	0.739129	0.781271	0.808497
MaMiPot + SMOTEFUNA	0.800156	0.841051	0.797734	0.718040	0.810062	0.748091	0.759677	0.817304
KM-MaMiPot	0.773794	0.798508	0.762578	0.717323	0.790414	0.643064	0.786494	0.804595
KM-MaMiPot + SMOTE	0.846944	0.861814	0.846889	0.723527	0.804240	0.823989	0.773762	0.810550
KM-MaMiPot + ADASYN	0.847376	0.861301	0.836773	0.719405	0.801792	0.847156	0.773247	0.817563
KM-MaMiPot + KM-SMOTE	0.807255	0.819546	0.794443	0.716589	0.785193	0.738819	0.780272	0.805896
KM-MaMiPot + BD-SMOTE1	0.834871	0.852711	0.826871	0.719288	0.792780	0.781528	0.778673	0.811831
KM-MaMiPot + BD-SMOTE2	0.830795	0.838143	0.820777	0.705769	0.803241	0.763260	0.789278	0.802447
KM-MaMiPot + DBSMOTE	0.836136	0.819056	0.804570	0.739113	0.787150	0.754919	0.779625	0.803739
KM-MaMiPot + CCR	0.844875	0.843951	0.850372	0.777307	0.828688	0.775788	0.795330	0.832854
KM-MaMiPot + MWMOTE	0.845634	0.851358	0.833766	0.721198	0.809637	0.803087	0.780139	0.806032
KM-MaMiPot + SM-TM	0.843359	0.856223	0.849500	0.727507	0.811527	0.822001	0.775405	0.808785
KM-MaMiPot + SM-ENN	0.853819	0.860529	0.850700	0.728168	0.837736	0.838866	0.779028	0.854448
KM-MaMiPot + SM-IPF	0.844883	0.859488	0.845809	0.725342	0.813463	0.822886	0.780437	0.811766
KM-MaMiPot + SWIM	0.838055	0.853071	0.847251	0.758185	0.808467	0.733983	0.797198	0.816538
KM-MaMiPot + SMOTEFUNA	0.802415	0.841079	0.799016	0.719030	0.813921	0.756256	0.766457	0.815085
GAN	0.762337	0.817871	0.772301	0.700282	0.793119	0.644172	0.817653	0.823167

Figure 4.4: Avarage AUC score across all datasets

4.4 Result Analysis

This section focuses on comparing the performance of the proposed methods with other sampling techniques. The evaluation is conducted on ensembles consisting of eight classifiers.

Figure 4.2 illustrates the results obtained from the cross-validation process, with the F1-score evaluation metric reflecting the average performance across all datasets. The "None" model represents the implementation of the classifier using the original unbalanced data. It becomes apparent that oversampling methods generally yield higher F1-scores compared to the undersampling methods.

Based on the F1-score metric, MaMiPot demonstrates good performance overall, except for the AdaBoost classifier. It can be concluded that the AdaBoost classifier does not perform well when there are changes in the position of the samples.

The majority of classifiers exhibit poor performance with the GAN, except for AdaBoost, which achieves the highest score. The Near-Miss algorithm appears to have the poorest performance. This is likely due to Near-Miss removing an excessive number of samples, which subsequently leads to the classifier lacking essential information about the distribution of the data.

Table 4.2 shows the ranking of different resampling methods based on their average F1-score across all datasets and classifiers. The ranking is determined by aggregating the scores of all classifiers. The results indicate that resampling a dataset can have a substantial effect on its F1-score. It demonstrates that out of the 50 resampling methods evaluated, 44 of them successfully improved the F1-score. Among these methods, the top-rated one is the Version 1 of Borderline-SMOTE.

