E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Studying the Perturbation-based Oversampling Technique for Imbalanced Classification Problems

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumna: Mehsun Ihtiyan Saalim

Director: José Antonio Sanz Delgado

Pamplona, 6 de septiembre de 2023

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ABSTRACT

This study aims to implement, analyze, and compare the effectiveness of a novel technique known as Perturbation-Based Oversampling (POS). This technique is designed to address class imbalance in machine learning by augmenting the minority class instances by strategically perturbing features using a hyperparameter '*p*'. Two additional variations, namely POS 1.0 and POS 2.0, have been proposed as extensions of the original POS approach. Detailed experiments have been conducted across diverse datasets, presenting a comprehensive performance evaluation in terms of precision when compared to a selection of established methods designed to tackle unbalanced classification challenges.

Key words: classification, imbalanced problems, oversampling, features, imbalanced ratio, perturbation

Index

1. INTRODUCTION

In today's fast-paced world, where technology keeps evolving at an incredible rate, machine learning has become a catalyst for innovation. The impressive ability of algorithms to learn from data and progressively enhance their performance, has powered all sorts of applications in different fields. Within this arena, supervised learning stands out as a fundamental concept.

Supervised learning, a subset of machine learning, holds high importance in understanding how machines can learn to make informed decisions. A canonical supervised learning problem is classification, whose goal is to enable algorithms to classify input data into different groups based on their inherent traits. This involves training the algorithm on a labeled dataset, where the outcomes are already known, serving as the ground truth for model training. This process allows algorithm to discover patterns and connections.

The process involves selecting an appropriate classification algorithm, training the model on training data and evaluating it on test data. In the training stage the adjustment of the model's parameters is important since it enables it to capture patterns and relationships between features and labels. After training the model, it is evaluated measuring its performance on test data in terms of accuracy and other metrics. This approach is crucial in applications that require accurate categorization of data into predefined classes, such as medical diagnoses and text analysis.

Regarding the dataset's characteristics, the importance cannot be overstated. The characteristics of the dataset highly influence the model's capacity to comprehend and generalize patterns. That is why when the dataset presents certain anomalies within the training data, addressing these challenges becomes indispensable for ensuring accurate and reliable machine learning outcomes. In the context of this specific work, we address situations where input set of data lacks a balanced distribution of classes. Specifically, when dealing with a classification problem, there are many more instances of same classes than others and standard classifiers get overwhelmed by the large classes and ignore the small ones. This is known as class imbalance problem [1].

One of the most common real-world scenarios involving imbalanced data is in fraud detection [2]. In credit card transactions, the chances of having fraudulent transactions are meagre, compared to legitimate transactions. It leads to the imbalanced proportions of labels in the dataset. This creates an imbalance between the positive class (fraudulent transactions) and the negative class (legitimate transactions. The model will have no problem identifying a legitimate transaction but will have trouble identifying a fraudulent one since the model treats the minority class as noise or outliers. While the majority class may dominate in terms of instances, it is often the minority class that holds vital information or represents critical scenarios. In the case of fraud detection, correctly identifying fraudulent transactions can save financial institutions and individuals from significant losses. It becomes evident that that it is vital to obtain a balanced prediction performance that accurately captures the details and importance of the underrepresented class.

In recent years, various techniques have been proposed to combat this challenge, which can be grouped into two types: algorithm-level and data-level approaches [3]. Algorithm-level approaches handle class imbalanced data by changing the mechanism of existing classifiers, while data-level approaches manipulate the data distribution to rebalance the number of instances between two classes [4]. In this case, we will focus on data-level approaches. One common approach involves resampling the dataset to balance the class distribution. Oversampling the minority class or undersampling the majority class are techniques that artificially adjust the proportions of instances, allowing the model to give equal importance to both classes during training.

Recent research findings show the effectiveness of generating instances that follow similar distributions as the minority class [5] or spread along the embedded probability density [6] for improving the predictive performance. However, these methodologies necessitate the fitting of density functions or employing manifold learning techniques to transform the input space into an induced manifold. These approaches are computationally intensive and might find challenges when dealing with small amount of minority instances in an imbalanced classification problem.

In this context, a new approach emerges from researchers at the Department of Electrical and Computer Engineering, University of Alberta, Canada, aiming to design a straightforward yet effective oversampling technique based on perturbations. The Perturbation-based Oversampling (POS) method balances datasets by generating new instances by perturbing the existing ones. This new technique introduces a new aspect: the incorporation of a hyperparameter 'p' to regulate the perturbation's variance. This innovation provides a level of flexibility that allows the algorithm to accommodate data with varying characteristics.

The aim of this study is to investigate and confirm the effectiveness of this new proposal in terms of achieving strong performance and solving the issue of imbalanced datasets. Moreover, we will compare the results obtained using the POS technique to those achieved by other established models from the literature and current advanced methods. In addition, new approaches to this method will be proposed since certain ambiguities and shortcomings in the method’s description were identified analyzing this study. Therefore, we have chosen to reinterpret and modify specific elements based on individual judgment.

2. PRELIMINARIES

2.1. Classification Problems

2.1.1. Classifiers

A supervised classification problem is about building a model that can effectively classify the input data that has been provided based on its specific features. The dataset utilized for training this model consists of input instances paired with corresponding known outputs, forming the training dataset. Once the classifier is trained on this dataset, its performance is assessed on a new and unseen dataset known as test dataset. This new dataset is made up of unfamiliar input instances with known classes. Achieving accurate classification is not about achieving high accuracy on the training data, the model must generalize well to this new unseen data.

Classifiers are algorithms or models that learn from labeled training data to make predictions or decisions about the class labels of new, unseen data points. There are various types of classifiers, each with its own strengths, weaknesses, and suitable use cases.

2.1.1.1. K-Nearest Neighbors (KNN)

KNN is a simple yet effective machine learning algorithm that classifies data points based on the majority class of their k-nearest neighbors in the feature space. It's a lazy learning algorithm that doesn't explicitly build a model during training.

During the training phase, KNN stores the feature vectors and corresponding class labels of the training dataset. No explicit model is built during this phase; the algorithm simply memorizes the training data.

In the prediction phase, when a new data point needs to be classified, KNN identifies the 'k' nearest neighbors of the new point in the feature space. The value of 'k' is a hyperparameter that you need to specify. It's the number of neighbors used to determine the class label of the new point. The distance metric is used to calculate the distance between data points in the feature space. The class label of the new point is then determined by the majority class among its 'k' nearest neighbors. In other words, the class label with the highest frequency among these neighbors is chosen.

KNN is a simple to understand and implement algorithm, however, the algorithm is sensitive to scaling of features.

Scaling data is an important preprocessing step to ensure that all features contribute equally to the model's performance. Scaling helps bring the features to a similar scale, preventing some features from dominating others due to their larger magnitudes. In this study **Min-Max Scaling (Normalization)** has been used. This type of scaling transforms features to a range between 0 and 1.

2.1.1.2. Decision Trees

Decision trees are a popular machine learning approach for both classification and regression tasks. They create a tree-like model where internal nodes represent feature tests, branches correspond to test outcomes, and leaf nodes signify predicted labels or values. The algorithm learns to make decisions by recursively partitioning the dataset based on features, aiming to maximize the purity of subsets for classification tasks or minimize the variance for regression tasks. This makes decision trees easy to visualize and understand, allowing for transparent insights into the decision-making process.

One notable decision tree algorithm is CART (Classification and Regression Trees). This type of tree employs builds binary trees where each internal node represents a feature test, the branches correspond to the outcomes of that test (typically a binary decision), and the leaf nodes contain the predicted class labels for classification or values for regression. What sets CART apart is its ability to automatically determine the best feature and threshold for splitting the data at each node. It uses impurity measures like the Gini index or mean squared error to assess the quality of these splits. CART's flexibility and simplicity make it a powerful tool for various machine learning applications.

CART is particularly well-suited for classification problems, where it strives to partition the data in a way that maximizes the purity of each resulting subset. The purity measure is typically the Gini impurity, which quantifies the likelihood of misclassifying a randomly chosen sample from the subset. By iteratively partitioning the data based on the best feature and threshold, CART constructs a tree that optimizes the classification accuracy.

2.1.2. Model Validation Methodologies

The concept of generalization is highly important in classification. Overfitting occurs when the model adjusts perfectly to the training data, getting overly complex, capturing noise, but fails to predict accurately unknown data. On the contrary, underfitting is given when a model is too simplistic to capture the patterns and performs poorly on both training and new data. The aim is to find the balance between these extremes to obtain robust and accurate classification results. This pursuit of balance is where model validation methodologies play a pivotal role.

Before implementing a model in real-world applications, it is essential to ensure their reliability and performance. Model validation methodologies are techniques used to assess how a trained model might perform on new data. They provide a way to validate whether a model's predictions generalize accurately beyond the data it was trained on. Ideally, a substantial dataset for training the model should be complemented by a separate and big dataset for evaluating its performance. However, this is usually not the case, and the datasets size is limited. The following techniques are used to obtain the training and validation (test) sets.

2.1.2.1. Hold-out

Hold-out validation involves splitting the dataset into two parts: a training and a validation set. The model is trained on the training set and evaluated on the validation set. A common split ratio usually is 70-30 or 80-20 between training and validation sets. Moreover, to maintain data consistency, the proportion between classes must remain constant, therefore, stratification is applied.

Stratification is a technique that ensures that the distribution of the classes in the obtained sets is similar to the distribution of the classes in the original set.

2.1.2.2. Leave-one-out

Leave-one-out is a technique where each data point is used as the test set while the remaining data is used for training. The process is repeated until it has used every point in the dataset as test instance. This technique can be computationally expensive for larger datasets.

2.1.2.3. K-Fold Cross Validation

This study employs the K-fold cross validation, which consists of dividing the dataset into k fold or sets. The model is trained using k-1 sets of data and the remaining set is used for validation. This process is repeated k times using a different fold as validation each time. For the study we have chosen k = 5, a common value in this type of validation. The obtained k sets must have similar size. Although the k sets have been derived from the same original dataset, each of the datasets should be distinct, with no data in common. In other words, each instance from the original dataset must only be part of one set. The k partitions are randomly generated in each iteration. To maintain data consistency, stratification is applied the maintain the proportion between classes constants.

Figure 1 shows graphically the process of 5-fold cross validation.

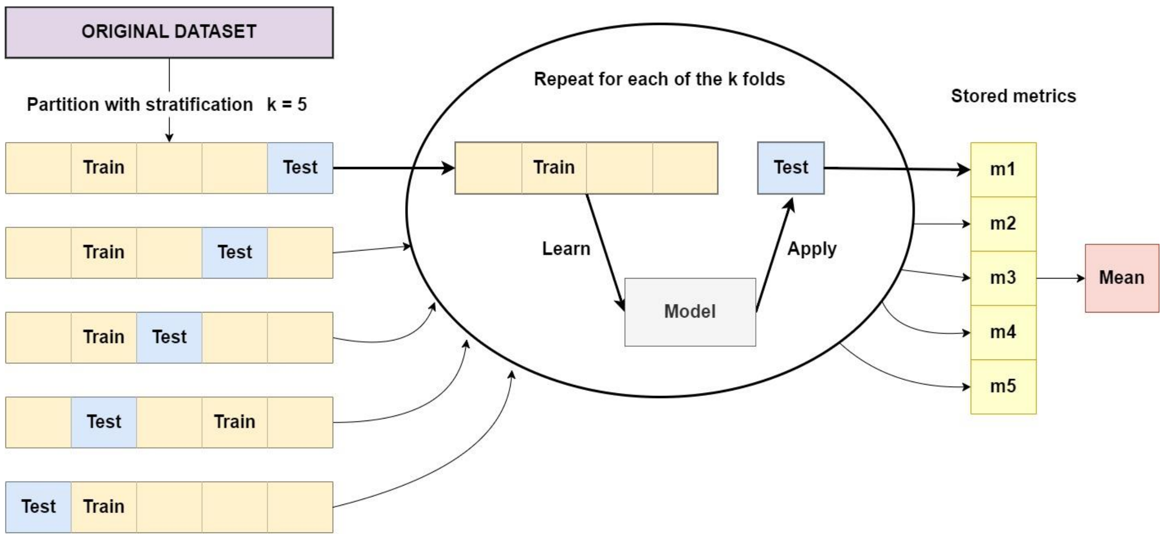


Figure 1: Diagram representing the 5- fold cross validation technique.

As shown in the diagram illustrating the 5-fold cross-validation technique (*Figure 1*), following the model's training using the training dataset, we assess the model's performance on the test dataset. This evaluation is done using the evaluation metrics, which are further explained in section 2.1.2. The overall model evaluation is given by the average evaluation outcomes from each of the k partitions.

Using this technique, we ensure that the outcomes achieved are unliked to the data used to train the model, obtaining in this way a classifier capable of generalizing to new data.

2.1.3. Model Evaluation Metrics

As stated before, once a model is trained it must be evaluated on a new unfamiliar dataset. The model's performance is evaluated using evaluation metrics. These tools assess the performance allowing us to understand how well the model's predictions align with the actual outcomes. Metrics provide a standardized way to compare different models, validate their suitability for real-world scenarios, and identify areas that may need improvement.

Depending on the task and nature of the data, there are various types of model evaluation metrics. Each metric is designed to capture different aspects of the model's performance.

Accuracy, precision, recall, and F1 score are widely employed for classification tasks. ROC (Receiver Operating Characteristic) curves and AUC (Area Under the Curve) are usually better when analyzing binary classifiers' performance. Additionally, AUCPR (Area Under the Precision-Recall Curve) is particularly useful for imbalanced datasets.

Given a **confusion matrix** (*Table 1),* which is a tabulation of actual and predicted class labels, we can obtain different metrics for the evaluation of the classifier.

		Prediction	
		Positive	**Negative**
Real	**Positive**	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Table 1: Confusion matrix

- **True Positive (TP):** The instances where the model correctly predicted the positive class when the actual class was positive.

- **True Negative (TN):** The instances where the model correctly predicted the negative class when the actual class was negative.

- **False Positive (FP):** The instances where the model incorrectly predicted the positive class when the actual class was negative.

- **False Negative (FN):** The instances where the model predicted incorrectly the negative class when the actual class was positive.

The confusion matrix provides a more comprehensive view of a model's performance. From these numbers, various performance metrics can be calculated such as the *Accuracy, False Positive Rate*, *True Negative Rate, Precision and Recall*.

2.1.3.1. Accuracy

Accuracy (*Formula 1*) is one of the most basic and commonly used evaluation metrics in classification tasks. It measures the proportion of correctly classified instances out of the total instances in a dataset.

$$\boldsymbol{acc = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} = \frac{TP + TN}{TP{+}FP{+}FN{+}TN}}$$

Formula 1: Accuracy

This metric assumes the homogeneous distribution of the classes in the dataset. Due to this, despite its simplicity and popularity, it may not be the most suitable metric for imbalance datasets. In our case, where we are dealing with imbalanced data, accuracy can be misleading. The model might achieve a high accuracy by correctly predicting the majority class but fail to accurately predict the minority class, which is the class of greater interest.

Going back to the fraud detection example, where most transactions are legitimate, a model that labels all transactions as legitimate would still obtain high accuracy due to the skewed class distribution. However, it would completely fail to detect fraudulent transactions.

Due to this vulnerability to imbalanced datasets, accuracy is not the best option. It is important to consider additional evaluation metrics that consider the characteristics of imbalanced data.

2.1.3.2. False Positive Rate (FPR)

FPR is a metric that measures the proportion of actual negative instances that are incorrectly classified as positive by the model (*Formula 2*). It is useful when we want to focus on minimizing the misclassification of negative instances.

$$\boldsymbol{FPR = \frac{FP}{FP + TN}}$$

Formula 2: False Positive Rate

Since FPR primarily focuses on negatives, it might not adequately address the challenges posed by the positive (minority) class, which in this case the primary concern.

2.1.3.3. True Negative Rate (TNR)

TNR measures the proportion of actual negative instances that are correctly classified as negative by the model (*Formula 3)*. TNR is useful for scenarios where correctly identifying negative instances is crucial and false negatives are costly.

$$\boldsymbol{TNR = \frac{TN}{TN + FP}}$$

Formula 3: True Negative Rate

Like FPR, TNR predominantly focuses on the performance of the negative class. In cases with imbalanced datasets and a strong emphasis on accurately classifying the positive minority class, it doesn't provide a global evaluation.

In fraud detection or similar applications of imbalanced problems, the main concern is correctly identifying fraudulent cases (positive class) to prevent financial losses. Metrics like Precision, Recall, F1 Score and GM, which specifically consider the positive class, are more appropriate for evaluating model performance in such imbalanced scenarios. These metrics help balance the trade-off between correctly identifying positive instances and minimizing false positives, making them more suitable for our case.

2.1.3.4. Precision

Precision measures the proportion of correctly predicted positive instances (TP) out of all instances that the model predicted as positive (TP, FP), (*Formula4)*. With this metric, we analyze the positive or minority class, which holds significant information despite being underrepresented. It is a very useful metric when the cost of false positives is high, as it indicates the accuracy of positive predictions relative to the total predicted positives.

$$\boldsymbol{precision = \frac{TP}{TP + FP}}$$

Formula 4: Precision

In fraud detection, high precision indicates that the model’s positive predictions are reliable and less prone to false alarms.

2.1.3.5. Recall

Recall or *True Positive Rate* measures the proportion of correctly predicted positive instances (TP) out of all actual positive instances (TP, FN), (*Formula 5*). It is important when the cost of false negatives is high, as it assesses the model’s ability to identify all instances of the positive class.

$$\boldsymbol{recall = TPR = \frac{TP}{TP + FN}}$$

Formula 5: Recall

In the example about fraud detection, recall suggests that the model is effective at capturing a significant portion of actual positive instances, which is crucial in scenarios where the negative class dominates the dataset.

After analyzing these metrics, it becomes evident that in scenarios with imbalanced datasets, a need for more balanced evaluation metrics arises. Two metrics that address this concern are the *Geometric Mean (GM)* and the *F1 Score*.

2.1.3.6. F1 – Score

The F1 Score is a single score that combines precision and recall. It balances both metrics. It is well-suited for imbalanced datasets where the minority class is vital. It captures the nuances of the positive class while taking false positives and false negatives into account.

$$\boldsymbol{F1 = 2 * \frac{Precision * Recall}{Precision + Recall}}$$

Formula 6: F1-Score

2.1.3.7. Geometric Mean (GM)

The Geometric mean is a metric that provides a balanced assessment since it considers both positive and negative classes' performance (*Formula 7*). It is a more balanced indicator of overall model performance, particularly in scenarios with imbalanced datasets. This is crucial because imbalanced datasets often have a majority class that dominates, leading to high accuracy but poor performance on the minority class. GM considers both classes and aims to provide a reliable assessment that considers true positives and true negatives.

$$\boldsymbol{GM = \sqrt{Recall * TNR}}$$

Formula 7: Geometric Mean

Both GM and F1 Score address the need for more balanced evaluation metrics in imbalanced scenarios. They offer a comprehensive view of model performance, considering the challenges posed by skewed class distributions. In applications like fraud detection, where correctly identifying the minority class is crucial, these metrics provide a more accurate assessment of the model's effectiveness and help strike a balance between different aspects of performance.

2.1.3.8. Area Under the Curve (AUC)

AUC is a metric used to assess the performance of binary classifiers by measuring the area under the Receiver Operating Characteristic (ROC) curve. AUC provides a score that summarizes the curve and can be used to compare classifiers. The higher the AUC, the better the model's performance at distinguishing between the positive and negative classes.

A ROC curve is a graph that shows how well a classification model performs. The ROC curve plots the True Positive Rate - Recall (how often the model correctly identifies positive cases) against the False Positive Rate (how often it mistakenly identifies negative cases as positive) at various thresholds. It helps us see how the model makes decisions at different levels of certainty. By looking at this graph, we can understand how good the model is and choose the threshold that gives us the right balance between correct and incorrect predictions. (*Figure 2*)

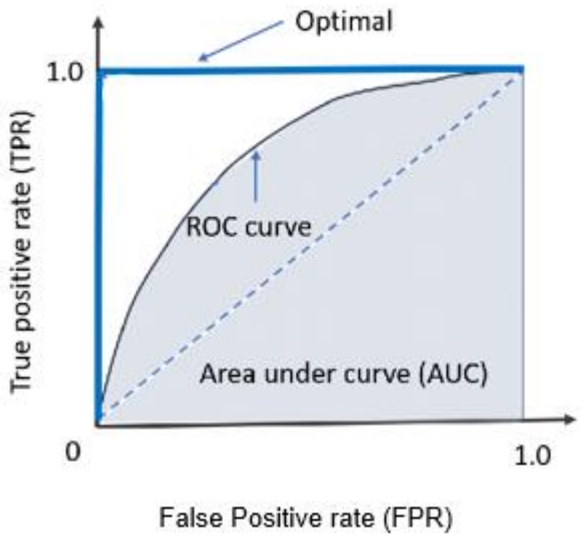


Figure 2: AUC-ROC curve [7]

In imbalanced datasets, where one class dominates the other, AUC is valuable because it assesses a model's ability to differentiate between the classes. A high AUC suggests that the model can effectively distinguish between positive and negative instances, even when they are imbalanced. When AUC=1, we have a perfect classifier that correctly distinguishes between all positive and negative class points. If AUC =0, the classifier would predict all negatives as positives and all positives as negatives. It is particularly useful in applications like medical diagnosis, where correctly identifying rare diseases is critical.

2.1.3.9. Area Under the Precision-Recall Curve (AUCPR)

AUCPR is a metric used to evaluate binary classifiers by measuring the area under the Precision-Recall curve.

A precision-recall curve (or PR Curve) plots precision (y-axis) against recall (x-axis) for different probability thresholds, focusing on the positive class's performance. AUCPR quantifies the classifier's ability to balance precision and recall (*Figure 3, b*).

The Precision-Recall AUC is just like the ROC AUC. It summarizes the curve with a range of threshold values as a single score. It provides insights into how well a model can make precise positive predictions while capturing as many true positives as possible. A high AUCPR indicates that the model maintains high Precision even when Recall is important. As with AUC, the perfect classifier is given when AUCPR = 1.

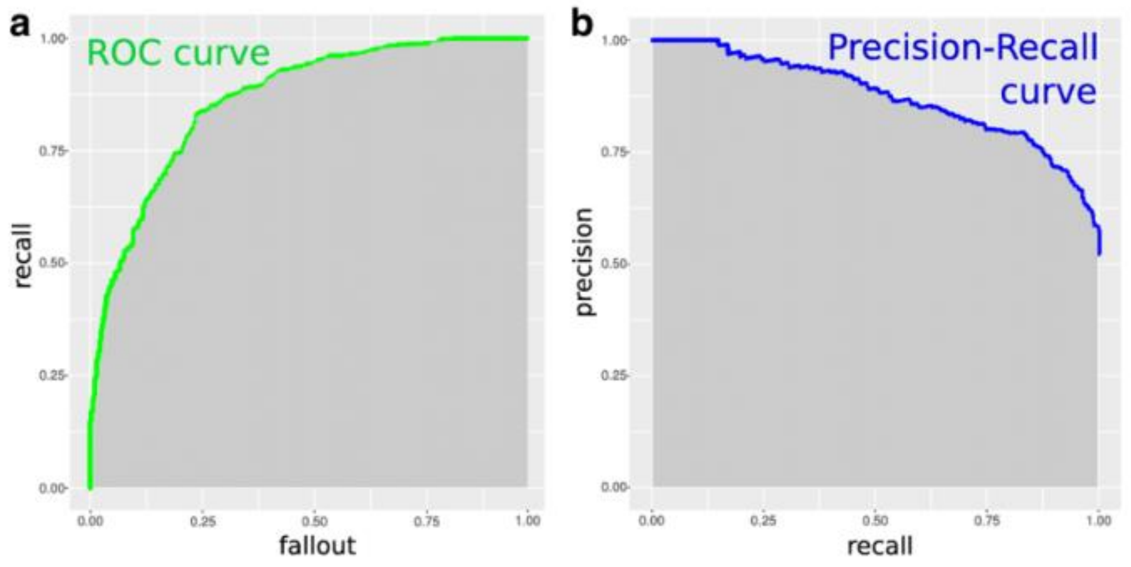


Figure 3:AUC and AUCPR comparison [8]

Comparing both AUC and AUCPR (*Figure 3*), the first graph, a , represents a ROC curve with the recall (true positive rate) score on the y axis and the fallout (false positive rate) score on the x axis. The second graph, b, is a Precision-Recall curve, with the precision score on the y axis and the recall score on the x axis. In both graphs the gray area is the Area Under the Curve respectively (AUC and AUCPR).