Table 4.2: Sampling methods ranked by F1 score

method	rank	method	rank
BD-SMOTE1	1	MaMiPot + SWIM	26
MaMiPot + SM-ENN	2	BD-SMOTE2	27
KM-MaMiPot + SM-ENN	3	KM-MaMiPot + SMOTEFUNA	28
SWIM	4	MaMiPot + DBSMOTE	29
NCR	5	KM-MaMiPot + DBSMOTE	30
KM-MaMiPot + SMOTE	6	KM-MaMiPot + MWMOTE	31
KM-MaMiPot + SM-IPF	7	MWMOTE	32
SMOTEFUNA	8	MaMiPot + SMOTEFUNA	33
ENN	9	KM-MaMiPot + KM-SMOTE	34
SMOTE-TM	10	MaMiPot + MWMOTE	35
CCR	11	MaMiPot + KM-SMOTE	36
MaMiPot + SMOTE	12	MaMiPot + BD-SMOTE2	37
MaMiPot + SM-TM	13	ADASYN	38
MaMiPot + SM-IPF	14	KM-MaMiPot + BD-SMOTE2	39
KM-MaMiPot + SM-TM	15	T-links	40
SMOTE	16	KM-MaMiPot + ADASYN	41
SMOTE-IPF	17	OSS	42
MaMiPot + CCR	18	MaMiPot + ADASYN	43
DBSMOTE	19	SMOTE-ENN	44
KM-MaMiPot + CCR	20	None	45
KM-MaMiPot + BD-SMOTE1	21	GAN	46
KM-SMOTE	22	KM-MaMiPot	47
ROS	23	MaMiPot	48
KM-MaMiPot + SWIM	24	RUS	49
MaMiPot + BD-SMOTE1	25	NearMiss	50

Figure 4.3 illustrates the average results for geometric mean across all datasets.

The G-mean scores show that LR, KNN, and SVM classifiers perform considerably better than NB, DT, and AdaBoost classifiers across all resampling techniques. This can have a number of causes, including:

- The fact that LR, KNN, and SVM classifiers are capable of handling both linear and non-linear correlations in the data makes them particularly suitable for a variety of classification problems. In contrast, the performance of the naive bayes, decision tree, and AdaBoost may be hindered by their inability to recognize intricate patterns and relationships in the data.
- Gaussian naive bayes makes an assumption that all features are independent and that data in each feature is normally distributed, which might not be true for this application. Although decision tree and decision tree based classifiers can handle high-dimensional feature spaces or complicated relationships between features, it still might be challenging.

Table 4.3: Sampling methods ranked by G-mean

method	**rank**	**method**	**rank**
RUS	1	MaMiPot + MWMOTE	26
SMOTE-TM	2	MaMiPot + SWIM	27
SMOTE	3	KM-MaMiPot + BD-SMOTE1	28
CCR	4	MaMiPot + BD-SMOTE1	29
SMOTE-IPF	5	SMOTEFUNA	30
MaMiPot + SM-ENN	6	KM-MaMiPot + BD-SMOTE2	31
KM-MaMiPot + SM-ENN	7	MaMiPot + BD-SMOTE2	32
ADASYN	8	NearMiss	33
ROS	9	DBSMOTE	34
MWMOTE	10	KM-MaMiPot + DBSMOTE	35
KM-MaMiPot + ADASYN	11	KM-MaMiPot + SMOTEFUNA	36
KM-MaMiPot + CCR	12	MaMiPot + DBSMOTE	37
MaMiPot + CCR	13	MaMiPot + SMOTEFUNA	38
SWIM	14	NCR	39
KM-MaMiPot + SM-IPF	15	ENN	40
BD-SMOTE1	16	KM-MaMiPot + KM-SMOTE	41
KM-MaMiPot + SMOTE	17	MaMiPot + KM-SMOTE	42
KM-MaMiPot + SM-TM	18	KM-SMOTE	43
MaMiPot + ADASYN	19	GAN	44
MaMiPot + SMOTE	20	KM-MaMiPot	45
MaMiPot + SM-TM	21	T-links	46
MaMiPot + SM-IPF	22	OSS	47
KM-MaMiPot + MWMOTE	23	MaMiPot	48
BD-SMOTE2	24	SMOTE-ENN	49
KM-MaMiPot + SWIM	25	None	50

Table 4.3 shows the ranking of different resampling methods based on their average geometric mean across all datasets and classifiers.

Among the evaluated techniques, Random Under-sampling (RUS) achieved the highest G-mean score, indicating its effectiveness in addressing the class imbalance issue. It is followed by the SMOTE-Tomek Links (SMOTE-TM) and the regular SMOTE.

Figure 4.3 illustrates the average results for the AUC metric across all datasets.

Although the AUC scores generally align with the G-mean results, the differences observed for each classifier are less pronounced in comparison to the G-mean results. Based on the ranking of projected probabilities, AUC primarily analyzes the model's capacity to distinguish between positive and negative cases. The classifier's capacity to concurrently optimize sensitivity and specificity is also captured by G-mean. Classifiers that successfully discriminate between good and bad occurrences by appropriately rating them tend to perform consistently across both criteria.

Table 4.4: Sampling methods ranked by AUC score

method	rank	method	rank
CCR	1	MaMiPot + MWMOTE	26
SMOTE-TM	2	MaMiPot + SWIM	27
SMOTE	3	KM-MaMiPot + BD-SMOTE1	28
SMOTE-IPF	4	MaMiPot + BD-SMOTE1	29
RUS	5	SMOTEFUNA	30
MaMiPot + SM-ENN	6	DBSMOTE	31
KM-MaMiPot + SM-ENN	7	KM-MaMiPot + BD-SMOTE2	32
ROS	8	MaMiPot + BD-SMOTE2	33
ADASYN	9	KM-MaMiPot + DBSMOTE	34
MaMiPot + CCR	10	NCR	35
KM-MaMiPot + CCR	11	MaMiPot + DBSMOTE	36
SWIM	12	KM-MaMiPot + SMOTEFUNA	37
MWMOTE	13	ENN	38
BD-SMOTE1	14	MaMiPot + SMOTEFUNA	39
KM-MaMiPot + ADASYN	15	KM-SMOTE	40
KM-MaMiPot + SM-IPF	16	KM-MaMiPot + KM-SMOTE	41
KM-MaMiPot + SM-TM	17	MaMiPot + KM-SMOTE	42
KM-MaMiPot + SMOTE	18	GAN	43
MaMiPot + SM-IPF	19	T-links	44
MaMiPot + SMOTE	20	OSS	45
MaMiPot + ADASYN	21	KM-MaMiPot	46
MaMiPot + SM-TM	22	MaMiPot	47
KM-MaMiPot + SWIM	23	SMOTE-ENN	48
KM-MaMiPot + MWMOTE	24	None	49
BD-SMOTE2	25	NearMiss	50

Despite their similarities, AUC and G-mean offer different viewpoints on classifier performance. While G-mean integrates sensitivity and specificity at a certain threshold, AUC is a threshold-independent statistic that focuses on ranking predicted probabilities.

As a result, comparing the two measures can offer a more thorough insight of classifier performance.

Table 4.4 shows the ranking of different resampling methods based on their average AUC score across all datasets and classifiers. CCR achieved highest AUC score Following it are the SMOTE-Tomek Links (SMOTE-TM) and regular SMOTE.

Chapter 5

Conclusions

This thesis has explored the challenges and techniques associated with imbalanced classification in machine learning. We have investigated various resampling methods, such as oversampling, undersampling, MaMiPot and Generative Adversarial Networks to mitigate the impact of class imbalance. We aimed to assess established and widely recognized methods in comparison to newer approaches.