In imbalanced scenarios, AUC and AUCPR offer a more nuanced assessment of a model's ability to correctly classify instances, particularly when the distribution of classes is uneven. While AUC focuses on overall classification performance, AUCPR is especially relevant when the positive class is underrepresented, ensuring that the model maintains precision in critical applications.

2.2. Imbalanced Classification Problems

An imbalanced classification problem is given when the distribution of classes in the training dataset is highly skewed, meaning that one class has significantly more examples than the other. This imbalance can be challenging for many classification algorithms, as they tend to perform poorly when one class is heavily outnumbered by the other. The underrepresented class is called the minority class while the more prevalent is the majority class.

To quantify this imbalance in the number of instances of a dataset, the Imbalanced Ratio (IR) [9] is usually used. The IR is calculated by dividing the number of instances in the majority class by the number of instances in the minority class (*Formula 8*).

$$\boldsymbol{IR = \frac{N^{o}\ of\ Majority\ Class\ Instances}{N^{o}\ of\ Minority\ Class\ Instances}}$$

Formula 8: Imbalanced Ratio

The imbalanced ratio provides insight into the severity of the class imbalance in the dataset. It's an important parameter to consider because it helps in understanding the relative prevalence of different classes. A higher imbalanced ratio indicates a more severe class imbalance, which can impact the performance of machine learning algorithms.

Imbalanced classification poses several challenges. Firstly, there is bias towards the majority class. Most of the machine learning algorithms, aim to maximize overall accuracy, which obtains high value in performance, but it does not reflect the reality since it predicts the majority class accurately while ignores the minority class almost completely. This is a severe problem since the minority class represents crucial instances.

Moreover, the ability of the model to generalize is compromised. The model fails to generalize effectively to new, unseen data, particularly for the minority class. As a result of this, it fails to classify instances from the minority class.

Additionally, instances from the minority class are often overlapped by those from the majority class. This overlap makes it challenging for the classifier, often treating them as outliers or noise.

Furthermore, in certain scenarios, minority classes exist as small, disjointed clusters, which adds an extra layer of complexity to the challenge of accurately classifying these instances.

However there exist various ways to solve the imbalanced classification problem, broadly classified into two categories: algorithm-level [4] and data-level [3] approaches.

2.2.1. Algorithm- level approaches

Algorithm-level approaches are designed to tackle class imbalance by modifying the inner workings of existing classifiers. These methods involve adjusting the algorithms' internal mechanisms to ensure they can better accommodate imbalanced data. However, such approaches tend to be specific to the classifier being used, and their effectiveness often hinges on expert knowledge of the task and classifier at hand. To improve performance, careful fine-tuning of hyperparameters is usually required.

Cost-sensitive learning is a method that modifies the learning algorithm to assign different misclassification costs to different classes. This makes the classifier pay more attention to the minority class.

In addition to this, ensemble learning techniques are often combined with AdaBoost [10], Random Forest [11] or XGBoost [12] methods to enhance the overall performance. They can handle class imbalances by combining multiple models but face high computational costs.

2.2.2. Data-level approaches

Data-level approaches focus on manipulating the data distribution to achieve a better balance between the instances of the two classes. These techniques aim to balance the representation of the minority and majority classes, thereby improving the performance of classifiers on the minority class. In this work we will focus on data-level approaches. There are two primary types of data-level approaches: undersampling and oversampling.

2.2.2.1. Undersampling

Undersampling techniques focus on reducing the number of instances in the majority class to match the number of instances in the minority class. These techniques can help prevent the classifier from being biased towards the majority class and may lead to an improvement in the generalization of the model. However, can lead to loss of information and potential underfitting due to the reduced training data.

2.2.2.1.1. Random Under-Sampling (RUS) [13]

Random Under Sampling is a non-heuristic method that randomly deletes as many instances as needed from the majority class until the distribution of the classes is balanced (*Figure 4*).

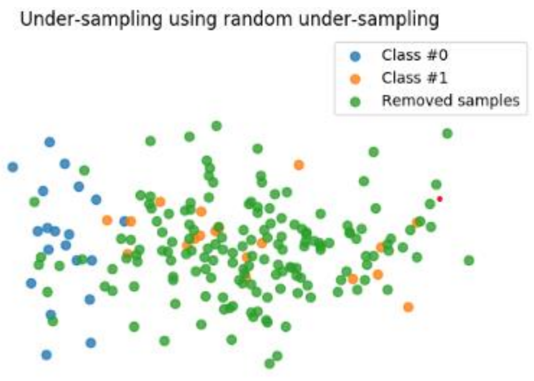


Figure 4. Random Under Sampling – RUS [14]

RUS is a very simple method to implement, with a very reduced computational cost. As it is a random method, it discards noise instances, but it can also potentially discard valuable information, leading to reduced representational capacity. However, it is a commonly used method since it is computationally fast simple to implement and can obtain good results.

2.2.2.1.2. Tomek Links (TL) [15]

Tomek Links is a technique used to identify pairs of instances where one instance belongs to the minority class and the other to the majority class. These pairs are known as Tomek links. In order to be consider as a Tomek Link, instances are each other's nearest neighbors. After identifying the Tomek Links, the technique eliminates the instance of the majority class (*Figure 5*). The idea of TL is that when instances from distinct classes are close to each other, they might be interpreted as ambiguous or noisy and could negatively impact the classifier's performance.

The goal is to improve the separation between the classes' decision boundaries. One of the benefits of TL is that the concept is straightforward and does not require the generation of synthetic data or resampling.

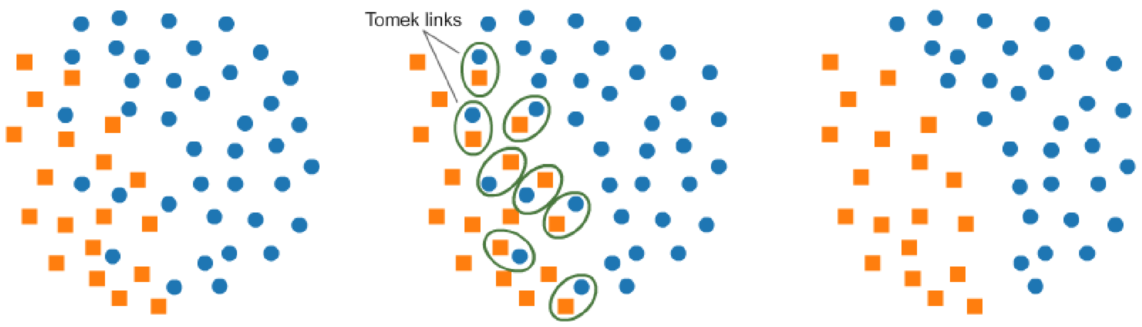


Figure 5: Tomek Links Technique - TL [16]

However, TL can be overly aggressive and can remove correctly classified instances that are merely close to the other class.

2.2.2.1.3. Condensed Nearest Neighbor (CNN) [17]

Condensed Nearest Neighbor is a technique that reduces the majority class instances while retaining the essential information needed to classify the minority class accurately.

CNN initially creates an empty subset in which introduces the first instance from the majority class and all instances from the minority class. Through an iterative process, CNN examines majority class instances to determine if they can be correctly classified using the existing subset. Misclassified instances are progressively added with the nearest neighbor algorithm, and the process continues until additional majority class instances cannot be integrated without inducing misclassification. This ensures that instances that are crucial for accurate classification are preserved.

However, CNN faces computational complexity due to iterative processes, potential bias towards instances closer to the minority class, and a risk of overfitting if not carefully managed during application. Additionality if the dataset presents noisy data, it doesn't eliminate it, it retains it.

2.2.2.1.4. One-Sided Selection (OSS) [18]

OSS combines elements of TL and CNN to improve classifier performance by selecting a subset of majority class instances that are both informative and representative.

- CNN removes examples from the negative class that are far from the decision boundary. This helps to retain instances that contribute to the decision boundary.
- Tomek links remove examples from the negative class that are considered as noise or examples on the boundary. (*Figure 6*)

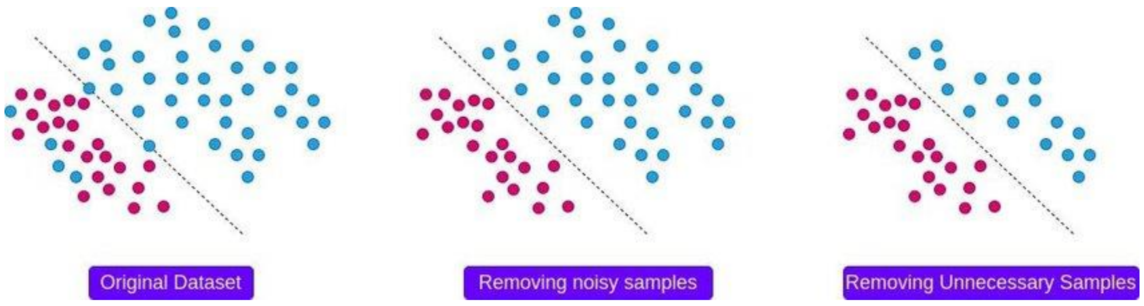


Figure 6: One-Sided Selection – OSS [19]

One-Sided Selection effectively addresses class imbalance by achieving a balanced dataset while preserving the quality of closely related majority class instances to the minority class. This improves model generalization.

Moreover, the use of Tomek Links eliminates noisy instances from the majority class, augmenting data quality. By retaining informative majority class instances that contribute to the decision boundary, OSS can lead to more accurate classification in regions where the classes overlap.

Despite its advantages, OSS may still discard some majority class instances, potentially resulting in a slight loss of information. In addition, this technique is computationally intensive. Careful parameter tuning is essential to ensure optimal performance and balance between overfitting and underfitting.

2.2.2.1.5. Neighborhood Cleaning Rule (NCL)[20]

NCL focuses into identifying and removing noisy instances from the majority class while preserving the informative ones. NCL starts by selecting instances from the majority class based on their similarity to instances from the minority class. These selected instances from the majority class are evaluated for their closeness to the minority class instances. Instances that are near the minority class and classified incorrectly are considered noisy and removed.

After identifying the instances from the majority class that are close to the minority class and have been misclassified, NCL takes an additional step. It focuses on the minority class instances that are wrongly classified by the K-nearest neighbors (KNN) algorithm.

In the context of NCL, this step becomes crucial. NCL examines the misclassified minority class instances and identifies which instances from the majority class are responsible for these misclassifications. These influential majority class instances are known as the "neighbors" in the NCL acronym. NCL then proceeds to eliminate these neighbors.

This method significantly reduces noise in the dataset by removing majority class instances that are misclassified. However, the performance of NCL can be influenced by the choice of parameters, such as the threshold for identifying noisy instances and the similarity measure used.

2.2.2.2. Oversampling

Oversampling artificially increases the number of instances in the minority class, creating a more balanced class distribution and mitigating the biases introduced by class imbalance. These instances are normally generated by duplication, replication, or synthetic generation methods. It offers benefits in terms of improved classifier performance but requires careful consideration to avoid potential pitfalls like overfitting and increased complexity.

2.2.2.2.1. Random Over Sampling – ROS [13]

The Random Over Sampling method randomly selects instances from the minority class and duplicates them until the desired balance is achieved. This helps prevent the classifier from being overly biased toward the majority class and can lead to improved classification results on the minority class.

As seen on the graph (*Figure 7*), the ROS method duplicates the existing instances until the number of instances in the minority class is the same as the number of instances in the majority class. The duplicated instances on the graph are represented with a higher intensity of blue.

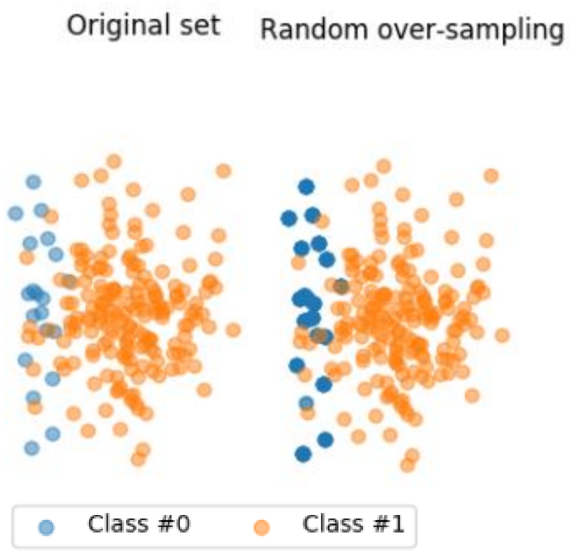


Figure 7: Random Over Sampling - ROS [21]

This method is an easy and fast solution to balance the class distribution, but it can lead to overfitting, where the classifier becomes too focused on duplicated instances. Additionally, the duplication of instances might lead to data redundancy, where the model could rely heavily on the same information, reducing the model's adaptability.

2.2.2.2.2. Synthetic Minority Over-Sampling – SMOTE [22]

SMOTE generates new synthetic instances that lie in the space between existing minority class instances. This technique helps increase the representation of the minority class while introducing diversity into the dataset, thereby enhancing the classifier's ability to generalize.

SMOTE selects a minority class instance and identifies its k nearest neighbors within the minority class instances. For each selected instance, SMOTE creates new instances by selecting one of its k nearest neighbors and generating a synthetic instance along the "space" connecting them. The synthetic instance's features are generated by interpolating between the features of the selected instance and its neighbor (*Figure 8*).

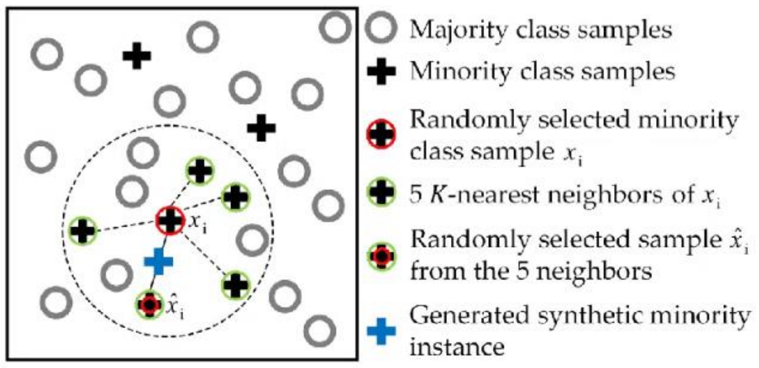

x_i
$\hat{x}_i$
Majority class samples
Minority class samples
Randomly selected minority class sample x_i
5 K-nearest neighbors of x_i
Randomly selected sample $\hat{x}_i$ from the 5 neighbors
Generated synthetic minority instance

Figure 8: Synthetic Minority Over-Sampling- SMOTE technique [23]

SMOTE introduces diversity to the dataset, reducing the model's tendency to memorize the training data resulting in a reduction of the chances of overfitting.

However, potential drawbacks include the risk of introducing noise through these new instances, sensitivity to the choice of the parameter k (number of nearest neighbors), and the potential impact on the decision boundary, which could amplify noise or outliers in overlapping class regions. Careful parameter tuning and noise management are important considerations when applying SMOTE.

2.3. Development environment

Python 3.8, a version of Python 3, has been employed for the development of this work. This high-level programming language is known for its simplicity and readability. Python 3 has an extensive ecosystem of useful libraries, particularly in the domains of machine learning and addressing imbalanced classification problem.

Scikit-learn (sklearn) is a versatile machine learning library that offers tools for classification, regression, clustering, and more. In this project, it has been utilized for tasks ranging from learning algorithms (k-nearest neighbors, decision trees) to evaluating and selecting models (cross-validation and metrics), performing data preprocessing using the preprocessing module, and evaluating using the metrics module.

Imbalanced-learn is a specialized library within scikit-learn specifically designed to tackle imbalanced classification problems, offering a diverse range of techniques for resampling datasets. The imports that have been used in the project encompass techniques for calculating metrics such as geometric mean scores, which evaluate the overall performance of models in imbalanced classification scenarios. Additionally, the imblearn.pipeline module has been employed to construct data processing pipelines integrated with model training, streamlining the workflow through preprocessing steps. Moreover, techniques from imblearn.under_sampling and imblearn.over_sampling modules have been used to address class imbalance.

NumPy is a fundamental library for numerical computations in Python. It provides support for large, multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays.

Pandas is another essential library for data manipulation and analysis. It offers data structures like DataFrames that allow to organize, clean, and preprocess the data efficiently.

Matplotlib is a popular data visualization library in Python. It allows to create various types of plots, charts, and graphs to visualize your data and results. It has been essential to create visualizations that helped understand the distribution of classes and performance metrics.

3. PERTURBATION-BASED OVERSAMPLING

In the context of addressing imbalanced classification challenges, a range of techniques have emerged to counter this issue as stated before (section 2.2). This study focuses on data-level techniques, particularly in a new approach of oversampling.

Recent research shows the effectiveness of generating instances that align with the minority class distribution [5] or disperse through embedded probability density [6] to enhance predictive performance. However, these methods require density function fitting or manifold learning techniques, leading to computational intensity.

This study investigates the novel approach within this framework, introduced by researchers affiliated with the Department of Electrical and Computer Engineering at the University of Alberta, Canada. Their approach, detailed in the paper titled "Perturbation-based oversampling technique for imbalanced classification problems," aims to introduce an innovative oversampling technique. This method is designed to generate new instances like the original minority instances, dispersing them throughout the feature space to mitigate overfitting.

Notably, it avoids the necessity to identify k-nearest neighbors for efficient execution. The proposed technique, known as Perturbation-Based Oversampling (POS), accomplishes this by generating novel instances from existing minority instances. This rebalances the dataset by perturbing each feature of the minority instances, with the extent of perturbation controlled by a hyperparameter 'p'.

The objective of this study is to explore and validate the efficacy of this novel proposition in terms of delivering robust performance and resolving imbalanced dataset challenges. Furthermore, a comparative analysis will be conducted between the outcomes obtained through the utilization of the POS technique and the results by established models from existing methodologies mentioned before (section 2.2.2).

Through the analysis of this paper, certain ambiguities were identified along with shortcomings in the method's description. Therefore, we have chosen to reinterpret and modify specific elements based on individual judgment.

3.1. Functioning of the Perturbation-Based Oversampling (POS) Technique

As stated before, the POS technique operates by generating additional instances from the existing minority instances, thus obtaining dataset rebalancing through controlled perturbations.

POS generates a new instance by perturbing the feature of a random instance from the minority class.

The *j*th feature of a new instance **z** within a *d*-dimensional space $\mathbf{R}^d$ is determined using the following equation:

$$z_j = x_j + \lambda_j$$

Formula 8: Equation that obtains a new instance z.

Where $\mathbf{z}_j$ represents the *j*th feature of the new instance, x_j corresponds to the *j*th feature of a reference or seed instance, and λ_j indicates a specific perturbation applied to that feature. The variables **x** and λ_j belong to the d-dimensional space $\mathbf{R}^d$, and *d* signifies the total number of features involved. It is assumed that the perturbation is independent of the seed instance, and the features within the perturbation are mutually independent. The variable λ_j follows a normal distribution $N(0, 1/m^p)$ where "m" represents the number of minority instances and "*p*" is the hyperparameter that controls the variance of the perturbation. This hyperparameter "*p*" must be correctly fitted and has to verify $p > 0$.

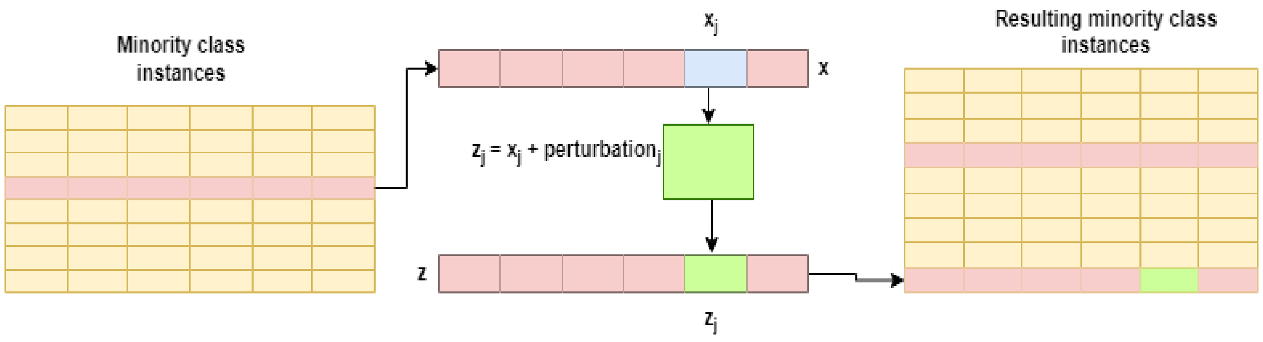

Minority class instances
x_j
x
$z_j = x_j + \text{perturbation}_j$
z
z_j
Resulting minority class instances

Figure 9: Explanation diagram of how the POS method obtains each new instance. This is repeated until the dataset is balanced.

As it can be seen on the diagram (*Figure 9*), a random sample is obtained from the majority class, x, and a perturbation to the *j*th feature of this sample is applied, resulting in an alteration of that feature, z_j. In this way a new instance z is obtained and is introduced into the set of minority class instances. This process is repeated until the dataset is balanced, and the amount of minority class instances and majority class instances is the same.

The pseudocode of the algorithm for the implementation of the Perturbation-Based Oversampling method is as follows (*Figure 10*):

```
Algorithm 1 Perturbation-based oversampling
Require:
  $P$, minority instances;
  $N$, majority instances;
  $p$, hyperparameter;
  $q$, number of instances to be generated;
Ensure: A rebalanced dataset $D$
  Set $i = 0$, $S = \phi$
  while $i < q$ do
      Randomly draw a seed instance from $P$
      Generate a new example $\mathbf{z}$ using (1)
      Put $\mathbf{z}$ in $S$
      $i = i + 1$
  end while
  Return $D = P \bigcup N \bigcup S$
```

Figure 10: Perturbation-Based Oversampling (POS) Algorithm in which generates z using Formula 8

The provided algorithm lacks clarity regarding the selection of feature "j" and whether the process is intended for each individual feature of the instance. To address this ambiguity, a decision has been made to introduce multiple versions of the algorithm, aimed at refining, and improving its initial proposal. This step has been taken to ensure a more comprehensive understanding of the technique and to explore potential enhancements in terms of feature selection and perturbation strategies. By offering alternative versions of the algorithm, the aim is to provide greater clarity and specificity in the perturbation process, thereby contributing to a more robust and effective oversampling technique.

3.2. POS 1.0.

The Perturbation Based Oversampling 1.0 (POS 1.0) method is an oversampling technique designed to address class imbalance in datasets. Its approach involves generating new instances for the minority class by applying perturbations to the features of existing instances.

The key difference between POS 1.0 and the original POS lies in how the new instance is calculated. In POS 1.0, the perturbation is applied to all features of the instance, as the algorithm does not explicitly mention how the specific feature to be perturbed is chosen. Therefore, POS 1.0 employs a broader methodology, uniformly perturbing all features within each generated instance. It is important to note that the same perturbation is applied uniformly across all features. This perturbation is derived from a normal distribution with parameters $N(0, 1/m^p)$, where "m" represents the number of minority instances and "p" controls the variance of the perturbation.