We conducted experiments on a set of 25 imbalanced datasets with varying degrees of class imbalance to evaluate the performance of all methods. The evaluation was based on F1-score, geometric mean, and AUC score as the chosen metrics. These metrics were specifically selected because they are well-suited for assessing performance in the context of imbalanced data.

Table 5.1 displays the ranking of the top five methods according to their F1-scores. It is clear that both plain oversampling methods and the combination of MaMiPot with oversampling yield the most favorable outcomes.

Rank	**Method**
1	BD-SMOTE1
2	MaMiPot + SM-ENN
3	KM-MaMiPot + SM-ENN
4	SWIM
5	NCR

Table 5.1: Five higiest scored sampling methods based on F1-score

The top five sampling methods, ranked by the geometric mean, are presented in Table 5.2. The findings regarding the geometric mean are rather unexpected. Surprisingly, a straightforward technique such as random undersampling is rated as the most effective sampling method for maximizing the geometric mean.

Rank	Method
1	RUS
2	SMOTE-TM
3	SMOTE
4	CCR
5	SMOTE-IPF

Table 5.2: Five highest scored sampling methods based on geometric mean

Table 5.3 presents the top five sampling methods, ranked by the AUC score. Interestingly, the methods that achieve the highest scores for AUC are quite similar to those that achieve the highest scores for the geometric mean. This suggests that sampling methods which excel at maximizing the geometric mean also tend to be effective in maximizing the AUC score.

Rank	Method
1	CCR
2	SMOTE-TM
3	SMOTE
4	SMOTE-IPF
5	RUS

Table 5.3: Five highest scored sampling methods based on AUC score

Based on our experiments, it is evident that both oversampling and MaMiPot can significantly improve classifier performance in imbalanced datasets. However, our findings indicate that the Generative Adversarial Network (GAN) yielded disappointing results. One possible explanation for this outcome could be the small size of our datasets, which might have resulted in the insufficient convergence of the GAN due to the limited information contained in the small training batches.

5.1 Future Directions

Our future goals encompass the following:

- Expand the application of our methods to handle imbalanced problems involving multiple classes.
- Assess the impact of data size and dimensionality on the effectiveness of our techniques.
- Enhance the GAN approach to improve its performance in imbalanced classification scenarios.

List of Figures

List of Tables

Appendix A

Source Code

Source code is available at https://github.com/pio93/mst_code.git

Bibliography

[1] Yunqian Ma and Haibo He. Imbalanced learning: foundations, algorithms, and applications. 2013.

[2] Sukarna Barua, Md Monirul Islam, Xin Yao, and Kazuyuki Murase. Mwmote–majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Transactions on knowledge and data engineering*, 26(2):405–425, 2012.

[3] Hossein Ghaderi Zefrehi and Hakan Altınçay. Mamipot: a paradigm shift for the classification of imbalanced data. *Journal of Intelligent Information Systems*, pages 1–26, 2022.

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014. doi: 10.1145/3422622.

[5] J Derrac, S Garcia, L Sanchez, and F Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *J. Mult. Valued Logic Soft Comput*, 17, 2015.

[6] David Roxbee Cox. *Analysis of binary data*. Routledge, 2018.

[7] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[8] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.

[9] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[10] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[11] D Richard Cutler, Thomas C Edwards Jr, Karen H Beard, Adele Cutler, Kyle T Hess, Jacob Gibson, and Joshua J Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007.

[12] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[14] Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126, pages 1–7. ICML, 2003.

[15] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

[16] Ivan Tomek. Two modifications of cnn. 1976.

[17] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, volume 97, page 179. Citeseer, 1997.

[18] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In *Artificial Intelligence in Medicine: 8th Conference on Artificial Intelligence in Medicine in Europe, AIME 2001 Cascais, Portugal, July 1–4, 2001, Proceedings 8*, pages 63–66. Springer, 2001.

[19] Joonho Gong and Hyunjoong Kim. Rhsboost: Improving classification performance in imbalance data. *Computational Statistics & Data Analysis*, 111:1–13, 2017.