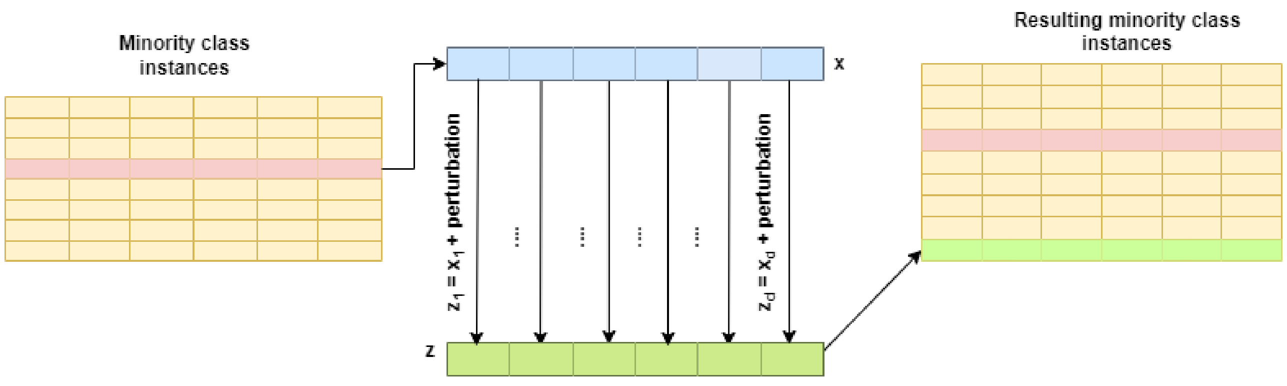


Figure 11: Explanation diagram of how the POS 1.0 method obtains each new instance. This is repeated until the dataset is balanced.

As it can be seen on the diagram (*Figure 11*), the procedure of the POS 1.0 method is different from the original Perturbation-Based Oversampling method. In this case, a unique perturbation obtained using a normal distribution is applied to each one of the features of the randomly selected instance from the minority class. Then the resulting instance is annexed to the minority class set. This process is repeated *q* times (until the number of instances in both classes is the same).

In summary, POS 1.0 revolves around generating new instances for the minority class by applying perturbations to all features, distinguishing it from the original POS in the manner of calculating the new instance.

3.3. POS 2.0.

In the case of POS 2.0, the key distinction lies in the method used to calculate the newly generated instance. In contrast to the original POS approach, POS 2.0 employs a more sophisticated strategy. In POS 2.0, an array of perturbations is created, each corresponding to a specific feature in the instance. These perturbations are obtained from a normal distribution $N(0, 1/m^p)$, with varying values for each feature.

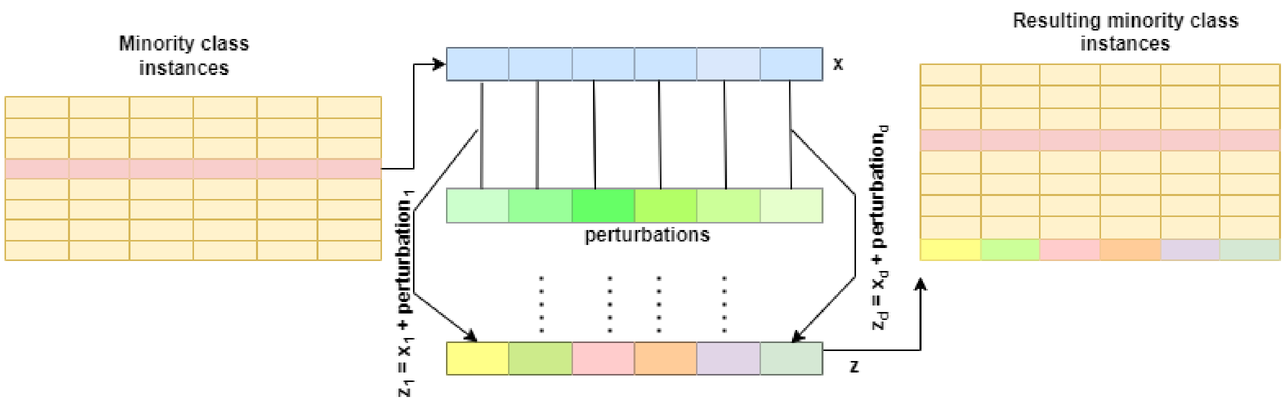


Figure 12: Explanation diagram of how the POS 2.0 method obtains each new instance. This is repeated until the dataset is balanced.

This means that a different perturbation value is applied to each feature, capturing the diverse characteristics of the dataset. Essentially, POS 2.0 introduces individualized perturbations to each feature, allowing for a more nuanced adjustment of the newly generated instances. This enhanced approach provides greater flexibility in the generation process, potentially leading to improved performance when addressing imbalanced classification problems.

In the plots presented below (Figure 13), the operational mechanisms of each method become evident. The original dataset displays an imbalance between the positive class (depicted in red) and the negative class (depicted in blue). All variations of the methods address this imbalance by perturbing the existing minority class instances, each employing a distinct approach as previously explained. POS 1.0 and POS 2.0 introduce a higher level of perturbation compared to the original POS, as discussed in sections 3.2 and 3.3 since these variants apply perturbation to all features within an instance, while the original POS method perturbs only a subset of features. Importantly, these methods ensure that the generated new instances still belong to the minority class, despite the introduction of diversity.

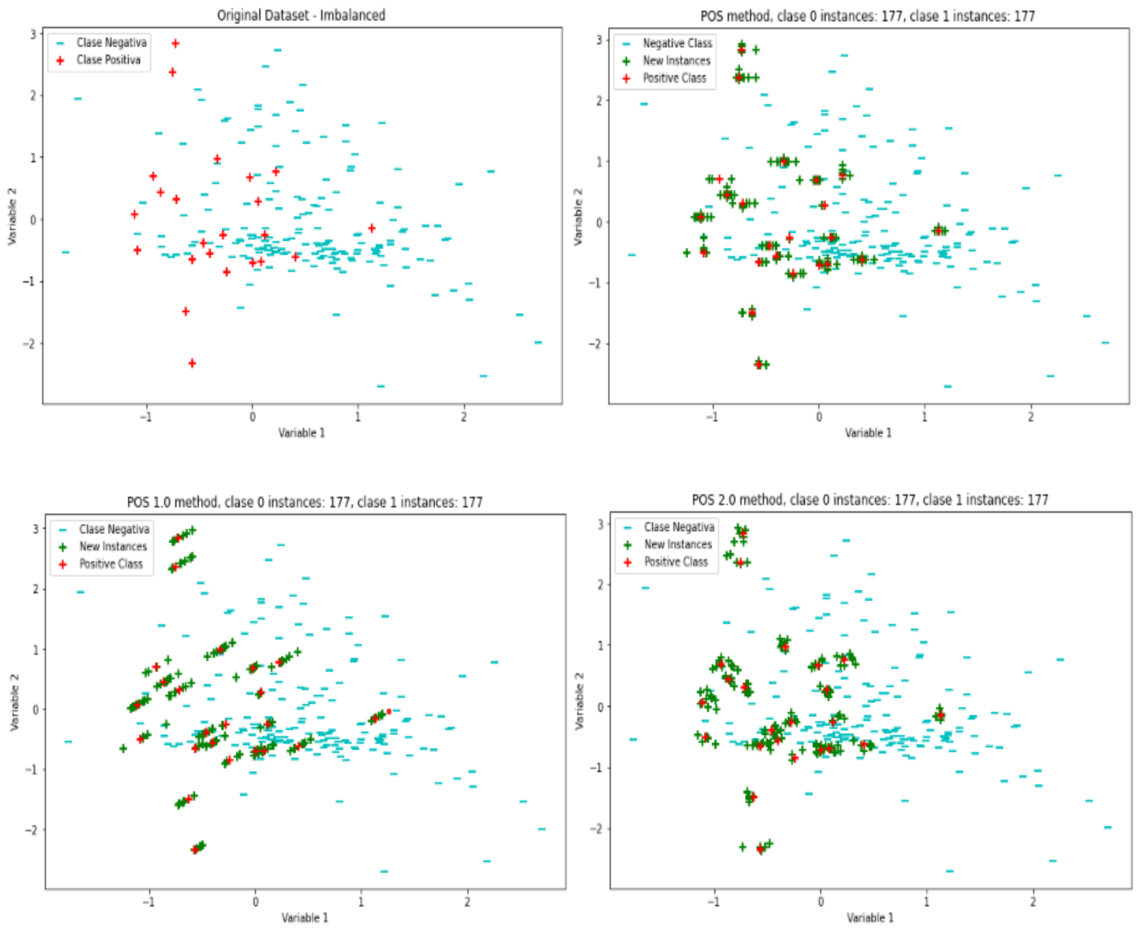


Figure 13: Comparison between the Original Imbalanced Dataset, POS, POS 1.0, and POS 2.0

3.4. Hyperparameter ‘p’

The hyperparameter "p" in Perturbation-based Oversampling (POS) plays a significant role in controlling the magnitude of perturbation applied to generate new instances for the minority class. In the POS method, the diversity is introduced into the minority class by creating synthetic instances through perturbation. Perturbation is the process of adding small variations or noise to existing instances to create new, similar instances. The hyperparameter "*p*" determines the extent of this perturbation, influencing the level of diversity and similarity between the original and generated instances.

In POS, when generating a new instance, the perturbation is calculated by adding a value sampled from a normal distribution with mean 0 and a standard deviation determined by the formula "$\mathbf{1/m^{p}}$", where "**m**" represents the number of minority instances and *“p” > 0*. The value of "***p***" directly affects the spread of the distribution, which in turn impacts the degree of perturbation added to each feature.

Choosing the right value for "*p*" is crucial because it strikes a balance between creating diverse instances and maintaining the representativeness of the minority class. If "*p*" is too large, the perturbation will be minimal, leading to instances that are very similar to the originals and potentially resulting in overfitting. On the other hand, if "*p*" is too small, the perturbation will be significant, introducing too much diversity and potentially generating instances that are too dissimilar from the original distribution, which might lead to misclassification.

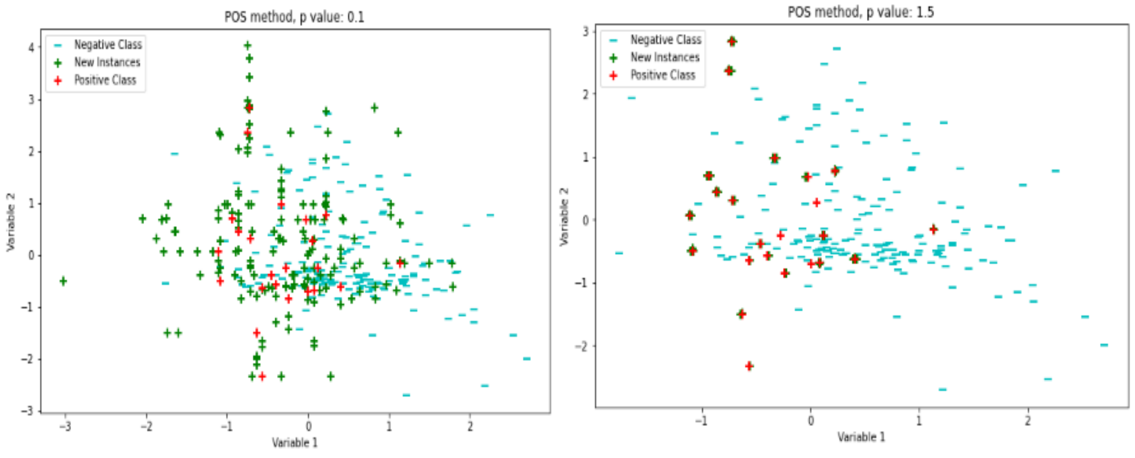


Figure 14: Comparison of the results obtained with a low and a high value for "p".

As shown in the plot (*Figure 14*) it becomes evident that a lower value of “*p”* (e.g., *p* = 0.1) leads to a notable increase in dataset diversity due to the higher magnitude of perturbation applied to instances. On the contrary, employing a higher value for “*p*” (e.g., *p* = 1.5) results in the opposite outcome.

The relationship between "**m**" and "***p***" influences the balance between introducing diversity to the synthetic instances and maintaining proximity to the original data distribution. When "*p*" is large and "m" is small, then "$1/m^p$" tends towards zero, the magnitude of perturbation is reduced, meaning that the introduced variation in the features of instances becomes very small. This can be beneficial in situations where the number of instances in the minority class is small, and a more cautious generation of new instances is desired to prevent them from straying too far from the original data.

As "*p*" increases, the magnitude of perturbation applied to instances will decrease, providing flexibility to handle class imbalance problems with varying amounts of minority instances.

In practical terms, selecting an appropriate value for "*p*" requires experimentation and domain knowledge. A higher value of "*p*" might be suitable when the dataset has a clear class boundary and only slight variations are needed. A lower value of "*p*" might be chosen when the class distribution is highly imbalanced, and greater diversity is required to better represent the minority class.

Ultimately, the choice of "*p*" in POS should be guided by a balance between the need to introduce diversity and the goal of maintaining a representative minority class distribution. Proper tuning of "*p*" can significantly impact the effectiveness of POS in addressing class imbalance while avoiding overfitting or introducing excessive noise.

4. EXPERIMENTAL STUDY AND IMPLEMENTATION OF THE METHODS

4.1. Experimental Framework

The performance of each method is evaluated using the 5-fold cross-validation method as stated before (sec. 2.1.2.3). For each dataset, the evaluation metrics are computed by taking the average of their values over multiple runs. This is achieved using the Python library scikit-learn, which provides a method called *model_selection.StratifiedKFold()*. It applies stratification to ensure data consistency.

In this context, the method employs 5-fold cross-validation, which means the dataset is divided into 5 subsets or folds. Moreover, the *shuffle* parameter is set to True, introducing randomness in the data splitting process. The parameter *random_state* is set to 42, ensuring reproducibility by using a fixed random seed for the shuffling.

The initial datasets are treated in order to be used. As mentioned in section 2.1.1.1, the dataset is scaled using ***Normalization*** to avoid some features from dominating others due to their large magnitudes. Python library scikit-learn offers a convenient tool called *preprocessing.MinMaxScaler().*

To conduct a consistent comparison under the same conditions for all models, common parameters and aspects are set.

- **Classifiers:** Two kinds of classifiers are used. All the experiments are conducted for each classifier to verify if the behavior is stable. The main classifiers used is KNN classifier with its default parameters. This classifier is obtained from ***sklearn.neighbors. KNeighborsClassifier()***. Then, the experiments are conducted again but using the CART decision tree, obtained from ***sklearn.tree.DecisionTreeClassifier().***

- **Evaluation Metrics:** The model is evaluated using four metrics. The main metric used is the Geometric Mean obtained from the Python library *imblearn* as ***imblearn.metrics.geometric_mean_score().*** Also, the F1 score, AUC and AUCPR metrics are used. Obtained as: ***imblearn.metrics.f1_score(), imblearn.metrics.roc_auc_score ()*** and ***imblearn.metrics.average_precision_score()*** respectively.

- **Number of neighbors:** The number of neighbors to be considered when making prediction (k_neighbors) is fixed to 5, the default value.

To implement the POS methods and the versions proposed POS 1.0 and POS 2.0, the implementations of a similar algorithms was analyzed. Precisely , ROS and SMOTE obtained from a GitHub repository: https://github.com/scikit-learn-contrib/imbalanced-learn/blob/9f8830e/imblearn/over sampling/ random over sampler.py and https://github.com/scikit-learn-contrib/imbalanced-learn/tree/9f8830e/imblearn/over sampling/ smote

By examining these oversampling methos and their Python implementations, a similar structure has been adopted to simplify and ensure their proper functionality, facilitating a seamless comparison.

4.2. Datasets

In order to conduct various experiments involving all implemented models and pre-existing models to test their performance and analyze and compare their results, a set of 28 datasets has been employed.

The datasets have been sourced from a repository named **KEEL-dataset** (Knowledge Extraction Evolutionary Learning), which serves as a comprehensive collection of datasets widely used in the machine learning community. The selected datasets exhibit evident class imbalance, encompassing diverse range of Imbalanced Ratios (IR), and instance counts. The datasets consist of real or integer values.

The table below (*Table 2*) provides an overview of all the datasets along with their respective characteristics. The table is organized from datasets with fewer to more instances of the minority class:

Dataset	IR	Nº instances	Nº features	Nº minority class instances
glass-0-4_vs-5	9.22	92	9	9
glass-0-6_vs_5	10.00	108	9	9
shuttle-6_vs_2-3	22.00	230	9	10
glass4	15.47	214	9	13
glass2	11.59	214	9	17
glass-0-1-5_vs_2	9.12	172	9	17
poker-8_vs_6	85.88	1477	10	17
ecoli-0-3-4_vs_5	9.00	200	7	20
ecoli-0-6-7_vs_3-5	9.09	222	7	22
ecoli-0-1_vs_2-3-5	9.17	244	7	24
ecoli-0-3-4-7_vs_5-6	9.28	257	7	25
poker-8-9_vs_6	58.40	1485	10	25
winequality-white-3-9_vs_5	58.28	1482	11	25
glass6	6.38	214	9	29
new-thyroid1	5.14	215	5	35
yeast6	41.40	1484	8	35
yeast5	32.73	1484	8	44
yeast-2_vs_4	9.08	514	8	51

yeast4	28.10	5484	8	51
winequality-red-4	29.17	1599	11	53
glass-0-1-2-3_vs_4-5-6	3.20	214	9	55
glass0	2.00	214	9	70
glass1	1.82	214	9	76
yeast-0-2-5-7-9_vs_3-6-8	9.14	1004	8	99
yeast-0-2-5-6_vs_3-7-8-9	9.14	1004	8	99
yeast3	8.10	1484	8	163
yeast1	2.46	1484	8	429
page-blocks0	8.79	5472	10	559

Table 2: List of datasets and their characteristics used for the testing of the models.

This study is structured into two distinct phases, each addressing critical aspects of the analysis. The initial phase centers around an in-depth exploration of the hyperparameter 'p' and its consequential impact on two key factors: the population of instances within the minority class and the Imbalanced Ratio (IR). This comprehensive investigation aims to shed light on how varying values of 'p' interact with the dataset, affecting the distribution of classes and influencing the overall balance and performance.

Subsequently, the second phase revolves around a comparative assessment of the newly introduced methods in contrast to well-established existing techniques. Once studied the hyperparameter '*p*' and the values it should have in different scenarios to obtain optimal performance, a comparative analysis is conducted. The study seeks to elucidate the method's strengths, weaknesses, and unique attributes, while also evaluating its overall effectiveness within the context of imbalanced classification problems. Through this multifaceted exploration, a comprehensive understanding of the method's performance and its distinct contributions to the field of imbalanced learning is expected to emerge.

4.3. KNN Classifier - Phase 1: Study of hyperparameter '*p*'

As stated in section 3.4, an optimal "*p*" value maintains this equilibrium: if too high, instances closely mimic originals, risking overfitting; if too low, instances may deviate excessively, causing misclassification.

The relationship between "*p*" and the minority instances count "m" could guide the perturbation magnitude. A higher "*p*" is preferred for imbalanced datasets, maintaining proximity to the original data, while a lower "*p*" is chosen when a clearer class boundary exists.

Additionally, the imbalance ratio could influence the choice of "*p*" as it guides the balance between generating diverse synthetic instances and maintaining proximity to the original data distribution. A higher IR indicates a more severe class imbalance, necessitating a careful selection of "*p*" to ensure that generated instances effectively bridge the gap between the minority and majority classes. The choice of "*p*" directly influences POS's effectiveness in combating class imbalance while preserving data representation.

To verify these statements, a series of experiments have been conducted using a subset of datasets that adhere to the desired characteristics for each experiment. For these experiments, a grid search has been performed, classifying with the KNN classifier, testing various values of the hyperparameter "p" (ranging from 0.1 to 1 in increments of 0.1, and including values 1.5, 2, 3, 4, and 5) for each of the methods—POS, POS 1.0, and POS 2.0. These experiments have been conducted for the original POS method and its variants POS 1.0 and POS 2.0. The goal was to assess their performance using geometric mean as evaluation metric on the selected datasets.

4.3.1. POS

In the initial experiment, datasets with a fixed and low number of instances were chosen. This selection aimed to examine the impact of Imbalanced Ratios (IR), which varied from low to high values, and to discern any discernible patterns that might guide the optimal choice of "*p*." By observing the interaction between IR and the selection of "*p*," the objective was to identify an appropriate "*p*" value that aligns with specific IR scenarios.

4.3.1.1. Low fixed number of instances for POS method

In this experiment, a subset of 9 datasets was chosen from the original collection, each containing approximately 200 instances. At first glance to the table (*Table 3*) it appears that the values of "*p*" should be high, but without any direct relationship to IR.

"p" values	glass1.dat	glass0.dat	glass-0-1-2-3_vs_4-5-6.dat	new-thyroid1.dat	glass6.dat	ecoli-0-6-7_vs_3-5.dat2	glass2.dat	glass4.dat	shuttle-6_vs_2-3.dat
0.1	0.781165	0.800397	0.923718	0.964733	0.888274	**0.877038**	0.557738	0.924221	0.997714
0.2	0.778102	0.807587	0.944517	0.979569	0.888274	0.874490	0.476193	0.924221	0.997714
0.3	0.767450	0.789282	0.944517	0.979569	**0.905223**	0.870409	0.513505	0.924221	0.997714
0.4	**0.781819**	0.781971	0.944517	0.976772	**0.905223**	0.864053	0.508834	0.924221	0.997714
0.5	0.775411	0.795545	0.944517	0.976772	0.902424	0.860565	0.563207	0.924221	0.997714
0.6	0.768250	0.791652	0.954780	0.976772	0.904979	0.860565	0.559922	0.921639	0.997714
0.7	0.775464	0.794133	0.954780	0.976772	0.904979	0.860565	0.588090	0.921639	0.997714
0.8	0.777784	0.794133	**0.957882**	0.976772	0.904979	0.862367	**0.727619**	0.921639	0.997714
0.9	0.770026	0.808158	**0.957882**	0.976772	0.904979	0.862367	0.696913	0.921639	**1.000000**
1.0	0.773956	**0.808309**	**0.957882**	0.976772	0.904979	0.862367	0.591709	0.921639	**1.000000**
1.5	0.760737	0.803997	**0.957882**	**0.988770**	0.904979	0.862367	0.558789	0.921639	**1.000000**
2.0	0.760737	0.803997	0.947775	**0.988770**	0.904979	0.864646	0.568756	**0.924256**	0.941421
3.0	0.764500	0.803997	0.947775	**0.988770**	0.904979	0.864646	0.571093	**0.924256**	0.941421
4.0	0.764500	0.803997	0.947775	**0.988770**	0.904979	0.864646	0.571093	**0.924256**	0.941421
5.0	0.764500	0.803997	0.947775	**0.988770**	0.904979	0.864646	0.571093	**0.924256**	0.941421
IR	1.82	2	3.2	5.14	6.38	9.09	11.59	15.47	22
dim	214	214	214	215	214	222	214	214	230

Table 3: Performance results for low number of instances (200 instances) and varied values for IR. Classifier: KNN, Evaluation metric: GM, Method: POS, ordered from lowest to highest value of IR

To ensure a clear observation of the influence of the Imbalanced Ratio (IR) without the interference of other factors such as number of instances, the same experiment was conducted using datasets with a fixed but higher number of instances.

4.3.1.2. High fixed number of instances for POS method

By conducting the same experiment with datasets of high but fixed number of instances, the specific effect of the Imbalanced Ratio (IR) on the hyperparameter "*p*" can be isolated and evaluated. By keeping the number of instances constant, any variation in the results can be directly attributed to differences in the values of "*p*" and their interactions with the IR. This provides a clearer and more precise perspective on how the relationship between IR and "*p*" impacts the performance of the POS methods in addressing class imbalance problems.