[20] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[21] Sukarna Barua, Md. Monirul Islam, Xin Yao, and Kazuyuki Murase. Mwmote–majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Transactions on Knowledge and Data Engineering*, 26(2):405–425, 2014. doi: 10.1109/TKDE.2012.232.

[22] Halimu Chongomweru and Asem Kasem. A novel ensemble method for classification in imbalanced datasets using split balancing technique based on instance hardness (sbal_ih). *Neural Computing and Applications*, 33(17):11233–11254, 2021.

[23] Yuxi Xie, Min Qiu, Haibo Zhang, Lizhi Peng, and Zhenxiang Chen. Gaussian distribution based oversampling for imbalanced data classification. *IEEE*

Transactions on Knowledge and Data Engineering, 34(2):667–679, 2022. doi: 10.1109/TKDE.2020.2985965.

[24] Krystyna Napierala and Jerzy Stefanowski. Types of minority class examples and their influence on learning classifiers from imbalanced data. *Journal of Intelligent Information Systems*, 46:563–597, 2016.

[25] Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1322–1328, 2008. doi: 10.1109/IJCNN.2008.4633969.

[26] Georgios Douzas, Fernando Bacao, and Felix Last. Improving imbalanced learning through a heuristic oversampling method based on k-means and smote. *Information Sciences*, 465:1–20, 2018.

[27] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In *Advances in Intelligent Computing: International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I 1*, pages 878–887. Springer, 2005.

[28] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Dbsmote: density-based synthetic minority over-sampling technique. *Applied Intelligence*, 36:664–684, 2012.

[29] Michał Koziarski and Michał Woźniak. Ccr: A combined cleaning and resampling algorithm for imbalanced data classification. *International Journal of Applied Mathematics and Computer Science*, 27(4):727–736, 2017.

[30] Ahmad S Tarawneh, Ahmad BA Hassanat, Khalid Almohammadi, Dmitry Chetverikov, and Colin Bellinger. Smotefuna: Synthetic minority over-sampling technique based on furthest neighbour algorithm. *IEEE Access*, 8:59069–59082, 2020.

[31] Shiven Sharma, Colin Bellinger, Bartosz Krawczyk, Osmar Zaiane, and Nathalie Japkowicz. Synthetic oversampling with the majority class: A new perspective on handling extreme imbalance. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 447–456, 2018. doi: 10.1109/ICDM.2018.00060.

[32] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[33] José A Sáez, Julián Luengo, Jerzy Stefanowski, and Francisco Herrera. Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, 291:184–203, 2015.

[34] Guillem Collell, Drazen Prelec, and Kaustubh R Patil. A simple plug-in bagging ensemble based on threshold-moving for classifying binary and multiclass imbalanced data. *Neurocomputing*, 275:330–340, 2018.

[35] CA Hartanto, S Kurniawan, D Arianto, and AM Arymurthy. Dcgan-generated synthetic images effect on white blood cell classification. In *IOP Conference Series: Materials Science and Engineering*, volume 1077, page 012033. IOP Publishing, 2021.

[36] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[37] Roghayeh Soleymani, Eric Granger, and Giorgio Fumera. F-measure curves: A tool to visualize classifier performance under imbalance. *Pattern Recognition*, 100:107146, 2020.

[38] Jason Brownlee. *Imbalanced classification with Python: better metrics, balance skewed classes, cost-sensitive learning*. Machine Learning Mastery, 2020.

[39] György Kovács. smote-variants: a python implementation of 85 minority oversampling techniques. *Neurocomputing*, 366:352–354, 2019. doi: 10.1016/j.neucom.2019.06.100. (IF-2019=4.07).

[40] Yuxiao Huang and Yan Ma. Cigan: A python package for handling class imbalance using generative adversarial networks. 08 2022. doi: 10.48550/arXiv.2208.02931.

[41] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[42] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.