In this experiment, another subset of 9 datasets was chosen from the original collection, in this case, containing instances in a range between 1482 and 1599 instances.

"p" values	yeast1	yeast3	yeast4	winequality-red-4	yeast5	yeast6	winequality-white-3-9_vs_5	poker-8-9_vs_6	poker-8_vs_6
0.1	0.679375	0.871918	0.746338	0.500005	0.949641	0.823866	0.452264	0.955455	0.924089
0.2	0.686339	0.872280	0.750715	0.532436	0.946801	0.836103	0.451523	**0.958190**	0.920719
0.3	0.690204	0.890172	0.779114	**0.589162**	0.944724	0.850352	**0.486917**	0.936805	0.921931
0.4	0.693700	0.893033	0.775166	0.545688	0.958277	0.834162	0.484886	0.939181	**0.925447**
0.5	0.698366	**0.900811**	**0.786432**	0.547007	**0.971098**	0.851812	0.449287	0.917988	0.861859
0.6	**0.701495**	0.896000	0.765779	0.570573	0.962818	**0.864232**	0.450351	0.899986	0.863645
0.7	0.692362	0.894730	0.766000	0.549147	0.954773	0.855550	0.451808	0.902270	0.865418
0.8	0.691304	0.891474	0.747687	0.510571	0.969856	0.861029	0.452134	0.903851	0.867657
0.9	0.691990	0.889349	0.742795	0.462480	0.971065	0.863512	0.452560	0.905505	0.870546
1.0	0.692455	0.890476	0.744862	0.463516	0.960924	0.849640	0.453465	0.906158	0.871425
1.5	0.695907	0.888032	0.700969	0.467818	0.949948	0.856168	0.339210	0.909588	0.874886
2.0	0.695907	0.888032	0.701536	0.468309	0.950299	0.842627	0.339429	0.910547	0.839351
3.0	0.695907	0.888032	0.701536	0.468615	0.950602	0.825625	0.339429	0.910547	0.839594
4.0	0.695907	0.888032	0.701536	0.468615	0.950602	0.825625	0.339429	0.910547	0.839594
5.0	0.695907	0.888032	0.701536	0.468615	0.950602	0.825625	0.339429	0.910547	0.839594
IR	2.46	8.1	28.1	29.17	32.73	41.4	58.28	58.4	85.88
dim	1484	1484	1484	1599	1484	1484	1482	1485	1477

Table 4: Performance results for 'p', for high number of instances (1500 instances) and varied values for IR. Classifier: KNN, Evaluation metric: GM, Method: POS, arranged in ascending order of IR.

Analyzing this new table of results (*Table 4*), if there were a direct dependency on the Imbalanced Ratio (IR), both the previous and this experiment should yield similar results. However, it's evident that the obtained results are quite different. In this new experiment, the optimal values for "*p*" are lower, contrary to the previous experiment where they were considerably high. This suggests that the influence might not primarily stem from the IR but from another factor, such as number of instances.

The observation that the results differ between the two experiments suggests that the relationship between the "*p*" value and the IR might not be as straightforward as initially thought. Instead of solely depending on the IR, other factors, such as data number of instances, could be influencing the optimal choice of "*p*."

Following this observation, a decision was made to conduct an experiment in which the Imbalanced Ratio (IR) was kept constant, while datasets with varying number of intances were selected.

4.3.1.3. Fixed IR for POS method

To carry out this experiment, a subset of 10 datasets with a consistent imbalanced ratio of approximately 9 was chosen.

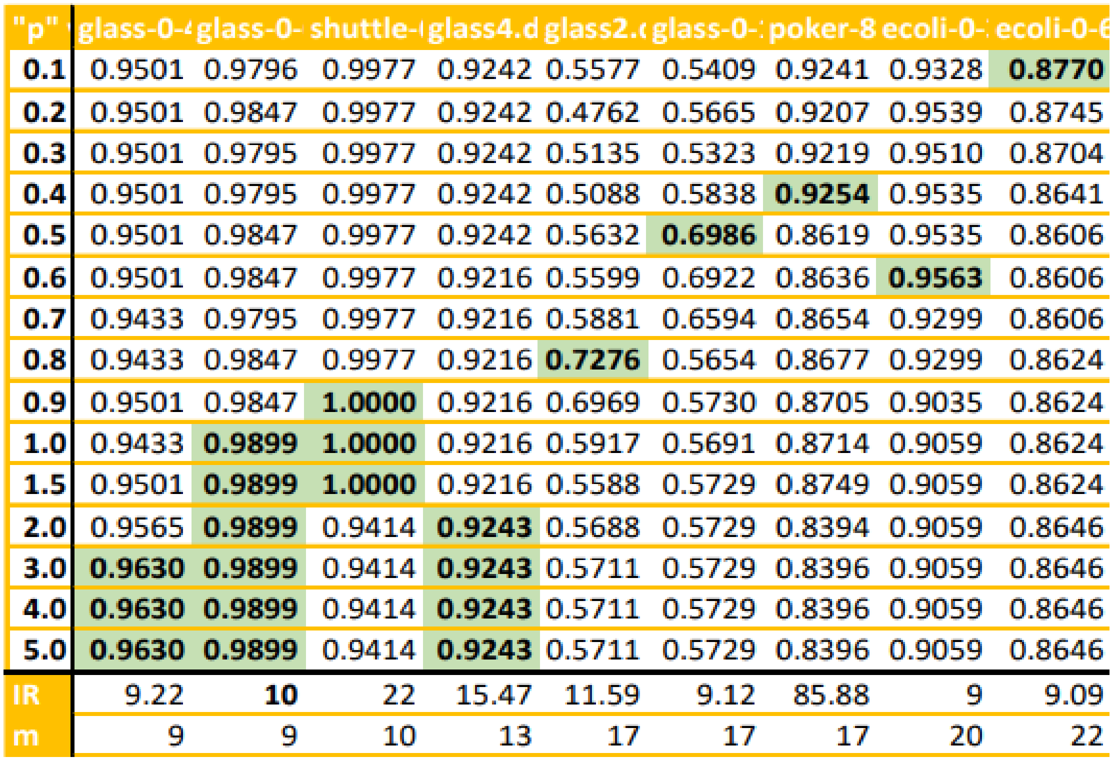
"p" values	glass-0-4_vs_5.dat	glass-0-6_vs_5.dat	glass-0-1-5_vs_2.dat	ecoli-0-3-4_vs_5.dat	ecoli-0-1_vs_2-3-5.dat	ecoli-0-3-4-7_vs_5-6.dat	yeast-2_vs_4.dat	yeast-0-2-5-7-9_vs_3-6-8.dat	yeast-0-2-5-6_vs_3-7-8-9.dat	page-blocks0.dat
0.1	0.950096	0.979608	0.540857	0.932758	0.915382	0.906824	0.908674	**0.903949**	0.780755	0.892589
0.2	0.950096	0.984673	0.566506	0.953918	0.935764	0.923518	**0.929861**	0.897623	0.778662	0.898707
0.3	0.950096	0.979473	0.532262	0.951038	**0.955642**	**0.923765**	0.928744	0.896670	0.779105	0.915101
0.4	0.950096	0.979473	0.583820	0.953461	0.940249	0.896069	0.924439	0.898904	0.785425	0.918871
0.5	0.950096	0.984673	**0.698634**	0.953461	0.938104	0.921476	0.912490	0.888836	0.808955	**0.922820**
0.6	0.950096	0.984673	0.692190	**0.956298**	0.913263	0.923476	0.912490	0.890933	**0.812397**	0.921323
0.7	0.943291	0.979473	0.659386	0.929878	0.908804	0.923476	0.915693	0.886385	0.797422	0.920419
0.8	0.943291	0.984673	0.565352	0.929878	0.906464	0.921476	0.902563	0.889476	0.798476	0.921563
0.9	0.950096	0.984673	0.573012	0.903458	0.906464	0.921476	0.904601	0.889476	0.796464	0.919745
1.0	0.943291	**0.989872**	0.569097	0.905881	0.906464	0.921476	0.904601	0.889476	0.799604	0.919103
1.5	0.950096	**0.989872**	0.572883	0.905881	0.908637	0.923714	0.905641	0.891597	0.802653	0.918249
2.0	0.956467	**0.989872**	0.572883	0.905881	0.908637	0.923714	0.905641	0.892109	0.803652	0.918249
3.0	**0.963033**	**0.989872**	0.572883	0.905881	0.908637	0.923714	0.905641	0.892109	0.803652	0.918249
4.0	**0.963033**	**0.989872**	0.572883	0.905881	0.908637	0.923714	0.905641	0.892109	0.803652	0.918249
5.0	**0.963033**	**0.989872**	0.572883	0.905881	0.908637	0.923714	0.905641	0.892109	0.803652	0.918249
IR	9.22	10	9.12	9	9.17	9.28	9.08	9.14	9.14	8.79
m	9	9	17	20	24	25	51	99	99	559
dim	92	108	172	200	244	257	514	1004	1004	5472

Table 5: Performance results for the different values of 'p' setting IR fixed at an average value of 9 and varying the number of instances . Classifier: KNN, Evaluation metric: GM, Method: POS, ordered from lowest to highest value of number of instances

Upon examining this chart, a discernible pattern becomes evident. Datasets with fewer instances exhibit higher optimal values for "p," while as the number of instances increases, the optimal "p" values tend to decrease. However, upon closer examination of the chart, it becomes apparent that the number of instances in the minority class is also arranged in ascending order along with the number of instances. Therefore, this observed pattern could potentially be attributed to the number of instances itself, or it might be influenced by the number of instances in the minority class (m). Further investigation is needed to determine the underlying cause of this pattern.

4.3.1.4. *Examining the influence of the number of minority class instances*

To investigate the impact of "m," the datasets and their corresponding performances were arranged in ascending order based on the number of minority instances.



"p"	glass-0-4	glass-0-	shuttle-	glass4.d	glass2.	glass-0-	poker-8	ecoli-0-	ecoli-0-6
0.1	0.9501	0.9796	0.9977	0.9242	0.5577	0.5409	0.9241	0.9328	**0.8770**
0.2	0.9501	0.9847	0.9977	0.9242	0.4762	0.5665	0.9207	0.9539	0.8745
0.3	0.9501	0.9795	0.9977	0.9242	0.5135	0.5323	0.9219	0.9510	0.8704
0.4	0.9501	0.9795	0.9977	0.9242	0.5088	0.5838	**0.9254**	0.9535	0.8641
0.5	0.9501	0.9847	0.9977	0.9242	0.5632	**0.6986**	0.8619	0.9535	0.8606
0.6	0.9501	0.9847	0.9977	0.9216	0.5599	0.6922	0.8636	**0.9563**	0.8606
0.7	0.9433	0.9795	0.9977	0.9216	0.5881	0.6594	0.8654	0.9299	0.8606
0.8	0.9433	0.9847	0.9977	0.9216	**0.7276**	0.5654	0.8677	0.9299	0.8624
0.9	0.9501	0.9847	**1.0000**	0.9216	0.6969	0.5730	0.8705	0.9035	0.8624
1.0	0.9433	**0.9899**	**1.0000**	0.9216	0.5917	0.5691	0.8714	0.9059	0.8624
1.5	0.9501	**0.9899**	**1.0000**	0.9216	0.5588	0.5729	0.8749	0.9059	0.8624
2.0	0.9565	**0.9899**	0.9414	**0.9243**	0.5688	0.5729	0.8394	0.9059	0.8646
3.0	**0.9630**	**0.9899**	0.9414	**0.9243**	0.5711	0.5729	0.8396	0.9059	0.8646
4.0	**0.9630**	**0.9899**	0.9414	**0.9243**	0.5711	0.5729	0.8396	0.9059	0.8646
5.0	**0.9630**	**0.9899**	0.9414	**0.9243**	0.5711	0.5729	0.8396	0.9059	0.8646
IR	9.22	**10**	22	15.47	11.59	9.12	85.88	9	9.09
m	9	9	10	13	17	17	17	20	22

Table 6: Performance results for the different values of 'p'. Classifier: KNN, Evaluation metric: GM, Method: POS, arranged in ascending order of 'm'

The table (*Table 6*) presents the first 9 datasets with the smallest minority class instances. A distinct trend becomes evident: as the number of instances in the minority class grows, the optimal "*p*" value that yields the highest performance diminishes. This observation reinforces the original assumption that the parameter "m" significantly influences the outcome. It can be observed that datasets with fewer than 13 instances in the minority class tend to exhibit "*p*" values greater than 1. On the other hand, datasets with slightly higher "m" values tend to have much smaller "p" values. This suggests that for datasets with a limited number of instances in the minority class (less than 13), higher values of "*p*" are preferred. Conversely, when datasets have a slightly higher count of minority instances, optimal "*p*" values become considerably smaller. In other words, the parameter "*p*" appears to adapt according to the scale of the minority instances, indicating its responsiveness to the characteristics of the dataset.

When the dataset has a higher number of instances in the minority class, these instances are likely to be more representative and, therefore, closer to the original data distribution. In this scenario, applying a smaller perturbation to the generated instances can be beneficial to maintain similarity with the original instances and preserve the minority class's representativeness.

Conversely, when the dataset has fewer instances in the minority class, there may be less diversity and representativeness within that class. Applying a larger perturbation to the generated instances can help introduce more variability and diversity in the synthetic instances, which could be advantageous for enhancing model generalization and addressing the lack of representativeness in the minority class.

Given that only a small number of datasets out of the 28 exhibit a minority class instance count of less than 13, a pragmatic approach is taken to address this scenario. For these datasets with relatively small minority class sizes, it is chosen to empirically set the value of the hyperparameter '*p*' to 5. This decision stems from the understanding that datasets with an insufficient number of minority class instances might benefit from a higher '*p*' value, enabling the generation of more diverse synthetic instances and helping to mitigate the challenge of limited representation.

However, for datasets where the number of minority class instances exceeds this threshold, a more comprehensive analysis is undertaken.

4.3.1.5. Optimal 'p' value for POS

The goal is to discern the optimal value of '*p*' that would lead to superior results in terms of classification performance and addressing class imbalance. This analysis involves systematically experimenting with different '*p*' values across the diverse range of datasets with higher number of minority class instances. By doing so, the study aims to capture the intricate interplay between '*p*,' the underlying data distribution.

"p" values	ecoli-0-6-7_vs_3-5.dat2	ecoli-0-1_vs_2-3-5.dat	ecoli-0-3-4-7_vs_5-6.dat	poker-8-9_vs_6.dat	winequality-white-3-9_vs_5.dat	glass6.dat	new-thyroid1.dat	yeast6.dat	yeast5.dat	yeast-2_vs_4.dat	yeast4.dat	winequality-red-4.dat	glass-0-1-2-3_vs_4-5-6.dat	glass0.dat	glass1.dat	yeast-0-2-5-7-9_vs_3-6-8.dat	yeast-0-2-5-6_vs_3-7-8-9.dat	yeast3.dat	yeast1.dat	page-blocks0.dat2	Mean
0.1	**0.8770**	0.9154	0.9068	0.9555	0.4523	0.8883	0.9647	0.8239	0.9496	0.9087	0.7463	0.5000	0.9237	0.8004	0.7812	**0.9039**	0.7808	0.8719	0.6794	0.8926	0.8261
0.2	0.8745	0.9358	0.9235	**0.9582**	0.4515	0.8883	0.9796	0.8361	0.9468	**0.9299**	0.7507	0.5324	0.9445	0.8076	0.7781	0.8976	0.7787	0.8723	0.6863	0.8987	0.8336
0.3	0.8704	**0.9556**	**0.9238**	0.9368	**0.4869**	**0.9052**	0.9796	0.8342	0.9447	0.9287	0.7791	**0.5892**	0.9445	0.7893	0.7675	0.8967	0.7791	0.8902	0.6902	0.9151	**0.8403**
0.4	0.8641	0.9402	0.8961	0.9392	0.4849	**0.9052**	0.9768	0.8342	0.9583	0.9244	0.7752	0.5457	0.9445	0.7820	**0.7818**	0.8989	0.7854	0.8930	0.6937	0.9189	0.8371
0.5	0.8606	0.9381	0.9215	0.9180	0.4493	0.9024	0.9768	0.8518	**0.9711**	0.9125	**0.7864**	0.5470	0.9445	0.7955	0.7754	0.8888	0.8090	**0.9008**	0.6984	**0.9228**	0.8385
0.6	0.8606	0.9133	0.9235	0.9000	0.4504	0.9050	0.9768	**0.8642**	0.9628	0.9125	0.7658	0.5706	0.9548	0.7917	0.7683	0.8909	**0.8124**	0.8960	**0.7015**	0.9213	0.8371
0.7	0.8606	0.9088	0.9235	0.9023	0.4518	0.9050	0.9768	0.8556	0.9548	0.9157	0.7660	0.5491	0.9548	0.7941	0.7755	0.8864	0.7974	0.8947	0.6924	0.9204	0.8343
0.8	0.8624	0.9065	0.9215	0.9039	0.4521	0.9050	0.9768	0.8610	0.9699	0.9026	0.7477	0.5106	**0.9579**	0.7941	0.7778	0.8895	0.7985	0.8915	0.6913	0.9216	0.8321
0.9	0.8624	0.9065	0.9215	0.9055	0.4526	0.9050	0.9768	0.8635	0.9711	0.9046	0.7428	0.4625	**0.9579**	0.8082	0.7700	0.8895	0.7965	0.8893	0.6920	0.9197	0.8299
1.0	0.8624	0.9065	0.9215	0.9062	0.4535	0.9050	0.9768	0.8496	0.9609	0.9046	0.7449	0.4635	**0.9579**	**0.8083**	0.7740	0.8895	0.7996	0.8905	0.6925	0.9191	0.8293
1.5	0.8624	0.9086	0.9237	0.9096	0.3392	0.9050	**0.9888**	0.8562	0.9499	0.9056	0.7010	0.4678	**0.9579**	0.8040	0.7607	0.8916	0.8027	0.8880	0.6959	0.9182	0.8218
2.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	**0.9888**	0.8426	0.9503	0.9056	0.7015	0.4683	0.9478	0.8040	0.7607	0.8921	0.8037	0.8880	0.6959	0.9182	0.8210
3.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	**0.9888**	0.8256	0.9506	0.9056	0.7015	0.4686	0.9478	0.8040	0.7645	0.8921	0.8037	0.8880	0.6959	0.9182	0.8203
4.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	**0.9888**	0.8256	0.9506	0.9056	0.7015	0.4686	0.9478	0.8040	0.7645	0.8921	0.8037	0.8880	0.6959	0.9182	0.8203
5.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	**0.9888**	0.8256	0.9506	0.9056	0.7015	0.4686	0.9478	0.8040	0.7645	0.8921	0.8037	0.8880	0.6959	0.9182	0.8203
IR	9.09	9.17	9.28	58.4	58.28	6.38	5.14	41.4	32.73	9.08	28.1	29.17	3.2	2	1.82	9.14	9.14	8.1	2.46	8.79	
m	22	24	25	25	25	29	35	35	44	51	51	53	55	70	76	99	99	163	429	559	

Table 7: Performance results for the different values of 'p'. Classifier: KNN, Evaluation metric: GM, Method: POS, arranged in ascending order of 'm' for datasets with more than m = 20

Upon examining this new table (*Table 7*), a distinct trend emerges: the optimal "*p*" values that lead to superior performance across datasets predominantly fall below 1, particularly under the threshold of 0.7. Although there are exceptions like the "*new-thyroid1*" dataset, the consensus aligns with this pattern. Delving deeper into the analysis, it becomes apparent that the average performance scores for various "*p*" values tend to exhibit a positive correlation with values below 0.7 (it can be observed in green in the last column.) Notably, the "*p*" value of **0.3** emerges as a standout choice, consistently delivering the most favorable outcomes across a diverse array of datasets.

Given these findings, we will establish a specific approach for selecting the "*p*" value based on the characteristics of the datasets. For datasets with a larger number of minority class instances "m", using smaller values of "p" leads to a higher magnitude of perturbation. In practical terms, this means that the synthetic instances created through oversampling will undergo more substantial alterations, effectively introducing a greater level of diversity into the dataset. This is beneficial for cases where there is a larger pool of minority instances to work with. The additional diversity introduced by higher perturbation can help prevent overfitting and improve generalization by creating synthetic instances that capture a wider range of potential instances. In these cases, we will set the "*p*" value to 0.3.

On the other hand, for datasets with a smaller number of minority instances, opting for higher values of "p" is more appropriate, as previously mentioned, we will fix the "*p*" value at 5. This approach aligns with the goal of minimizing perturbation while maintaining data fidelity for instances that are already scarce. In this scenario, a higher "p" results in a lower perturbation magnitude. This approach is suitable because there may be limited representativeness in the minority class due to the smaller number of instances. By applying a more conservative level of perturbation, the synthetic instances can be generated with caution, ensuring that they remain more aligned with the original data distribution. This helps prevent the generation of instances that are too dissimilar to the existing minority instances and thus maintains a coherent and representative class distribution.

In summary, tailoring the "*p*" value to the specific dataset characteristics acknowledges the nuanced interplay between data quantity, class imbalance, and the need for diversity. This strategic adaptation seeks to optimize the performance of the Perturbation-based Oversampling (POS) method across the entire spectrum of datasets.

4.3.2. POS 1.0

The entirety of these investigations and experiments has been meticulously undertaken again to verify if the observations hold in the context of POS 1.0 version.

4.3.2.1. Fixed number of instances for POS 1.0 method

Regarding the experiments carried out to examine the relationship between the imbalance ratio (IR) and the '*p*' values, varying fixed number of instances, consistent observations with the POS method were obtained. Tables showing these results are found in the Appendix. As it happens with the original POS method, the relation is not given with the IR, but is given with the number of instances in the minority class.

4.3.2.2. *Fixed IR for POS 1.0 method*

"p" values	glass-0-4_vs_5.dat	glass-0-6_vs_5.dat	glass-0-1-5_vs_2.dat	ecoli-0-3-4_vs_5.dat	ecoli-0-1_vs_2-3-5.dat	ecoli-0-3-4-7_vs_5-6.dat	yeast-2_vs_4.dat	yeast-0-2-5-7-9_vs_3-6-8.dat	yeast-0-2-5-6_vs_3-7-8-9.dat	page-blocks0.dat
0.1	0.9628	0.7899	0.4783	0.9116	0.8803	0.9157	0.8850	0.8994	0.7488	0.8549
0.2	0.9628	0.7899	0.4632	0.9116	0.9019	0.9134	0.8821	0.8979	0.7659	0.8681
0.3	0.9565	0.7899	0.4586	0.9384	0.8978	0.9112	0.8810	**0.9004**	0.7772	0.8759
0.4	0.9565	0.7899	0.4540	0.9384	**0.9196**	0.9090	0.9023	0.8953	0.7896	0.8854
0.5	0.9565	0.7899	0.4849	0.9384	**0.9196**	0.9067	0.8983	0.8923	0.7937	0.9024
0.6	0.9565	0.7899	0.4831	0.9384	**0.9196**	0.9067	**0.9078**	0.8910	0.7903	0.9142
0.7	0.9565	0.7899	0.5169	**0.9620**	**0.9196**	**0.9281**	0.9060	0.8858	0.8024	0.9198
0.8	0.9565	0.7899	0.5875	0.9591	0.9154	0.9259	0.8995	0.8870	**0.8037**	0.9191
0.9	0.9565	0.7899	0.5934	0.9591	0.9133	0.9237	0.9135	0.8864	0.7985	0.9209
1.0	0.9565	0.7899	0.5905	0.9323	0.9088	0.9215	0.9115	0.8859	0.8017	**0.9214**
1.5	0.9565	**0.9899**	**0.6051**	0.9059	0.9043	0.9237	0.9035	0.8895	0.8016	0.9192
2.0	0.9565	**0.9899**	0.5722	0.9059	0.9086	0.9237	0.9056	0.8921	0.8032	0.9192
3.0	0.9565	**0.9899**	0.5729	0.9059	0.9086	0.9237	0.9056	0.8921	0.8037	0.9192
4.0	**0.9630**	**0.9899**	0.5729	0.9059	0.9086	0.9237	0.9056	0.8921	0.8037	0.9192
5.0	**0.9630**	**0.9899**	0.5729	0.9059	0.9086	0.9237	0.9056	0.8921	0.8037	0.9192
IR	9.22	10	9.12	9	9.17	9.28	9.08	9.14	9.14	8.79
m	9	9	17	20	24	25	51	99	99	559
dim	92	108	172	200	244	257	514	1004	1004	5472

Table 8: Performance results for the different values of 'p' setting IR fixed at an average value of 9 and varying the number of instances. Classifier: KNN, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of number of instances

Much like the trend observed in the POS method, it can be noticed on the table (*Table 8*) a trend where the "*p*" value decreases as the number of instances increases. However, there is a slight difference between the two cases. In the previous method, the "*p*" value gradually decreases until reaching a minimum of 0.1. In contrast, in this scenario, the minimum "*p*" value observed is 0.3, suggesting a less pronounced reduction. This discrepancy warrants further investigation and analysis to better understand the underlying dynamics.

4.3.2.3. *Examining the influence of the number of minority class instances for POS 1.0*

Firstly, similar to the previous studies, we conducted an analysis on datasets with the fewest minority instances. The provided table (*Table 9*) illustrates consistent observations. Just as before, datasets with a smaller number of minority instances, denoted as 'm', exhibit improved performance with higher values of the hyperparameter 'p'. Nevertheless, in this scenario, the decline in the 'p' value is more gradual. In the earlier experiment involving the POS method, datasets with 'm' equal to or less than 13 demonstrated a preference for 'p' values greater than 1. However, in this experiment, even datasets with 17 instances in the minority class exhibit a preference for 'p' values below 1.

Despite this, the decision made in the POS method that datasets with 'm' less than or equal to 13 should have a 'p' value set to 5 will be maintained. This threshold will be not altered to 17 in this case, as datasets with 17 instances in the minority class show a preference for 'p' values near 1, rather than 5.

"p" values	glass-0-4_vs_5	glass-0-6_vs_5	shuttle-6_vs_2-3	glass4	glass2	glass-0-1-5_vs_2	poker-8_vs_6	ecoli-0-3-4_vs_5	6-7_vs_3-5
0.1	0.9628	0.7899	0.9414	0.8462	0.2558	0.4783	0.9640	0.9116	0.8651
0.2	0.9628	0.7899	0.9414	0.8436	0.3712	0.4632	0.9640	0.9116	0.8887
0.3	0.9565	0.7899	0.9414	0.8415	0.3682	0.4586	0.9637	0.9384	0.8834
0.4	0.9565	0.7899	0.9414	0.8768	0.3639	0.4540	0.9643	0.9384	**0.9022**
0.5	0.9565	0.7899	0.9414	0.8768	0.3624	0.4849	0.9640	0.9384	0.8979
0.6	0.9565	0.7899	0.9414	0.8742	0.4598	0.4831	0.9646	0.9384	0.8883
0.7	0.9565	0.7899	0.9414	0.8690	0.4970	0.5169	0.9657	**0.9620**	0.8654
0.8	0.9565	0.7899	0.9414	0.8690	0.4909	0.5875	**0.9671**	0.9591	0.8606
0.9	0.9565	0.7899	0.9414	0.8690	0.4894	0.5934	0.9405	0.9591	0.8606
1.0	0.9565	0.7899	0.9414	0.8690	0.4872	0.5905	0.9409	0.9323	0.8606
1.5	0.9565	**0.9899**	**0.9824**	**0.9243**	**0.5711**	**0.6051**	0.8752	0.9059	0.8624
2.0	0.9565	**0.9899**	**0.9824**	**0.9243**	**0.5711**	0.5722	0.8752	0.9059	0.8646
3.0	0.9565	**0.9899**	**0.9824**	**0.9243**	**0.5711**	0.5729	0.8396	0.9059	0.8646
4.0	**0.9630**	**0.9899**	**0.9824**	**0.9243**	**0.5711**	0.5729	0.8396	0.9059	0.8646
5.0	**0.9630**	**0.9899**	**0.9824**	**0.9243**	**0.5711**	0.5729	0.8396	0.9059	0.8646
IR	9.22	10	22	15.47	11.59	9.12	85.88	9	9.09
m	9	9	10	13	17	17	17	20	22

Table 9: Performance results for the different values of 'p'. Classifier: KNN, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of 'm'

4.3.2.4. Optimal 'p' value for POS 1.0

As analyzed briefly in section 4.3.2.2. and 4.3.2.3, at first glance the decline in the '*p*' value is more gradual and the minimum "*p*" value observed is 0.3, suggesting a less pronounced reduction. To analyze this, the same experiment as in section 4.3.1.5 was conducted but for the POS 1.0 method.

"p" values	ecoli-0-6-7_vs_3-5	ecoli-0-1_vs_2-3-5	ecoli-0-3-4-7_vs_5-6	poker-8-9_vs_6	winequality-white-3-9_vs_5	glass6	new-thyroid1	yeast6	yeast5	yeast-2_vs_4	yeast4	winequality-red-4	glass-0-1-2-3_vs_4-5-6	glass0	glass1	yeast-0-2-5-7-9_vs_3-6-8	yeast-0-2-5-6_vs_3-7-8-9	yeast3	yeast1	page-blocks0	MEAN
0.1	0.8651	0.8803	0.9157	0.9914	**0.3769**	0.8578	0.9338	0.7509	0.9271	0.8850	0.5814	0.2795	0.8958	**0.8180**	0.7755	0.8994	0.7488	0.8478	0.6214	0.8549	0.7853
0.2	0.8887	0.9019	0.9134	0.9904	0.3760	0.8578	0.9499	0.7718	0.9410	0.8821	0.5924	0.4571	0.9017	0.8142	0.7791	0.8979	0.7659	0.8552	0.6423	0.8681	0.8023
0.3	0.8834	0.8978	0.9112	0.9904	0.3751	0.8933	0.9499	0.8234	0.9396	0.8810	0.6242	0.4795	0.8986	0.8148	0.7858	**0.9004**	0.7772	0.8750	0.6589	0.8759	0.8118
0.4	**0.9022**	**0.9196**	0.9090	0.9904	0.3747	0.8908	0.9796	0.8210	0.9386	0.9023	0.6640	0.5270	0.9031	0.8074	0.7825	0.8953	0.7896	0.8832	0.6757	0.8854	0.8221
0.5	0.8979	**0.9196**	0.9067	0.9914	0.3747	0.8908	0.9768	0.8202	0.9520	0.8983	0.7125	0.5235	0.9136	0.8140	0.7825	0.8923	0.7937	0.8871	0.6837	0.9024	0.8267
0.6	0.8883	**0.9196**	0.9067	0.9917	0.3747	**0.9078**	0.9768	0.8199	0.9514	0.9078	0.7242	0.5341	0.9337	0.8140	**0.7878**	0.8910	0.7903	0.8962	0.6934	0.9142	0.8312
0.7	0.8654	**0.9196**	**0.9281**	**0.9935**	0.3747	0.9078	**0.9916**	0.8176	**0.9620**	0.9060	0.7340	0.5479	0.9337	0.7970	0.7771	0.8858	0.8024	**0.8984**	**0.6969**	0.9198	**0.8330**
0.8	0.8606	0.9154	0.9259	0.9735	0.3747	0.9050	**0.9916**	0.8345	0.9616	0.8995	0.7321	**0.5487**	0.9548	0.7974	0.7574	0.8870	**0.8037**	0.8932	0.6949	0.9191	0.8315
0.9	0.8606	0.9133	0.9237	0.9528	0.3744	0.9050	**0.9916**	0.8502	0.9616	**0.9135**	0.7330	0.5460	0.9548	0.7924	0.7672	0.8864	0.7985	0.8910	0.6963	0.9209	0.8317
1.0	0.8606	0.9088	0.9215	0.9292	0.3747	0.9050	**0.9916**	**0.8574**	0.9616	0.9115	**0.7482**	0.5344	0.9548	0.7887	0.7636	0.8859	0.8017	0.8892	0.6944	**0.9214**	0.8302
1.5	0.8624	0.9043	0.9237	0.9093	0.3388	0.9050	0.9768	**0.8574**	0.9503	0.9035	0.7152	0.4652	**0.9579**	0.8040	0.7645	0.8895	0.8016	0.8873	0.6959	0.9192	0.8216
2.0	0.8646	0.9086	0.9237	0.9102	0.3392	0.9050	0.9888	**0.8574**	0.9503	0.9056	0.7015	0.4678	0.9478	0.8040	0.7645	0.8921	0.8032	0.8880	0.6947	0.9192	0.8218
3.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	0.9888	0.8256	0.9506	0.9056	0.7015	0.4686	0.9478	0.8040	0.7645	0.8921	0.8037	0.8880	0.6959	0.9192	0.8204
4.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	0.9888	0.8256	0.9506	0.9056	0.7015	0.4686	0.9478	0.8040	0.7645	0.8921	0.8037	0.8880	0.6959	0.9192	0.8204
5.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	0.9888	0.8256	0.9506	0.9056	0.7015	0.4686	0.9478	0.8040	0.7645	0.8921	0.8037	0.8880	0.6959	0.9192	0.8204
IR	9.09	9.17	9.28	58.4	58.28	6.38	5.14	41.4	32.73	9.08	28.1	29.7	3.2	2	1.82	9.14	9.14	8.1	2.46	8.79	
m	22	24	25	25	25	29	35	35	44	51	51	53	55	70	76	99	99	163	429	559	

Table 10: Performance results for the different values of 'p'. Classifier: KNN, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of 'm' for datasets with more than m = 20

As evident from the table (*Table 10*), the optimal '*p*' values for the POS 1.0 method are not as low as those observed with the POS method. These optimal '*p*' values fall within a central range, typically between 1 and 0.5, as indicated by the prominently green-shaded area in the 'MEAN' column. Among these values, the specific '*p*' value of 0.7 consistently stands out as the one that yields the highest average performance across all the datasets.

This pattern aligns with the logic that the POS 1.0 method introduces perturbations not only to individual features but to the entire instances, resulting in a more extensive perturbation overall. Consequently, a much lower '*p*' value is not necessary to achieve a higher degree of perturbation. Instead, the moderate '*p*' values inherently yield a significant level of perturbation due to the comprehensive nature of the method's perturbation mechanism.

Taking into consideration all of these studies and analyses, a general rule is established for determining the hyperparameter value in POS 1.0. Specifically, for datasets with a notably small number of instances in the minority class, the optimal value of 'p' is set at 5, representing higher values. Conversely, for datasets featuring a larger number of instances in the minority class, the value of 0.7 is designated as the preferred choice for 'p'.

This approach accounts for the observed behavior and patterns across various experiments, wherein datasets with differing characteristics demonstrated distinct preferences for 'p' values. By tailoring the choice of 'p' based on the characteristics of the dataset, this rule aims to strike an optimal balance between introducing diversity through perturbation and maintaining the fidelity of the minority class distribution. This empirically derived strategy is poised to enhance the performance of the POS 1.0 method across a diverse array of imbalanced datasets.

4.3.3. POS 2.0

Once again, identical experiments have been meticulously conducted, mirroring those carried out for both the POS method and POS 1.0. The aim of these replications is to confirm the consistency of the findings across all versions. By subjecting the new approach, POS 2.0, to the same experimental studies, we aim to establish its robustness and ascertain whether the insights gained from the earlier versions hold true in this new context as well. This rigorous approach ensures that any conclusions drawn about the behavior of '*p*' in relation to dataset characteristics are not merely coincidental but rather grounded in the inherent nature of the approach itself.

4.3.3.1. Fixed number of instances for POS 2.0 method

Just as observed previously, no distinct patterns emerge when examining the relationship between IR and the optimal '*p*' values. As the results are similar, the tables can be found in the Appendix. This consistent lack of clear correlation strongly suggests that the phenomenon is not a matter of chance but rather a recurring characteristic that spans across all three versions of the POS method. This repetition reinforces the concept that the interaction between '*p*,' IR, and other variables is inherently complex and not easily explained by straightforward relationships.

4.3.3.2. Fixed IR for POS 2.0 method

We conducted experiments with POS 2.0 using a selection of datasets characterized by a consistent IR of approximately 9. The results (*Table 11*) were arranged in ascending order of number of instances, following the methodology employed in the previous POS versions. Remarkably, the outcomes for POS 2.0 closely mirror those of POS 1.0. The behavior of the hyperparameter '*p*' is quite similar, featuring a gradual descent. Interestingly, in this instance, the lowest optimal 'p' value is even greater than that of POS 1.0 (p = 0.3) at 0.4, signifying a more gradual decrease in '*p*' values.

"p" values	glass-0-4_vs_5	glass-0-6_vs_5	glass-0-1-5_vs_2	ecoli-0-3-4_vs_5	ecoli-0-1_vs_2-3-5	ecoli-0-3-4-7_vs_5-6	yeast-2_vs_4	yeast-0-2-5-7-9_vs_3-6-8	yeast-0-2-5-6_vs_3-7-8-9	page-blocks0
0.1	0.7351	0.6207	0.2291	0.9144	0.8364	0.8964	0.8208	0.8891	0.7039	0.8271
0.2	0.7351	0.6207	0.2291	0.9352	0.8560	0.8964	0.8276	0.8947	0.7111	0.8282
0.3	0.7205	0.6207	0.2291	0.9352	0.8560	0.8964	0.8276	0.8941	0.7180	0.8312
0.4	0.7205	0.6742	0.2291	**0.9380**	0.8808	**0.9237**	0.8411	0.9000	0.7292	0.8458
0.5	0.7205	0.6669	0.2272	0.9356	0.9252	0.9141	0.8725	0.8973	0.7515	0.8586
0.6	0.7205	0.6669	0.2253	0.9356	**0.9466**	0.9119	0.8928	0.9012	0.7974	0.8833
0.7	0.7205	0.7186	0.2233	0.9328	0.9259	0.9025	0.8998	**0.9026**	**0.8033**	0.9092
0.8	0.8997	0.9239	0.2155	0.9328	0.9213	0.8985	**0.9185**	0.8969	0.7877	0.9174
0.9	0.9565	0.9795	0.2602	0.9356	0.9085	0.9234	0.9176	0.8905	0.7893	**0.9225**
1.0	0.9565	0.9795	0.2602	0.9331	0.9108	0.9215	0.9144	0.8900	0.7997	0.9202
1.5	0.9437	0.9847	**0.6681**	0.9059	0.9086	0.9237	0.9045	0.8900	0.8021	0.9192
2.0	0.9501	**0.9899**	0.5692	0.9059	0.9086	0.9237	0.9056	0.8921	0.8032	0.9192
3.0	**0.9630**	**0.9899**	0.5729	0.9059	0.9086	0.9237	0.9056	0.8921	0.8032	0.9192
4.0	**0.9630**	**0.9899**	0.5729	0.9059	0.9086	0.9237	0.9056	0.8921	0.8032	0.9192
5.0	**0.9630**	**0.9899**	0.5729	0.9059	0.9086	0.9237	0.9056	0.8921	0.8032	0.9192
IR	9.22	10	9.12	9	9.17	9.28	9.08	9.14	9.14	8.79
m	9	9	17	20	24	25	51	99	99	559
dim	92	108	172	200	244	257	514	1004	1004	5472

Table 11: Performance results for the different values of 'p' setting IR fixed at an average value of 9 and varying the number of instances. Classifier: KNN, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of number of instances

4.3.3.3. Examining the influence of the number of minority class instances for POS 2.0

Following the same approach as in the previous methods, we organized the initial datasets with the lowest number of minority instances (m) in ascending order.

Observing the results presented in *Table 12*, a pattern analogous to that of the POS 1.0 method emerges. For the datasets with 17 or fewer instances in the minority class, the optimal '*p*' value is greater than 1, beyond which the optimal '*p*' value experiences a substantial decline. The first datasets, with the least number of minority class instances, exhibit an almost identical pattern to that of the POS method, making it apparent that retaining value of 5 for '*p*' is warranted, as with POS.

"p" values	glass-0-4_vs_5	glass-0-6_vs_5	shuttle-6_vs_2-3	glass4	glass2	glass-0-1-5_vs_2	poker-8_vs_6	ecoli-0-3-4_vs_5	ecoli-0-6-7_vs_3-5
0.1	0.7351	0.6207	0.6828	0.5370	0.1140	0.2291	0.4307	0.9144	0.7809
0.2	0.7351	0.6207	0.6828	0.5822	0.1140	0.2291	0.7145	0.9352	0.8702
0.3	0.7205	0.6207	0.6828	0.6267	0.1140	0.2291	0.7950	0.9352	0.8967
0.4	0.7205	0.6742	0.7414	0.6253	0.1140	0.2291	0.8241	**0.9380**	0.8684
0.5	0.7205	0.6669	0.7414	0.7629	0.1140	0.2272	0.9080	0.9356	**0.8924**
0.6	0.7205	0.6669	0.7414	0.7566	0.2265	0.2253	**0.9247**	0.9356	0.8847
0.7	0.7205	0.7186	0.9414	0.8032	0.2265	0.2233	0.9167	0.9328	0.8728
0.8	0.8997	0.9239	0.9414	0.8690	0.2716	0.2155	0.8421	0.9328	0.8683
0.9	0.9565	0.9795	0.9414	0.8664	0.2701	0.2602	0.8478	0.9356	0.8864
1.0	0.9565	0.9795	**1.0000**	0.9190	0.2686	0.2602	0.8563	0.9331	0.8837
1.5	0.9437	0.9847	**1.0000**	0.9189	**0.5782**	**0.6681**	0.8720	0.9059	0.8829
2.0	0.9501	**0.9899**	**1.0000**	**0.9243**	0.5644	0.5692	0.8752	0.9059	0.8646
3.0	**0.9630**	**0.9899**	0.9414	**0.9243**	0.5711	0.5729	0.8396	0.9059	0.8646
4.0	**0.9630**	**0.9899**	0.9414	**0.9243**	0.5711	0.5729	0.8396	0.9059	0.8646
5.0	**0.9630**	**0.9899**	0.9414	**0.9243**	0.5711	0.5729	0.8396	0.9059	0.8646
IR	9.22	10	22	15.47	11.59	9.12	85.88	9	9.09
m	9	9	10	13	17	17	17	20	22

Table 12: Performance results for the different values of 'p'. Classifier: KNN, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of 'm'

4.3.3.4. Optimal 'p' value for POS 2.0

The final phase of the analysis concerning the optimal '*p*' values is to study those datasets with higher number of minority instances, following the same approach applied to the preceding methodologies.

During this phase, as depicted in *Table 13*, the performance closely parallels that of the POS 1.0 method. Within this context, peak performance is attained with '*p*' values ranging from 1.5 to 0.7, as visibly indicated by the green-highlighted range in the 'MEAN' column, with the optimal value being 0.9. Furthermore, it's evident that performance experiences a notable decline when '*p*' values fall below 0.5. This underscores the criticality of in-depth analysis regarding the selection of '*p*' values for each respective method.

"p" values	ecoli-0-6-7_vs_3-5	ecoli-0-1_vs_2-3-5	ecoli-0-3-4-7_vs_5-6	poker-8-9_vs_6	winequality-white-3-9_vs_5	glass6	new-thyroid1	yeast6	yeast5	yeast-2_vs_4	yeast4	winequality-red-4	glass-0-1-2-3_vs_4-5-6	glass0	glass1	yeast-0-2-5-7-9_vs_3-6-8	yeast-0-2-5-6_vs_3-7-8-9	yeast3	yeast1	page-blocks0	Mean
0.1	0.7809	0.8364	0.8964	0.6887	0.0000	0.8195	0.9056	0.6864	0.8178	0.8208	0.2785	0.0000	0.8727	**0.8255**	0.7708	0.8891	0.7039	0.8193	0.6134	0.8271	0.6926
0.2	0.8702	0.8560	0.8964	0.8144	0.0893	0.8385	0.9056	0.6864	0.8178	0.8276	0.3046	0.0000	0.8671	0.8154	0.7646	0.8947	0.7111	0.8193	0.6127	0.8282	0.7110
0.3	**0.8967**	0.8560	0.8964	0.8833	0.0893	0.8601	0.9217	0.6853	0.8308	0.8276	0.3243	0.0000	0.8671	0.8117	0.7547	0.8941	0.7180	0.8255	0.6175	0.8312	0.7196
0.4	0.8684	0.8808	**0.9237**	0.9085	0.0893	0.8769	0.9527	0.7151	0.8418	0.8411	0.3756	0.0000	0.9174	0.8045	0.7663	0.9000	0.7292	0.8424	0.6251	0.8458	0.7352
0.5	0.8924	0.9252	0.9141	**0.9153**	0.1263	0.9078	0.9527	0.7128	0.8522	0.8725	0.4788	0.0630	0.9139	0.8120	0.7605	0.8973	0.7515	0.8659	0.6648	0.8586	0.7569
0.6	0.8847	**0.9466**	0.9119	0.9115	**0.5195**	**0.9268**	0.9649	0.7088	0.8783	0.8928	0.6102	0.2704	0.9447	0.8049	**0.7853**	0.9012	0.7974	0.8739	0.6815	0.8833	0.8049
0.7	0.8728	0.9259	0.9025	0.8965	0.4938	0.9242	**0.9824**	0.7880	0.9593	0.8998	0.6905	0.3919	0.9447	0.8112	0.7685	**0.9026**	**0.8033**	**0.9005**	0.6889	0.9092	0.8228
0.8	0.8683	0.9213	0.8985	0.9071	0.4903	0.9242	0.9796	0.8319	**0.9796**	**0.9185**	0.7721	0.4903	0.9588	0.8116	0.7721	0.8969	0.7877	0.8868	0.6936	0.9174	0.8353
0.9	0.8864	0.9085	0.9234	0.8944	0.5160	0.9050	0.9768	0.8609	0.9779	0.9176	**0.7925**	0.5243	**0.9481**	0.7899	0.7613	0.8905	0.7893	0.8864	**0.6958**	**0.9225**	**0.8384**
1.0	0.8837	0.9108	0.9215	0.8996	0.5141	0.9024	0.9739	**0.8724**	0.9674	0.9144	0.7801	**0.5434**	0.9378	0.7860	0.7584	0.8900	0.7997	0.8868	0.6919	0.9202	0.8377
1.5	0.8829	0.9086	0.9237	0.9077	0.4267	0.9050	0.9739	0.8518	0.9626	0.9045	0.7094	0.4656	0.9582	0.7811	0.7585	0.8900	0.8021	0.8877	0.6879	0.9192	0.8254
2.0	0.8646	0.9086	0.9237	0.9105	0.3392	0.9050	0.9739	0.8577	0.9499	0.9056	0.7012	0.4679	0.9582	0.7811	0.7586	0.8921	0.8032	0.8880	0.6879	0.9192	0.8198
3.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	0.9739	0.8256	0.9506	0.9056	0.7015	0.4686	0.9582	0.7811	0.7586	0.8921	0.8032	0.8880	0.6891	0.9192	0.8184
4.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	0.9739	0.8256	0.9506	0.9056	0.7015	0.4686	0.9582	0.7811	0.7586	0.8921	0.8032	0.8880	0.6891	0.9192	0.8184
5.0	0.8646	0.9086	0.9237	0.9105	0.3394	0.9050	0.9739	0.8256	0.9506	0.9056	0.7015	0.4686	0.9582	0.7811	0.7586	0.8921	0.8032	0.8880	0.6891	0.9192	0.8184
IR	9.09	9.17	9.28	58.4	58.28	6.38	5.14	41.4	32.73	9.08	28.1	29.17	3.2	2	1.82	9.14	9.14	8.1	2.46	8.79	
m	22	24	25	25	25	29	35	35	44	51	51	53	55	70	76	99	99	163	429	559	

Table 13: Performance results for the different values of 'p'. Classifier: KNN, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of 'm' for datasets with more than m = 20

Upon comprehensive evaluation and analysis of the datasets, we have obtained the optimal '*p*' values for each method in the context of varying dataset characteristics. Notably, the optimal '*p*' values exhibit a consistent trend across the methods for datasets with differing numbers of minority class instances. However, the precise values differ among the methods due to their unique mechanisms and approaches (*Table 14*).

	Low 'm'	***High 'm'***
POS	5	0.3
POS 1.0	5	0.7
POS 2.0	5	0.9

Table 14: optimal 'p' values for each method

This distinct selection of '*p*' values for each method highlights the necessity of careful evaluation and customization, even when working with similar oversampling techniques.

Once the optimal values for the proper tunning of hyperparameter *'p'* have been found a comprehensive comparison of the three methods POS, POS 1.0 and POS 2.0 is carried out. For each method, we assign the corresponding optimized '*p*' value and compare our approach against established oversampling and undersampling techniques such as Random Under-Sampling (RUS), Clustered Nearest Neighbors (CNN), One-Sided Selection (OSS), Tomek Links (TL), Neighbor Cleaning Rule (NCR), Random Over-Sampling (ROS), and Synthetic Minority Over-Sampling Technique

(SMOTE). The intent behind this comparison is to demonstrate the superiority of our methodology.

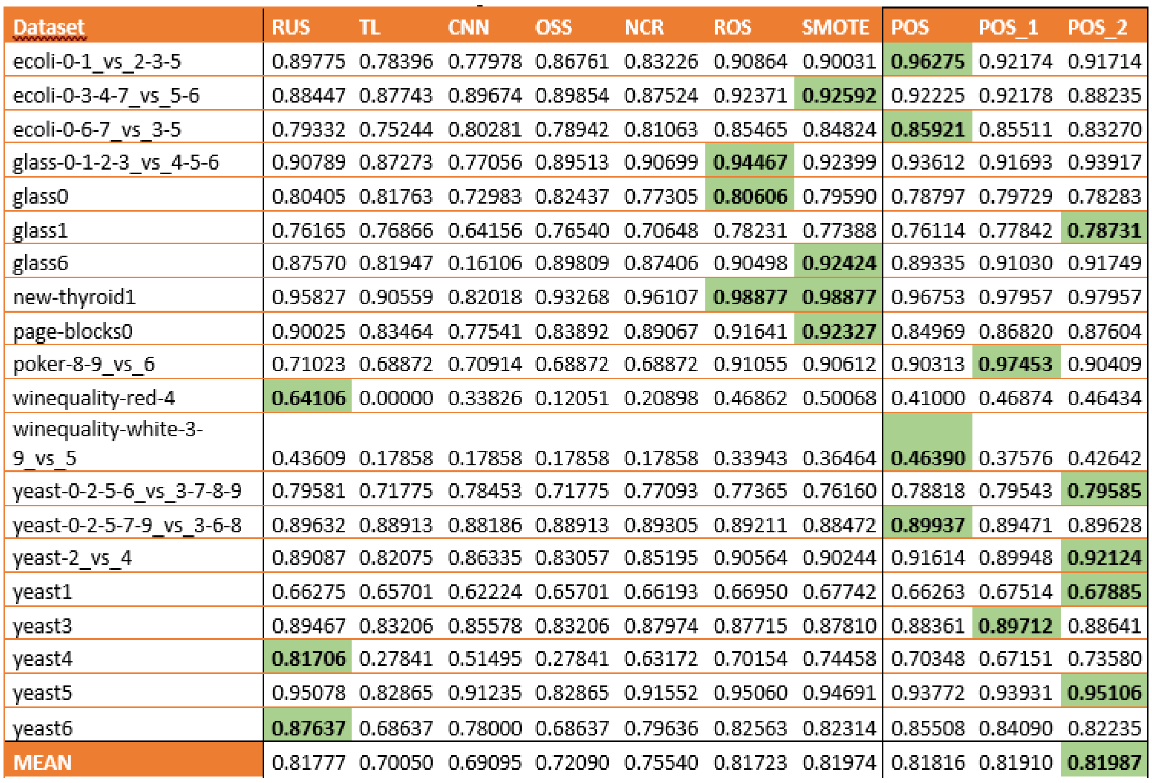
4.4. KNN Classifier - Phase 2: Method Comparative Analysis

To substantiate the claim that the POS methods or his variations have a better performance, we employ diverse evaluation metrics, ensuring a thorough assessment of stability and performance across various scenarios.

Four comparisons have been conducted, one for each evaluation metric. The metrics employed are the Geometric Mean (GM), F1 score, Area Under the Receiver Operating Characteristic Curve (AUC), and Area Under the Precision-Recall Curve (AUCPR). The datasets with number of instances in the minority class exceeding 20 has been evaluated. The performance for each method has been obtained and assessed across all datasets. Performance for each dataset and method is an average of the performance achieved across the 5-fold validation.

By utilizing this spectrum of evaluation measures, we ensure a comprehensive evaluation that verifies the effectiveness of the method in all situations.

4.4.1. Geometric Mean Comparison



Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.89775	0.78396	0.77978	0.86761	0.83226	0.90864	0.90031	**0.96275**	0.92174	0.91714
ecoli-0-3-4-7_vs_5-6	0.88447	0.87743	0.89674	0.89854	0.87524	0.92371	**0.92592**	0.92225	0.92178	0.88235
ecoli-0-6-7_vs_3-5	0.79332	0.75244	0.80281	0.78942	0.81063	0.85465	0.84824	**0.85921**	0.85511	0.83270
glass-0-1-2-3_vs_4-5-6	0.90789	0.87273	0.77056	0.89513	0.90699	**0.94467**	0.92399	0.93612	0.91693	0.93917
glass0	0.80405	0.81763	0.72983	0.82437	0.77305	**0.80606**	0.79590	0.78797	0.79729	0.78283
glass1	0.76165	0.76866	0.64156	0.76540	0.70648	0.78231	0.77388	0.76114	0.77842	**0.78731**
glass6	0.87570	0.81947	0.16106	0.89809	0.87406	0.90498	**0.92424**	0.89335	0.91030	0.91749
new-thyroid1	0.95827	0.90559	0.82018	0.93268	0.96107	**0.98877**	**0.98877**	0.96753	0.97957	0.97957
page-blocks0	0.90025	0.83464	0.77541	0.83892	0.89067	0.91641	**0.92327**	0.84969	0.86820	0.87604
poker-8-9_vs_6	0.71023	0.68872	0.70914	0.68872	0.68872	0.91055	0.90612	0.90313	**0.97453**	0.90409
winequality-red-4	**0.64106**	0.00000	0.33826	0.12051	0.20898	0.46862	0.50068	0.41000	0.46874	0.46434
winequality-white-3-9_vs_5	0.43609	0.17858	0.17858	0.17858	0.17858	0.33943	0.36464	**0.46390**	0.37576	0.42642
yeast-0-2-5-6_vs_3-7-8-9	0.79581	0.71775	0.78453	0.71775	0.77093	0.77365	0.76160	0.78818	0.79543	**0.79585**
yeast-0-2-5-7-9_vs_3-6-8	0.89632	0.88913	0.88186	0.88913	0.89305	0.89211	0.88472	**0.89937**	0.89471	0.89628
yeast-2_vs_4	0.89087	0.82075	0.86335	0.83057	0.85195	0.90564	0.90244	0.91614	0.89948	**0.92124**
yeast1	0.66275	0.65701	0.62224	0.65701	0.66193	0.66950	0.67742	0.66263	0.67514	**0.67885**
yeast3	0.89467	0.83206	0.85578	0.83206	0.87974	0.87715	0.87810	0.88361	**0.89712**	0.88641
yeast4	**0.81706**	0.27841	0.51495	0.27841	0.63172	0.70154	0.74458	0.70348	0.67151	0.73580
yeast5	0.95078	0.82865	0.91235	0.82865	0.91552	0.95060	0.94691	0.93772	0.93931	**0.95106**
yeast6	**0.87637**	0.68637	0.78000	0.68637	0.79636	0.82563	0.82314	0.85508	0.84090	0.82235
MEAN	0.81777	0.70050	0.69095	0.72090	0.75540	0.81723	0.81974	0.81816	0.81910	**0.81987**

Table 15: Performance result table for the different undersampling and oversampling methods. Classifier: KNN, Evaluation metric: GM

As evident from the performance results table of the methods (*Table 15*), it is apparent that the methods examined in this study, POS, POS 1.0, and POS 2.0, exhibit strong performance. In most cases, they achieve the best results, although methods such as SMOTE, ROS and RUS (despite the random nature of these last two) also demonstrate impressive performance. In this context, the method displaying the highest performance is one of the novel approaches implemented in this study, **POS 2.0**, obtaining an average result across datasets of **0.81987.** Although following it closely is the method **SMOTE** with a performance of **0.81974**. The next two best methods in performance are the last two methods studied in this study: **POS 1.0** with a performance of **0.81910** and **POS** with a performance of **0.81816**. This showcases their remarkable performance when the appropriate value of '*p*' is selected.

4.4.2. F1- Score Comparison

Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.79267	0.75556	0.65483	0.85556	0.78349	0.74676	0.74917	**0.86496**	0.82323	0.78323
ecoli-0-3-4-7_vs_5-6	0.63177	0.80848	0.80000	0.83071	0.79030	0.72165	0.73263	**0.83212**	0.82727	0.72337
ecoli-0-6-7_vs_3-5	0.46443	0.69429	0.74476	0.71762	0.68667	0.67186	0.68514	0.74242	**0.75641**	0.72571
glass-0-1-2-3_vs_4-5-6	0.84063	0.83168	0.70917	0.85645	0.83933	0.89039	0.87881	0.90069	0.88556	**0.91021**
glass0	0.73240	0.74446	0.64308	**0.75203**	0.70151	0.73008	0.72891	0.69984	0.72538	0.71106
glass1	0.69889	0.71257	0.55424	0.70827	0.66628	0.72105	**0.74729**	0.69112	0.71151	0.71787
glass6	0.72381	0.73758	0.10159	0.81667	0.77809	0.79124	0.81429	**0.81688**	**0.81688**	0.86173
new-thyroid1	0.91385	0.88923	0.79879	0.89875	0.92718	0.94833	0.94833	0.94051	**0.95795**	**0.95795**
page-blocks0	0.68822	0.77415	0.69853	0.77905	0.75967	0.78156	0.77160	0.79405	0.79763	**0.79828**
poker-8-9_vs_6	0.06742	0.63333	0.50168	0.63333	0.63333	0.79394	0.64785	0.56619	**0.79627**	0.56294
winequality-red-4	0.11806	0.00000	0.19236	0.06410	0.07964	0.16244	0.13944	**0.19595**	0.14933	0.14587
winequality-white-3-9_vs_5	0.11034	0.11429	0.11667	0.11429	0.11429	0.17714	0.08244	**0.22460**	0.20889	0.17512
yeast-0-2-5-6_vs_3-7-8-9	0.49509	0.62871	0.62779	**0.62878**	0.61493	0.50063	0.44760	0.54805	0.52877	0.52055
yeast-0-2-5-7-9_vs_3-6-8	0.71288	**0.81438**	0.76274	0.81435	0.80533	0.70257	0.66364	0.75481	0.74396	0.72758
yeast-2_vs_4	0.72808	0.77596	**0.79933**	0.77986	0.78492	0.74677	0.72646	0.75197	0.72426	0.73356
yeast1	0.55574	0.53827	0.49201	0.53827	0.56840	0.56268	**0.57467**	0.55224	0.54105	0.55054
yeast3	0.67502	0.74608	0.74744	0.74608	0.76019	0.68867	0.68393	0.72085	**0.76364**	0.70339
yeast4	0.23677	0.15678	0.31080	0.15678	0.37935	0.33794	0.32651	0.34245	0.32533	**0.35018**
yeast5	0.42645	0.69447	0.65666	0.69447	**0.71218**	0.69627	0.63780	0.61923	0.68351	0.67562
yeast6	0.25223	0.53097	0.52539	0.53097	**0.53524**	0.45071	0.35655	0.43343	0.47251	0.40532
MEAN	0.54324	0.62906	0.57189	0.64582	0.64602	0.64113	0.61715	0.64962	0.66197	0.63700

Table 16: Performance result table for the different undersampling and oversampling methods. Classifier: KNN, Evaluation metric: F1- Score

In this case, it is also clearly observable in *Table 16* that with this performance metric, even though lower performance values are generally obtained (averaging around 0.6), the methods POS, POS 1.0, and POS 2.0 once again exhibit remarkably high performance, yielding superior results in most instances. In this scenario, the leading method is **POS 1.0**, boasting an average of **0.66197**, followed by **POS** with **0.64962**, **NCR** with **0.64602**, and **OSS** with **0.64582**. These results once again underscore that one of our newly introduced methods in this study, specifically POS 1.0, stands out as the top performer, followed closely by the POS method investigated within the scope of this study.

4.4.3. AUC Comparison

Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.90409	0.81500	0.82409	0.88000	0.85045	0.91045	0.91273	**0.93045**	0.89955	0.89500
ecoli-0-3-4-7_vs_5-6	0.88843	0.89140	0.90705	0.91140	0.88922	0.92549	**0.92762**	0.86781	0.90927	0.88071
ecoli-0-6-7_vs_3-5	0.80250	0.79750	0.83500	0.81750	0.82750	0.87500	0.86250	**0.90000**	0.87500	0.84000
glass-0-1-2-3_vs_4-5-6	0.90985	0.87913	0.79600	0.89913	0.90985	0.94591	0.93591	**0.96144**	0.94447	0.95447
glass0	0.81145	0.82069	0.73522	**0.82783**	0.79113	0.81096	0.81108	0.81010	0.80973	0.80628
glass1	0.76583	0.77818	0.67442	0.77461	0.72956	0.78405	0.78138	**0.80597**	0.76294	0.79379
glass6	0.87883	0.83649	0.51712	0.90315	0.88108	0.90901	0.91568	**0.93919**	0.92523	0.91982
new-thyroid1	0.96032	0.91151	0.84286	0.93452	0.96310	0.98889	0.98889	0.91746	**0.99444**	0.98889
page-blocks0	0.90068	0.84656	0.79808	0.85014	0.89338	0.91740	**0.92379**	0.90098	0.91041	0.91813
poker-8-9_vs_6	0.75479	0.76000	0.77555	0.76000	0.76000	0.91760	0.90281	0.94938	**0.99726**	0.91144
winequality-red-4	**0.65080**	0.49968	0.56996	0.51754	0.52816	0.59271	0.60581	0.61219	0.58824	0.58168
winequality-white-3-9_vs_5	0.63890	0.53897	0.53828	0.53897	0.53863	0.58936	0.55917	0.62524	0.60902	**0.64764**
yeast-0-2-5-6_vs_3-7-8-9	0.79881	0.75490	0.80024	0.75490	0.78998	**0.80599**	0.78334	0.78364	0.78559	0.78896
yeast-0-2-5-7-9_vs_3-6-8	**0.90740**	0.89561	0.88843	0.89561	0.89840	0.89406	0.87633	0.89401	0.88235	0.88708
yeast-2_vs_4	0.89592	0.84223	0.87519	0.85115	0.86790	**0.90789**	0.90465	0.89454	0.87376	0.89438
yeast1	0.68331	0.67605	0.64359	0.67605	0.69235	0.69047	**0.70001**	0.67190	0.66881	0.66463
yeast3	**0.90476**	0.84395	0.86298	0.84395	0.88448	0.88999	0.88895	0.87995	0.87048	0.87815
yeast4	**0.82475**	0.54756	0.62890	0.54756	0.69877	0.71537	0.77277	0.77329	0.71942	0.73244
yeast5	0.95368	0.84514	0.91771	0.84514	0.92049	0.95243	0.94861	**0.95486**	0.92153	0.93854
yeast6	0.85765	0.73906	0.80497	0.73906	0.81822	0.83989	**0.86654**	0.82989	0.79945	0.82319
MEAN	0.83464	0.77598	0.76178	0.78841	0.80663	0.84315	0.84343	**0.84511**	0.83735	0.83726

Table 17: Performance result table for the different undersampling and oversampling methods. Classifier: KNN, Evaluation metric: AUC

Once again, in the table (*Table 17*) can be seen that the POS, POS 1.0, and POS 2.0 methods demonstrate a good performance. In this case, the best-performing method was the original version of the **POS** method with **0.84511**, followed by **SMOTE** and **ROS** (both oversampling methods) with **0.84315** and **0.84343**, respectively. Notably, one of the new proposals from this study, **POS 1.0**, also achieved competitive performance with a score of **0.83735**. However, it's worth mentioning that **POS 2.0** isn't far behind, with only a slight difference of 0.00009 in performance, reaching a score of **0.83726**.

4.4.4. AUCPR Comparison

In this last table (*Table 18)*, the trend is once again noticeable: for a significant number of datasets, the methods POS, POS 1.0, and POS 2.0 consistently exhibit the best performance. When considering the average performance values, it becomes evident that the **POS 1.0** method stands out as the strongest performer, boasting a performance score of **0.54115**. Following closely behind, the **OSS** method achieves a competitive performance with a score of **0.53878**, while the **TL** method also demonstrates respectable performance with a score of **0.52175**.

Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.67099	0.66682	0.55043	**0.78449**	0.67466	0.60836	0.60336	0.63539	0.63061	0.61061
ecoli-0-3-4-7_vs_5-6	0.47069	0.71420	0.68411	**0.75028**	0.68087	0.55983	0.57372	0.50422	0.70447	0.60531
ecoli-0-6-7_vs_3-5	0.29426	0.60290	0.64952	0.61086	0.53341	0.50633	0.53338	0.56306	**0.68553**	0.51122
glass-0-1-2-3_vs_4-5-6	0.73927	0.75206	0.62316	0.78275	0.73834	0.81476	0.79964	0.87972	0.86738	**0.88273**
glass0	0.59195	0.61325	0.52250	0.62038	0.54429	0.58951	0.58526	0.61039	**0.62079**	0.60780
glass1	0.60041	0.63354	0.50941	0.62799	0.51129	0.61154	0.64022	**0.64056**	0.59647	0.63525
glass6	0.57140	0.60638	0.17058	0.71611	0.64402	0.65821	0.68876	**0.80046**	0.79956	0.74400
new-thyroid1	0.85335	0.83148	0.73688	0.83034	0.87835	0.90556	0.90556	0.79826	**0.95000**	0.90000
page-blocks0	0.51007	0.63598	0.53972	0.64225	0.59927	0.63110	0.61841	**0.66571**	0.64054	0.64454
poker-8-9_vs_6	0.03492	0.52808	0.33241	0.52808	0.52808	0.65746	0.47281	0.41888	**0.78730**	0.40843
winequality-red-4	0.05500	0.03314	0.08814	0.05917	0.04198	0.05814	0.05620	**0.07077**	0.05657	0.05192
winequality-white-3-9_vs_5	0.05429	0.05552	0.06885	0.05552	0.05552	0.07235	0.03276	0.07433	**0.09345**	0.09150
yeast-0-2-5-6_vs_3-7-8-9	0.30327	**0.46272**	0.43201	**0.46272**	0.42129	0.30742	0.26334	0.34235	0.33394	0.32143
yeast-0-2-5-7-9_vs_3-6-8	0.54348	**0.69123**	0.61161	**0.69123**	0.67427	0.52613	0.47966	0.59088	0.56056	0.56393
yeast-2_vs_4	0.56629	0.67151	**0.68523**	0.67728	0.67381	0.58742	0.55861	0.62552	0.65771	0.61809
yeast1	0.41167	0.43072	0.39332	0.43072	0.40335	0.42272	**0.43099**	0.41642	0.41071	0.40319
yeast3	0.50058	0.59409	0.58966	0.59409	**0.61932**	0.51056	0.50518	0.51718	0.51741	0.53614
yeast4	0.12468	0.08933	0.12786	0.08933	**0.18379**	0.15126	0.16431	0.15279	0.14489	0.13894
yeast5	0.27136	0.50891	0.46972	0.50891	0.52052	0.52026	0.45942	0.39660	**0.52071**	0.45535
yeast6	0.13659	0.31322	0.30688	0.31322	**0.32408**	0.24737	0.18927	0.20837	0.24440	0.22058
MEAN	0.41523	0.52175	0.45460	0.53878	0.51253	0.49731	0.47804	0.49559	**0.54115**	0.49755

Table 18: Performance result table for the different undersampling and oversampling methods. Classifier: KNN, Evaluation metric: AUCPR

Taking a comprehensive view of all the provided tables and considering the performance across various evaluation metrics, a clear pattern emerges. Among all the existing methods, none exhibit the remarkable level of consistency in their results as observed with the POS, POS 1.0, and POS 2.0 methods. Invariably, one among these three methods consistently claims the top spot in terms of performance, while the remaining methods often secure positions among the highest performers. Specifically, the POS 1.0 method consistently yields high performance results, regardless of the evaluation metric employed. In most cases, it ranks as the top-performing method, and in the remaining instances, it is not far from achieving that status.

Furthermore, it's essential to highlight that the efficacy of the original POS method isn't simply maintained; rather, the new approaches proposed in this study, POS 1.0 and POS 2.0, have demonstrated a compelling tendency to surpass in various situation the performance of the original POS method across the some of the evaluation metrics (*Table 15* and *Table 18*). This lends further belief to the notion that these refined variations carry the potential to elevate the performance of the overall approach.

We can draw the conclusion that the initial hypothesis holds true with the KNN classifier, and the methods demonstrate exceptional performance. Nonetheless, it remains uncertain whether they would exhibit similar excellence when employed with different types of classifiers, such as Decision Trees.

4.5. CART Classifier - Phase 1: Study of hyperparameter '*p*'

To verify the stability of the methods across different contexts, such as employing alternative classifiers, a decision has been made to replicate the entire experimental procedure once more. However, this time, the classifier chosen is the CART decision tree algorithm.

By revisiting the experimentation process using the CART decision tree classifier, the aim is to assess whether the exceptional performance observed previously with the KNN classifier remains consistent across different classifier paradigms. This comparison will provide valuable insights into the generalizability and robustness of the methods across various machine learning models.

4.5.1. Fixed number of instances for POS, POS 1.0 and POS 2.0

As it happened in the preliminary experiments utilizing the KNN classifier (*sec. 4.3.1.1, sec 4.3.1.2, sec4.3.2.1, sec 4.3.3.1*), no discernible patterns using the CART classifier could be identified between the alteration of Imbalance Ratio (IR) and the optimal value of parameter '*p*' for any of the three methods. Detailed tables containing this information are provided in the Appendix for reference.

4.5.2. Fixed IR for POS, POS 1.0 and POS 2.0

Similar observations persisted across all methods when the Imbalance Ratio (IR) was held constant at an average value of 9 (*sec 4.3.1.3, sec 4.3.2.2, sec 4.3.3.2*). Likewise, when the datasets were sorted in ascending order of their number of instances, a recurring pattern emerged: the optimal '*p*' values decreased, mirroring the pattern witnessed with KNN classifiers. These tables can be checked in the Appendix section.

However, a slightly distinct set of observations surfaced when the datasets were organized according to the least number of instances within the minority class.

4.5.3. Examining the influence of the number of minority class instances for POS

Upon analyzing the performance using the CART classifier (*Table 19*), notable differences become evident compared to the analogous outcomes with the POS method employing the KNN classifier (*sec 4.3.1.4*). Even though preferred values for 'p' when 'm' (number of minority class instances) is 13 or fewer still tend to be greater than 1 in most instances, there's a shift in the optimal '*p*' value. Specifically, for datasets with a low count of minority class instances, the optimal value for '*p*' now appears closer to 1.5 as opposed to the previous value of 5.

The shift in optimal '*p*' values could be attributed to the inherent characteristics of the classifiers. Specifically, in the case of the CART classifier, it demonstrates better performance when subjected to a slightly higher degree of perturbation in comparison to KNN. When confronted with datasets featuring a limited number of instances in the minority class, the CART classifier adapts by employing a smaller '*p*' value than KNN to induce a slightly higher perturbation since perturbation is given by N(0, 1/m^p). This choice aims to create more refined and precise decision boundaries within the feature space, enabling the classifier to effectively capture complex patterns in the minority class distribution. By intensifying the perturbation, the CART classifier generates complex boundaries that assist in better discerning the nuances of the minority class instances, ultimately enhancing its classification performance for challenging scenarios.

"p" values	glass-0-4_vs_5	glass-0-6_vs_5	shuttle-6_vs_2-3	glass4	glass2	glass-0-1-5_vs_2	poker-8_vs_6	ecoli-0-3-4_vs_5
0.1	0.9815	0.9791	0.9954	0.8834	0.6816	0.6207	0.3747	0.9081
0.2	0.9815	0.9793	0.9954	0.8783	0.6324	0.5527	0.1613	0.9024
0.3	0.9877	0.9743	0.9954	0.8803	0.6504	0.5553	0.2602	0.9423
0.4	0.9877	0.9743	0.9977	0.8425	0.5888	0.5108	0.4181	0.9452
0.5	0.9877	0.9793	0.9977	0.8797	0.6220	0.6127	0.4718	0.8757
0.6	0.9692	0.9793	0.9977	0.8854	0.7163	0.6525	**0.4722**	**0.9477**
0.7	0.9873	0.9793	0.9977	0.9436	0.5876	**0.6588**	0.3733	0.9242
0.8	0.9873	0.9793	**1.0000**	0.9436	0.6068	0.5119	0.3756	0.9035
0.9	0.9873	0.9679	**1.0000**	0.8560	0.6609	0.5147	0.3770	0.8725
1.0	0.9873	0.9793	**1.0000**	0.8560	**0.7472**	0.5659	0.1630	0.8196
1.5	0.9873	**0.9844**	**1.0000**	**0.9461**	0.6858	0.4597	0.2633	0.8484
2.0	0.9287	0.9364	**1.0000**	0.7762	0.4986	0.5756	0.2633	0.8484
3.0	0.9873	0.9312	**1.0000**	0.7721	0.3912	0.4203	0.2633	0.8484
4.0	**0.9936**	0.9364	**1.0000**	0.7721	0.3912	0.4203	0.2633	0.8484
5.0	**0.9936**	0.9364	**1.0000**	0.7721	0.3912	0.4203	0.2633	0.8484
IR	9.22	10.00	22.00	15.47	11.59	9.12	85.88	9.00
m	9	9	10	13	17	17	17	20
dim	92	108	230	214	214	172	1477	200

Table 19: Performance results for the different values of 'p'. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS, arranged in ascending order of 'm'

4.5.4. Optimal '*p*' value for POS

As highlighted in *Table 20*, the trend of preferred '*p*' values remains consistent, once again indicating values lower than 1, specifically less than 0.7. This is demonstrated by the green-shaded cells in the last column labeled "MEAN." This pattern echoes the observations seen earlier with the KNN classifier (*sec. 4.3.1.5*). While a few exceptions exist, such as the *ecoli-0-1_vs_2-3-5* dataset, these have an insignificant impact on the overall average results. In this specific case, the optimal '*p*' value would be 0.4 instead of 0.5. However, this marginal alteration is not substantial enough to impact the results significantly. As a result, the optimal '*p*' values identified for the KNN classifier continue to hold true when applied to the CART classifier.

"p" values	ecoli-0-6-7_vs_3-5	ecoli-0-1_vs_2-3-5	ecoli-0-3-4-7_vs_5-6	poker-8-9_vs_6	winequality-white-3-9_vs_5	glass6	new-thyroid1	yeast6	yeast5	yeast-2_vs_4	yeast4	winequality-red-4	glass-0-1-2-3_vs_4-5-6	glass0	glass1	yeast-0-2-5-7-9_vs_3-6-8	yeast-0-2-5-6_vs_3-7-8-9	yeast3	yeast1	page-blocks0	MEAN
0.1	0.8365	0.8613	0.8772	0.4903	0.4878	0.9120	0.9478	0.6944	0.8915	0.8235	0.6543	**0.4888**	0.8890	0.7703	0.7834	0.8422	0.6860	0.8257	0.6405	0.9277	0.7665
0.2	0.8053	0.8653	0.8957	**0.6026**	0.4499	0.9123	0.9115	0.6692	0.9143	0.8244	0.6490	0.5033	0.8890	0.7635	0.7791	0.8555	0.7283	0.8472	0.6524	0.9227	0.7720
0.3	**0.8460**	0.8563	0.9078	0.4182	0.4219	0.9120	0.9109	0.7413	0.9023	0.8240	0.7264	0.4346	**0.9174**	**0.8092**	0.7877	0.8424	0.7182	0.8573	0.6675	**0.9374**	0.7719
0.4	0.7913	0.8327	0.9004	0.5732	0.3864	0.9145	**0.9684**	0.7494	0.9149	0.8563	0.6579	0.4989	0.8888	0.7823	0.7650	0.8582	0.7130	**0.8589**	0.6446	0.9245	**0.7740**
0.5	0.8141	0.8109	0.9050	0.4246	0.4243	0.9094	0.9335	0.7543	0.9135	0.8274	0.7333	0.4386	0.9124	0.7374	**0.7993**	0.8630	**0.7440**	0.8333	0.6439	0.9201	0.7671
0.6	0.7648	0.8172	0.8893	0.4178	0.3991	0.9100	0.9201	0.7774	0.9008	**0.8746**	**0.7648**	0.3798	0.9064	0.7166	0.7834	**0.8640**	0.6736	0.8531	**0.6684**	0.9156	0.7598
0.7	0.7974	0.8426	0.8650	0.4264	**0.5101**	0.9005	0.9371	**0.7924**	**0.9294**	0.7965	0.6664	0.4182	0.9115	0.7395	0.7498	0.8549	0.6895	0.8062	0.6555	0.9254	0.7607
0.8	0.7729	0.8189	0.8704	0.4182	0.3988	0.9220	0.9480	0.7119	0.9142	0.8659	0.6686	0.3843	0.9014	0.7618	0.7377	0.8601	0.6741	0.8017	0.6628	0.9167	0.7505
0.9	0.7793	0.7937	0.8884	0.2137	0.3989	0.9027	0.9416	0.7818	0.8729	0.8572	0.6949	0.3715	0.9014	0.7685	0.7453	0.8243	0.6837	0.8035	0.6582	0.9238	0.7403
1.0	0.8065	0.8539	0.8706	0.0000	0.4270	**0.9394**	0.9416	0.7499	0.8538	0.8459	0.6540	0.4526	0.8985	0.7670	0.7362	0.8630	0.6858	0.8064	0.6505	0.9240	0.7363
1.5	0.7557	0.8452	**0.8933**	0.0891	0.2499	0.8835	0.9564	0.6140	0.8207	0.8483	0.5957	0.2678	0.8773	0.8005	0.7918	0.8568	0.6623	0.7967	0.6505	0.9139	0.7085
2.0	0.7506	**0.8668**	0.8189	0.0891	0.2797	0.8835	0.9618	0.5973	0.8336	0.8483	0.5950	0.2083	0.8897	0.8005	0.7918	0.8568	0.6623	0.7967	0.6494	0.9123	0.7046
3.0	0.7506	**0.8668**	0.8189	0.0891	0.2517	0.8835	0.9647	0.5973	0.8336	0.8483	0.5950	0.2083	0.8897	0.8005	0.7918	0.8563	0.6872	0.7929	0.6276	0.9115	0.7033
4.0	0.7506	**0.8668**	0.8189	0.0891	0.2517	0.8835	0.9647	0.5973	0.7861	0.8483	0.5466	0.2083	0.8897	0.7899	0.7876	0.8511	0.6674	0.7846	0.6269	0.9115	0.6960
5.0	0.8033	**0.8668**	0.7967	0.0891	0.2517	0.8835	0.9647	0.5973	0.7893	0.8612	0.5326	0.2086	0.9029	0.7899	0.7648	0.8508	0.6424	0.7846	0.6269	0.9115	0.6959
IR	9.09	9.17	9.28	58.4	58.28	6.38	5.14	41.4	32.73	9.08	28.1	29.17	3.2	2	1.82	9.14	9.14	8.1	2.46	8.79	
m	22	24	25	25	25	29	35	35	44	51	51	53	55	70	76	99	99	163	429	559	

Table 20:Performance results for the different values of 'p'. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS, arranged in ascending order of 'm' for datasets with more than m = 20

In a broader overview, it can be summarized that the optimal '*p*' values exhibit consistency (for both KNN and CART classifiers) across datasets with a substantial number of instances in the minority class. However, a slight level of fluctuation becomes apparent when examining datasets characterized by a lower number of minority instances. Despite this variation, a prevailing trend remains intact: a preference for larger '*p*' values in scenarios involving lower number of minority instances, consistently surpassing the threshold of 1.

4.5.5. Examining the influence of the number of minority class instances for POS 1.0

As stated before (*sec. 4.5.2.)* A slightly different pattern of observations emerged compared to what was observed for POS 1.0 (*sec. 4.3.2.3*) when the datasets were arranged based on the minimum number of instances present within the minority class.

This table (*Table 21*) exhibits several discrepancies when compared to the table in section 4.3.2.3, which analyzed the same case for POS 1.0 but with the KNN classifier. In the previous scenario, for datasets with number of instances in the minority class less than 13 (even some datasets with 17 instances), the best performance was consistently achieved with all '*p*' values equal to or greater than 1.5. This led to the selection of an optimal '*p*' value of 5. In the present case, we observe a different trend: only a single dataset adheres to this behavior. For the majority of the other datasets, the optimal performance is notably achieved with '*p*' set to 2. While it is true that performance with values greater than 2 doesn't significantly deteriorate, these results prompt the recommendation to fix '*p*' at 2 when employing the POS method with the CART classifier.

It's worth noting that the previously chosen value (5) with the KNN classifier wouldn't yield unfavorable results either, given these outcomes.

"p" values	glass-0-4_vs_5.dat	glass-0-6_vs_5.dat	shuttle-6_vs_2-3.dat	glass4.dat	glass2.dat	glass-0-1-5_vs_2.dat	poker-8_vs_6.dat	ecoli-0-3-4_vs_5.dat
0.1	0.9813	0.9276	0.9954	0.6723	0.5453	0.4987	0.1732	0.8465
0.2	0.9813	0.7077	0.9954	0.6113	0.5831	0.3811	0.3365	0.8485
0.3	0.9748	0.7114	0.9954	0.8702	0.5049	**0.6744**	0.7732	0.8717
0.4	0.9813	0.5633	0.9954	0.8597	0.5116	0.6036	0.8991	0.8485
0.5	0.9813	0.5688	0.9954	0.6616	0.6097	0.6009	0.8998	0.8746
0.6	0.9813	0.5077	0.9977	0.8465	0.6907	0.5954	0.8986	0.8432
0.7	0.9877	0.7847	0.9977	0.8465	0.6969	0.6514	**0.9729**	**0.8774**
0.8	0.9940	0.7051	0.9977	0.8296	0.6969	0.5457	0.9458	0.8428
0.9	0.9940	0.9207	0.9954	0.8648	0.6969	0.5457	0.8885	0.8147
1.0	0.9940	0.9795	0.9977	**0.9482**	**0.6991**	0.6518	0.7732	0.8134
1.5	0.9811	0.9738	**1.0000**	0.8713	**0.6991**	0.5421	0.7359	0.8195
2.0	**1.0000**	**0.9844**	**1.0000**	0.8088	0.6602	0.5485	0.7359	0.8484
3.0	0.9873	0.9312	**1.0000**	0.7721	0.5668	0.5485	0.7359	0.8484
4.0	0.9936	0.9364	**1.0000**	0.7721	0.5668	0.5485	0.7359	0.8484
5.0	0.9936	0.9364	**1.0000**	0.7721	0.2988	0.3806	0.3357	0.8484
IR	9.22	10	22	15.47	11.59	9.12	85.88	9.00
m	9	9	10	13	17	17	17	20
dim	92	108	230	214	214	172	1477	200

Table 21: Performance results for the different values of 'p'. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of 'm'

4.5.6. Optimal '*p*' value for POS 1.0

When considering the optimal '*p*' values for POS 1.0 with a substantial number of instances (greater than 13) and utilizing the CART decision tree classifier, it becomes evident in the table (*Table 22)* that the optimal '*p*' value is 0.6. This is highlighted in the green-shaded cells in the last column. Moreover, mirroring the observations from the previous instance of this experiment in POS 1.0 but employing the KNN classifier (sec. 4.3.2.4), it is evident that the best-performing '*p'* values are those below 1. In the earlier case, the selected optimal '*p*' value was 0.7, while in this instance, it is 0.6. However, this minute difference is of minor consequence and wouldn't result in significant alterations to the overall performance.

"p" values	ecoli-0-6-7_vs_3-5	ecoli-0-1_vs_2-3-5	ecoli-0-3-4-7_vs_5-6	poker-8-9_vs_6	winequality-white-3-9_vs_5	glass6	new-thyroid1	yeast6	yeast5	yeast-2_vs_4	yeast4	winequality-red-4	glass-0-1-2-3_vs_4-5-6	glass0	glass1	yeast-0-2-5-7-9_vs_3-6-8	yeast-0-2-5-6_vs_3-7-8-9	yeast3	yeast1	page-blocks0	MEAN
0.1	0.8221	0.7470	0.8474	0.9539	0.3391	0.8902	0.9440	**0.6753**	0.8563	**0.8935**	0.5695	0.2624	0.9256	0.8434	0.6910	0.8486	**0.6797**	0.8071	**0.6496**	0.9049	0.7575
0.2	0.8124	0.7742	0.8679	**1.0000**	0.3893	0.9042	0.9157	**0.6753**	0.8563	0.8605	0.5695	0.1848	0.9038	**0.8618**	0.7197	0.8484	**0.6797**	0.8071	**0.6496**	0.9017	0.7591
0.3	**0.8774**	0.7988	0.8509	**1.0000**	0.3008	0.8927	0.9510	**0.6753**	0.8161	0.8597	0.5695	0.3266	0.9191	0.8089	0.7490	0.8544	**0.6797**	0.8057	**0.6496**	0.9002	0.7643
0.4	0.8378	**0.9024**	0.8694	**1.0000**	0.3874	0.8927	0.9157	**0.6753**	0.7598	0.8484	0.5695	0.3828	0.9024	0.7994	0.7191	**0.8657**	0.6727	0.8028	**0.6496**	0.9120	0.7682
0.5	0.8563	0.8211	**0.8982**	**1.0000**	0.4758	0.9212	0.9419	0.5783	0.7437	0.8247	0.5437	0.3276	0.9161	0.7539	0.7153	0.8479	0.6727	0.8038	**0.6496**	0.9117	0.7602
0.6	0.8264	0.8847	0.8922	0.9547	**0.5490**	0.9094	0.9452	0.6333	0.7866	0.8151	0.5864	**0.4924**	0.9356	0.7914	**0.7820**	0.8591	0.6727	0.7786	**0.6496**	0.9103	**0.7827**
0.7	0.8284	0.8207	0.8940	0.9549	0.3007	0.9145	0.9421	0.5957	0.7920	0.8339	0.5442	0.2856	**0.9385**	0.8109	0.7236	0.8420	0.6727	**0.8075**	**0.6496**	0.9164	0.7534
0.8	0.8284	0.7819	0.8848	0.9549	0.3354	0.9120	0.9208	0.5961	0.8019	0.8572	0.5867	0.4038	0.9255	0.8273	0.7311	0.8476	0.6727	0.7713	**0.6496**	0.9159	0.7603
0.9	0.8495	0.7797	0.8868	0.9777	0.3004	0.8930	0.9231	0.5961	0.8181	0.8521	0.5745	0.4797	0.9255	0.7771	0.7536	0.8346	0.6727	0.7751	**0.6496**	**0.9233**	0.7621
1.0	0.8513	0.8201	0.8534	0.8000	0.3644	**0.9265**	**0.9647**	0.5528	**0.8687**	0.8397	**0.6407**	0.3941	0.9236	0.8049	0.7494	0.8486	0.6727	0.7941	0.6327	0.9171	0.7610
1.5	0.8244	0.8343	0.8743	0.7549	0.4250	0.9262	0.9618	0.6015	0.8527	0.8371	0.6261	0.2421	0.9013	0.7802	0.7066	0.8481	0.6727	0.7867	0.6327	0.9158	0.7502
2.0	0.7506	0.8624	0.8306	0.7549	0.1783	0.9122	0.9618	0.6562	0.8527	0.8361	0.6261	0.2414	0.8905	0.7915	0.6980	0.8481	0.6727	0.7858	0.6327	0.9132	0.7348
3.0	0.7506	0.8624	0.8306	0.7549	0.3043	0.9122	0.9618	0.6750	0.8656	0.8673	0.6261	0.2659	0.8905	0.7915	0.6990	0.8611	0.6450	0.7780	0.6397	0.9143	0.7448
4.0	0.7506	0.8624	0.8498	0.6441	0.3043	0.9122	0.9618	0.5973	0.8141	0.8152	0.5162	0.2084	0.8905	0.7969	0.7880	0.8454	0.6496	0.7843	0.6269	0.9143	0.7266
5.0	0.7506	0.8645	0.8172	0.0891	0.2517	0.8810	0.9618	0.5973	0.8023	0.8603	0.5326	0.2086	0.9029	0.7899	0.7648	0.8508	0.6424	0.7846	0.6269	0.9115	0.6945
IR	9.09	9.17	9.28	58.4	58.28	6.38	5.14	41.4	32.73	9.08	28.1	29.17	3.2	2	1.82	9.14	9.14	8.1	2.46	8.79	
m	22	24	25	25	25	29	35	35	44	51	51	53	55	70	76	99	99	163	429	559	

Table 22: Performance results for the different values of 'p'. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of 'm' for datasets with more than m = 20

4.5.7. Examining the influence of the number of minority class instances for POS 2.0

"p" values	glass-0-4_vs_5	glass-0-6_vs_5	shuttle-6_vs_2-3	glass4	glass2	glass-0-1-5_vs_2	poker-8_vs_6	ecoli-0-3-4_vs_5.
0.1	0.9748	0.9475	0.9977	0.7128	**0.7262**	0.4570	0.1732	0.8385
0.2	0.9232	0.9476	0.9977	0.7107	0.6593	0.4456	0.3365	0.8101
0.3	0.9499	0.9744	0.9977	0.6604	0.5316	0.5024	0.7732	0.8411
0.4	0.9312	0.9149	0.9954	0.7419	0.6850	**0.6589**	0.8991	0.8673
0.5	0.9692	0.8539	0.9931	0.4534	0.6873	0.5087	0.8998	0.8669
0.6	0.9680	0.8297	0.9954	0.8415	0.6870	0.6145	0.8986	0.8586
0.7	0.9626	0.8211	0.9931	0.8529	0.6969	0.6426	**0.9729**	**0.8985**
0.8	0.9688	0.8963	0.9977	0.8891	0.6956	0.6458	0.9458	0.8726
0.9	0.9877	0.7579	0.9977	0.9288	0.6961	0.6445	0.8885	0.8701
1.0	**0.9940**	0.8985	**1.0000**	**0.9415**	0.6985	0.6089	0.7732	0.8534
1.5	0.9308	0.9793	**1.0000**	0.7968	0.6103	0.5412	0.7359	0.8534
2.0	0.9748	**0.9897**	**1.0000**	0.7707	0.6156	0.5485	0.7359	0.8509
3.0	0.9351	0.9312	**1.0000**	0.7721	0.6168	0.5485	0.7359	0.8509
4.0	0.9936	0.9364	**1.0000**	0.7721	0.6168	0.5681	0.7359	0.8509
5.0	0.9936	0.9364	**1.0000**	0.7721	0.4607	0.5320	0.3357	0.8509
IR	9.22	10.00	22	15.47	11.59	9.12	85.88	9.00
m	9	9	10	13	17	17	17	20
dim	92	108	230	214	214	172	1477	200

Table 23: Performance results for the different values of 'p'. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of 'm'

Through this study in the above table (*Table 23*), we gain valuable insights into the optimal parameter values that enhance performance within the framework of POS 2.0, coupled with the CART classifier and datasets comprising fewer than 13 instances. Strikingly, the findings reiterate the significance of larger parameter values, consistently greater than 1, to achieve optimal results. In contrast to the findings from the same study but employing the KNN classifier (sec. 4.3.3.3), where the optimal 'p' value was 5, the value that would yield superior results in this CART scenario would be 2.

4.5.8. Optimal '*p*' value for POS 2.0

In the presented table (*Table 24*), akin to the previous scenario where the optimal '*p*' values were analyzed for datasets with a considerable number of instances in the minority class using POS 2.0 and the KNN classifier (sec. 4.3.3.4), a distinct pattern emerges. The optimal '*p*' values associated with superior performance are central values, specifically those ranging between 1 and 0.7. In this instance, the most favorable '*p*' value would be 0.8. Notably, this value diverges from the equivalent case with the KNN classifier, where the optimal '*p*' value was 0.9.

"p" values	ecoli-0-6-7_vs_3-5	ecoli-0-1_vs_2-3-5	ecoli-0-3-4-7_vs_5-6	poker-8-9_vs_6	winequality-white-3-9_vs_5	glass6	new-thyroid1	yeast6	yeast5	yeast4	yeast-2_vs_4	winequality-red-4	glass-0-1-2-3_vs_4-5-6	glass0	glass1	yeast-0-2-5-7-9_vs_3-6-8	yeast-0-2-5-6_vs_3-7-8-9	yeast3	yeast1	page-blocks0	MEAN
0.1	0.8244	0.7248	0.8280	0.2145	0.2660	0.9080	0.9508	0.6692	0.8401	0.5889	0.8468	0.4446	0.9191	0.7512	0.6931	0.8424	0.6521	0.8040	**0.6638**	0.9065	0.7169
0.2	0.8513	0.7419	0.8280	0.1780	0.4426	0.9236	0.9312	0.6685	0.8401	0.5889	0.8511	0.3187	0.9132	0.7749	0.6969	0.8595	0.6521	0.8028	**0.6638**	0.8995	0.7213
0.3	0.8226	0.7237	0.8260	0.5975	0.5870	0.9212	0.9451	0.6692	0.8532	0.5695	0.8705	0.4995	0.9132	0.7339	0.7450	0.8694	0.6581	0.8023	**0.6638**	0.9185	0.7595
0.4	0.8796	0.7237	0.8283	0.9542	0.5866	0.9070	0.9349	0.6692	0.7571	0.5695	0.8705	0.4254	0.9132	0.7904	0.7004	0.8543	0.6651	0.8105	**0.6638**	0.9129	0.7708
0.5	0.8820	0.7443	0.8699	0.9092	0.6188	0.9045	**0.9916**	0.6680	0.7560	0.5437	**0.8739**	0.4817	**0.9257**	0.7462	0.6762	0.8495	0.6651	0.8089	**0.6638**	0.9133	0.7746
0.6	**0.8950**	**0.7990**	0.8698	**0.9549**	0.5976	0.9120	0.9539	0.5785	0.7580	0.5604	0.8693	0.4793	0.9163	0.7647	0.7350	0.8692	0.6570	0.7987	**0.6638**	0.9130	0.7773
0.7	0.8805	0.8263	0.8738	**0.9549**	0.5643	0.9198	0.9458	0.5352	0.7885	0.5441	0.8708	0.5364	0.9032	**0.8305**	0.7245	0.8692	0.6570	0.7979	**0.6638**	0.9164	0.7801
0.8	0.8730	0.8231	0.8758	**0.9549**	**0.7239**	0.9074	0.9537	0.5787	0.8358	0.5862	0.8531	0.5146	0.9024	0.8228	0.7556	0.8634	0.6570	0.8146	**0.6638**	0.9243	**0.7942**
0.9	0.8262	0.8585	**0.8920**	**0.9549**	0.6149	0.9170	0.9482	0.5520	0.7877	0.5708	0.8701	**0.5390**	0.9021	0.8022	0.7471	0.8529	0.6570	0.8173	**0.6638**	**0.9331**	0.7853
1.0	0.8262	0.8706	0.8696	0.9533	0.4830	0.9239	0.9536	0.6516	0.8109	0.5715	0.8687	0.5332	0.8761	0.8006	0.7549	**0.8730**	**0.6869**	**0.8176**	**0.6638**	0.9195	0.7854
1.5	0.8344	0.7317	0.8554	0.9536	0.3652	**0.9367**	0.9564	**0.6740**	**0.8958**	**0.6418**	0.8274	0.3320	0.9150	0.8164	**0.7579**	0.8349	0.6570	0.8142	**0.6638**	0.9183	0.7691
2.0	0.8344	0.7749	0.8341	0.9536	0.3035	0.8788	0.9430	0.6555	0.8952	0.5237	0.8274	0.2682	0.9088	0.8140	0.7504	0.8349	0.6570	0.7901	0.6415	0.9074	0.7498
3.0	0.8344	0.7749	0.8341	0.9536	0.1773	0.8788	0.9429	0.6314	0.8526	0.5237	0.8274	0.2683	0.9088	0.8065	0.7564	0.8579	0.6762	0.8088	0.6415	0.9073	0.7431
4.0	0.8344	0.7749	0.8341	0.8153	0.1773	0.8788	0.9429	0.6302	0.7897	0.4246	0.8382	0.1468	0.9117	0.8028	0.7564	0.8470	0.6762	0.8031	0.6332	0.9077	0.7213
5.0	0.7532	0.7751	0.7903	0.0000	0.1773	0.8788	0.9458	0.6441	0.8025	0.4254	0.8594	0.1657	0.8995	0.7987	0.7456	0.8465	0.6714	0.8031	0.6332	0.9077	0.6762
IR	9.09	9.17	9.28	58.4	58.28	6.38	5.14	41.4	32.73	28.1	9.08	29.17	3.2	2	1.82	9.14	9.14	8.1	2.46	8.79	
m	22	24	25	25	25	29	35	35	44	51	51	53	55	70	76	99	99	163	429	559	

Table 24: Performance results for the different values of 'p'. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of 'm' for datasets with more than m = 20

Consistently observed across all cases using the CART classifier—POS, POS 1.0, or POS 2.0 —is a minimal variation in optimal '*p*' values for datasets characterized by a high number of instances in the minority class, compared to the KNN classifier. The marginal discrepancy is of such insignificant that even if the optimal '*p*' values for KNN are employed in the CART context, exceptional performance can be anticipated without any concern. This reaffirms the initial conclusions regarding optimal '*p*' values.

However, a shift in dynamics becomes apparent when dealing with datasets featuring a notably low number of minority instances. This can be seen in *Table 25*. Here, the '*p*' values demonstrate fluctuation. While previously, the ideal values for POS, POS 1.0, and POS 2.0 were all 5, the current findings suggest slightly smaller values, specifically 1.5 and 2, emerging as optimal. Notably, this trend maintains the essence of the initial concept where values greater than 1 are preferred. Nevertheless, this variation should be considered, possibly attributed to the distinct nature of the CART decision tree classifier as stated in (sec. 4.5.3)

		Low 'm'	***High 'm'***
POS	**KNN**	5	0.3
	CART	1.5	0.4
POS 1.0	**KNN**	5	0.7
	CART	2	0.6
POS 2.0	**KNN**	5	0.9
	CART	2	0.8

Table 25: Optimal 'p' values for each method and classifier

4.6. CART Classifier - Phase 2: Method Comparative Analysis

Once again, a comparison between the methods introduced in this study and existing approaches is conducted as in section 4.4. In this case, with the application of the CART classifier, the optimal '*p*' values established for the KNN classifier are evidently transferable. However, for enhanced performance in contrast to other existing methods, it is recommended to employ the '*p*' values tailored specifically for the CART classifier. Nonetheless, given the consistency of the methods and the optimal 'p' values identified for datasets abundant in instances of the minority class, utilizing the '*p*' values optimized for KNN wouldn't lead to unfavorable outcomes when employed in conjunction with the CART classifier.

4.6.1. Geometric Mean Comparison

The results presented in *Table 26* unmistakably highlight that the method POS 2.0 achieves the highest average performance. While OSS, NCR, and SMOTE showcase better performance in specific cases, the standout performer in this comparison remains the **POS 2.0** method, exhibiting an average performance score of **0.80477**. The subsequent contender is the **RUS** method, closely following with a score of **0.79270**. The next two methods in line are those introduced within this study, namely **POS** and **POS 1.0**, boasting performance values of **0.77789** and **0.76658**, respectively. These methods significantly outshine the remaining techniques included in the comparative analysis.

Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.89126	0.71444	0.75823	0.77970	0.88174	0.80577	0.82155	0.89232	0.88467	**0.90744**
ecoli-0-3-4-7_vs_5-6	0.76805	0.87717	0.83642	0.86785	0.87271	0.79032	0.84060	0.86799	0.85254	**0.88470**
ecoli-0-6-7_vs_3-5	0.78297	0.79843	0.77638	0.79288	0.78814	0.80302	0.82903	0.80085	0.83841	**0.85440**
glass-0-1-2-3_vs_4-5-6	0.90044	0.92868	0.82411	0.92511	0.90702	0.86521	0.90851	**0.93222**	0.92282	0.92850
glass0	0.77920	0.77643	0.71268	**0.83795**	0.78036	0.82223	0.81337	0.83362	0.77005	0.79290
glass1	0.69731	0.75769	0.67181	0.73039	0.69084	0.73015	0.74237	0.72898	0.71109	**0.76874**
glass6	0.91094	0.91453	0.80326	0.81812	0.92116	0.91467	**0.92616**	0.91985	0.91010	0.88489
new-thyroid1	0.95703	0.93372	0.92934	0.96373	0.93527	0.94856	0.94856	0.93195	**0.96387**	0.95776
page-blocks0	0.92278	0.91500	0.85392	0.91738	0.94609	0.90844	0.92856	0.91939	0.91320	**0.92600**
poker-8-9_vs_6	0.51864	0.00000	0.54485	0.08944	0.08852	0.08898	0.53312	0.50949	**0.97889**	0.92584
winequality-red-4	0.56804	0.29269	0.49707	0.22300	0.34241	0.12205	0.37972	0.50582	0.38850	**0.57607**
winequality-white-3-9_vs_5	0.53634	0.34056	0.31262	0.30362	0.40482	0.30349	0.36185	0.39947	0.21248	**0.68255**
yeast-0-2-5-6_vs_3-7-8-9	0.73560	0.67178	0.72546	0.70296	0.77284	0.64080	0.68909	0.71492	0.74191	**0.77506**
yeast-0-2-5-7-9_vs_3-6-8	0.86055	0.83522	0.82048	0.82610	**0.89632**	0.83919	0.85404	0.85808	0.87501	0.87564
yeast-2_vs_4	0.89280	0.82639	0.84983	0.82581	0.85156	0.84984	0.87879	0.86755	0.86805	**0.90827**
yeast1	0.66420	0.68059	0.60486	**0.68461**	0.66924	0.67373	0.66425	0.66039	0.64734	0.65852
yeast3	0.89041	0.83117	0.83137	0.83459	**0.89285**	0.80108	0.81375	0.86030	0.82082	0.83914
yeast4	**0.82929**	0.51042	0.56370	0.56818	0.63593	0.53408	0.64102	0.72601	0.58236	0.48331
yeast5	**0.93507**	0.87263	0.78413	0.78917	0.93096	0.78820	0.78219	0.85131	0.78262	0.84524
yeast6	**0.81300**	0.69540	0.66151	0.66704	0.77205	0.62993	0.73811	0.77738	0.66682	0.62039
MEAN	0.79270	0.70865	0.71810	0.70738	0.74904	0.69299	0.75473	0.77789	0.76658	**0.80477**

Table 26:Performance result table for the different undersampling and oversampling methods. Classifier: CART Decision Tree, Evaluation metric: GM

4.6.2. F1 – Score Comparison

Within the context of *Table 27*, a striking trend emerges that distinctly separates the new methodological approaches from the rest. These new techniques have managed to establish a considerable lead over their traditional counterparts, creating a significant performance disparity. Notably, **POS 1.0** emerges as the undeniable champion in this arena, showcasing an impressive average performance score of **0.65411**. This success is closely followed by **POS 2.0**, further reinforcing the efficacy of these newly introduced methods with a performance score of **0.63993**.

In comparison, the method **SMOTE** follows them achieving a performance score of **0.59265**. However, despite obtaining high performance, SMOTE has not obtained the best performance in any dataset when compared to the rest of the methods under consideration. These findings emphasize the groundbreaking potential and robust performance of the new approaches, setting them apart as frontrunners in the pursuit of effective data augmentation and improved classification outcomes.

Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.53183	0.57416	0.57576	0.59855	0.66926	0.66847	0.66974	0.65641	**0.68873**	0.68687
ecoli-0-3-4-7_vs_5-6	0.41279	0.73212	0.52772	0.65823	0.76109	0.62216	0.68065	0.65498	0.73074	**0.76112**
ecoli-0-6-7_vs_3-5	0.44556	0.63550	0.48078	0.57974	0.51576	0.65974	0.66444	0.63736	**0.76895**	0.71873
glass-0-1-2-3_vs_4-5-6	0.80614	0.86375	0.72157	0.86027	0.80512	0.79234	0.81453	0.89005	**0.90911**	0.90743
glass0	0.70310	0.72198	0.67578	0.73375	0.70923	0.73746	0.72912	**0.75937**	0.70358	0.73209
glass1	0.63698	0.68154	0.55988	0.63785	0.64741	0.67358	0.70322	**0.71222**	0.65364	0.65984
glass6	0.75033	0.83800	0.54646	0.67786	0.78284	0.79983	0.84043	**0.87879**	0.83333	0.78392
new-thyroid1	0.83444	0.91090	0.78013	**0.94628**	0.91194	0.92628	0.94167	0.92833	0.91179	0.87684
page-blocks0	0.73843	0.83473	0.72928	0.83457	0.82391	0.83611	0.82885	0.83169	0.82750	**0.83770**
poker-8-9_vs_6	0.04161	0.00000	0.30356	0.05714	0.00000	0.05000	0.29610	0.17208	**1.00000**	0.88718
winequality-red-4	0.09162	0.09463	0.12466	0.06355	0.08773	0.04485	0.13041	0.16089	0.12731	**0.13730**
winequality-white-3-9_vs_5	0.03567	0.23873	0.09467	0.17582	**0.25897**	0.25556	0.11207	0.18754	0.21868	0.08641
yeast-0-2-5-6_vs_3-7-8-9	0.34608	0.47891	0.40598	0.49745	0.54650	0.48133	0.45006	0.41348	0.57761	**0.57983**
yeast-0-2-5-7-9_vs_3-6-8	0.55739	0.69674	0.52658	0.66698	0.74547	0.73002	0.70286	0.67650	0.71817	**0.75740**
yeast-2_vs_4	0.65989	0.73497	0.69493	0.69108	0.68818	0.72150	0.74212	0.70290	0.76861	**0.77203**
yeast1	0.52837	0.54914	0.48219	**0.56048**	0.55753	0.54974	0.52079	0.52330	0.51401	0.52250
yeast3	0.61918	0.70266	0.66419	**0.71092**	0.71066	0.70053	0.67485	0.66090	0.69302	0.69850
yeast4	0.23020	0.27970	0.19901	0.31466	0.28254	0.31507	0.30479	0.29235	**0.39919**	0.35161
yeast5	0.47271	0.69023	0.56841	0.60675	**0.75346**	0.61063	0.61230	0.52158	0.65828	0.62034
yeast6	0.16427	**0.48968**	0.20977	0.40507	0.44373	0.46349	0.43405	0.35641	0.37993	0.42090
MEAN	0.48033	0.58740	0.49357	0.56385	0.58507	0.58194	0.59265	0.58086	**0.65411**	0.63993

Table 27: Performance result table for the different undersampling and oversampling methods. Classifier: CART Decision Tree, Evaluation metric: F1-Score

4.6.3. AUC Comparison

Upon reviewing *Table 28*, which presents the performance assessed through the AUC metric, it becomes evident once again that the POS, POS 1.0, and POS 2.0 methods consistently deliver strong performance outcomes. What's particularly striking is that upon examining the average performance scores, none of the other existing methods manage to surpass a score of 0.8. This stands in sharp contrast to the methods introduced in this study, which have achieved values exceeding 0.8. Remarkably, all three of these newly implemented methods emerge as the highest-performing techniques. Leading the pack is **POS 2.0**, attaining an impressive score of **0.8268**, closely followed by **POS 1.0** with a score of **0.80924**, and the original **POS** method at **0.80730**.

Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.85818	0.76682	0.78545	0.82864	0.87864	0.83727	0.83727	0.89455	0.88773	**0.91045**
ecoli-0-3-4-7_vs_5-6	0.78377	0.86284	0.81538	0.84989	0.89284	0.79849	0.86984	0.88115	0.88067	**0.89419**
ecoli-0-6-7_vs_3-5	0.80000	0.80000	0.78000	0.80250	0.77500	0.80500	0.84500	0.80750	0.85500	**0.86500**
glass-0-1-2-3_vs_4-5-6	0.88826	0.90922	0.82521	0.91307	0.90191	0.86176	0.88985	0.92134	0.92316	**0.93913**
glass0	0.77783	0.79458	0.75948	0.81305	0.79372	0.80505	0.80209	**0.82796**	0.77722	0.80259
glass1	0.71523	0.75068	0.67850	0.72067	0.70519	0.75513	0.77200	0.75790	0.72076	**0.78185**
glass6	0.89865	0.91712	0.81063	0.84910	0.90631	0.89505	**0.93108**	0.92523	0.90045	0.88964
new-thyroid1	0.95833	0.93730	0.90992	0.96238	0.93730	0.95159	0.96587	0.91786	0.95159	**0.96349**
page-blocks0	0.93156	0.91281	0.86315	0.91350	**0.94277**	0.90997	0.92855	0.92135	0.91339	0.92745
poker-8-9_vs_6	0.54932	0.49349	0.67308	0.51521	0.49281	0.51623	0.63555	0.61842	**1.00000**	0.91897
winequality-red-4	0.59651	0.53046	0.59563	0.51793	0.53017	0.50907	0.57618	0.60972	0.57757	**0.62169**
winequality-white-3-9_vs_5	0.52751	0.59691	0.61540	0.57176	0.63176	0.61485	0.59701	0.63358	0.58661	**0.70088**
yeast-0-2-5-6_vs_3-7-8-9	0.73228	0.71864	0.73115	0.73672	**0.79649**	0.71059	0.72206	0.71360	0.75864	0.77364
yeast-0-2-5-7-9_vs_3-6-8	0.85510	0.84487	0.82346	0.82714	0.87682	0.84985	0.86493	**0.88493**	0.85848	0.87485
yeast-2_vs_4	0.89650	0.85943	0.86429	0.82745	0.85884	0.83963	0.87621	0.87606	0.86731	**0.91330**
yeast1	0.65920	0.68168	0.62969	**0.69010**	0.68136	0.68277	0.66138	0.66187	0.65788	0.67222
yeast3	**0.89030**	0.84793	0.84700	0.84699	0.88734	0.82264	0.82435	0.85571	0.83069	0.84187
yeast4	**0.82183**	0.63941	0.63329	0.65255	0.67859	0.64464	0.69621	0.74947	0.68395	0.64884
yeast5	0.91111	0.85660	0.83819	0.78889	**0.93438**	0.76840	0.80903	0.87535	0.82153	0.82014
yeast6	0.81220	0.76350	0.69063	0.73216	0.79911	0.72305	0.78586	**0.81249**	0.73216	0.76350
MEAN	0.79318	0.77421	0.75848	0.76798	0.79507	0.76505	0.79452	0.80730	0.80924	**0.82618**

Table 28: Performance result table for the different undersampling and oversampling methods. Classifier: CART Decision Tree, Evaluation metric: AUC

The success of the new approaches becomes increasingly evident, surpassing not only the established methods but also outshining the original POS method itself. This further underscores the substantial improvement and innovation these newly introduced techniques bring to the field of data augmentation and classification performance.

4.6.4. AUCPR Comparison

Observing *Table 29*, it becomes apparent that when evaluating performance using the AUCPR metric, the achieved scores are generally quite low compared to the previous metrics. None of the traditional methods manage to surpass the threshold of 0.5 in performance. Despite this trend, the new approaches introduced in this study, specifically POS 1.0 and POS 2.0, exhibit performance scores exceeding 0.5. More precisely, **POS 1.0** obtains a score of **0.51443**, while **POS 2.0** achieves **0.50813**. Consequently, these two methods stand out as the only ones with noteworthy performance, setting them apart from the rest of the methods evaluated in this study.

Dataset	RUS	TL	CNN	OSS	NCR	ROS	SMOTE	POS	POS_1	POS_2
ecoli-0-1_vs_2-3-5	0.35383	0.43074	0.41425	0.43175	0.48673	0.51200	0.51390	0.48456	0.57205	**0.57491**
ecoli-0-3-4-7_vs_5-6	0.26879	0.58347	0.35605	0.48483	**0.64394**	0.47183	0.51612	0.47644	**0.60820**	0.57704
ecoli-0-6-7_vs_3-5	0.28183	0.46691	0.33293	0.40090	0.33814	0.49691	0.49887	0.33340	**0.63697**	0.56419
glass-0-1-2-3_vs_4-5-6	0.69549	0.78341	0.58986	0.77558	0.68892	0.69311	0.70361	**0.81527**	0.78177	0.79730
glass0	0.58553	0.61408	0.56722	0.62347	0.55975	0.63617	0.61969	**0.63652**	0.58951	0.60523
glass1	0.52846	0.58653	0.50143	0.54530	0.49493	0.59390	0.60957	0.59362	0.54625	**0.62117**
glass6	0.63202	0.73487	0.37824	0.55341	0.65154	0.69363	0.73141	**0.80035**	0.70730	0.62484
new-thyroid1	0.72889	0.85987	0.65999	0.90664	0.85483	0.88379	**0.90772**	0.71697	0.87943	0.80505
page-blocks0	0.57722	0.71348	0.55803	0.71282	0.69497	0.71604	0.70209	0.71513	0.71142	**0.71611**
poker-8-9_vs_6	0.02274	0.01684	0.24953	0.03616	0.01684	0.02949	0.16545	0.04700	**1.00000**	0.81269
winequality-red-4	0.04598	0.04249	0.04790	0.04307	0.04313	0.03755	0.05266	0.06035	0.06036	**0.06225**
winequality-white-3-9_vs_5	0.01940	0.13550	0.03840	0.10417	0.13333	0.12901	0.04557	**0.06826**	0.04768	0.06044
yeast-0-2-5-6_vs_3-7-8-9	0.19624	0.28813	0.22932	0.29702	0.34466	0.28885	0.26003	0.21542	0.32517	**0.34575**
yeast-0-2-5-7-9_vs_3-6-8	0.36974	0.51761	0.33681	0.47842	**0.58171**	0.56794	0.52189	0.49420	0.53011	0.57347
yeast-2_vs_4	0.48550	0.57333	0.54141	0.51482	0.51004	0.56020	0.58758	0.53571	0.58275	**0.64087**
yeast1	0.38949	0.42590	0.37197	**0.43627**	0.39683	0.43119	0.40550	0.40025	0.40326	0.42043
yeast3	0.43850	0.52563	0.48074	**0.53984**	0.53892	0.53374	0.49234	0.47459	0.50945	0.52884
yeast4	0.11821	0.11048	0.07793	0.13233	0.11038	0.13885	0.13088	0.13162	**0.18123**	0.11961
yeast5	0.30009	0.49522	0.35839	0.40157	**0.59445**	0.40631	0.40654	0.32544	0.41721	0.41340
yeast6	0.08092	0.27901	0.08841	0.21686	0.23851	0.28577	0.22159	0.17676	0.19856	**0.29904**
MEAN	0.35594	0.45917	0.35894	0.43176	0.44613	0.45531	0.45465	0.42509	**0.51443**	0.50813

Table 29: Performance result table for the different undersampling and oversampling methods. Classifier: CART Decision Tree, Evaluation metric: AUCPR

Upon comprehensive examination of the comparative analyses, a striking pattern emerges, showing the consistency of POS, POS 1.0, and POS 2.0 in terms of their performance. Notably, this consistency holds true regardless of whether the classifiers employed are KNN or CART. This remarkable stability can be attributed to the underlying robustness and effectiveness of these methods.

These new approaches have demonstrated an unparalleled level of performance, consistently outperforming the traditional methods across a diverse array of evaluation metrics. The results obtained not only affirm the superiority of these methods but also emphasize their resilience in varying scenarios. Both the KNN and CART classifiers have been subjected to these novel methods, revealing that the methods' effectiveness in improving data augmentation and classification performance is not dependent on the specific classifier utilized. In essence, the trio of POS techniques, POS, POS 1.0, and POS 2.0, offer a reliable solution for enhancing classification outcomes and overcoming the challenges posed by class imbalance.

5. Conclusion

In this study, the POS method has been explored. The Perturbation Based Sampling method involves introducing controlled perturbations through a hyperparameter '*p*' to the minority class in a dataset for imbalanced classification problems. This is done with the goal of enhancing classification algorithm performance.

Following an in-depth analysis of the original POS method, two refined versions were introduced in this study—namely POS 1.0 and POS 2.0. These versions aimed to clarify previously unclear aspects of the original method while also offering improved alternatives to it. By addressing ambiguities and incorporating enhancements, the objective was to elevate the methodology's effectiveness and utility in mitigating the challenges associated with class imbalance.

These methods demonstrate exceptional performance. However, the challenge lies in determining an optimal value for the hyperparameter 'p' that would yield even higher performance.

An extensive array of experiments has been carried out to ascertain the ideal value for the hyperparameter '*p*' across a diverse range of dataset scenarios. Initially, a prevailing hypothesis suggested that the Imbalance Ratio (IR) of the datasets would directly correlate with the optimal 'p' value, implying that the degree of class imbalance would directly influence how '*p*' should be set. However, upon meticulous investigation and rigorous experimentation, it became apparent that the relationship between the IR and '*p*' is far more intricate and nuanced than initially assumed. Through systematic exploration and analysis of various datasets with varying levels of class imbalance, it was discovered that the association between the IR and the optimal '*p*' value is not characterized by a linear or predictable pattern. In fact, the interplay between these factors often demonstrates substantial variability and complexity. Certain datasets with similar IRs exhibited contrasting trends in the optimal '*p*' value, highlighting the influence of additional dataset-specific attributes beyond the IR itself. Consequently, while the IR does indeed play a role in determining the optimal 'p' value, it is by no means the sole determinant.

A notable relationship between the hyperparameter '*p*' and the number of instances within the minority class has emerged from our investigation. Particularly in datasets characterized by a limited count of minority instances, we observed an intriguing trend where higher '*p*' values translated to better method performance. This trend was pronounced when employing a KNN classifier, as '*p*' values around 5 appeared to yield optimal outcomes. On the other hand, when utilizing decision trees (CART), an ideal '*p*' value got around 2. Although this relationship might initially appear somewhat variable, a thorough analysis of the complete range of experiments conducted revealed a consistent pattern. Across all scenarios explored, it was consistently observed that for datasets with a sparse minority class, the optimal '*p*' values were consistently greater than 1. This finding led us to establish a general guideline: regardless of the classifier employed, adopting '*p*' values greater than 1 ensures robust method performance in the majority of cases and scenarios.

For datasets with a substantial number of instances in the minority class, the POS method employs a lower '*p*' value compared to the '*p*' values utilized by the POS 1.0 and POS 2.0 methods. These latter methods adopt values that are relatively modest in magnitude but still somewhat higher. This distinction arises from the fact that POS 1.0 and POS 2.0 introduce more perturbation to instances compared to the original POS method. While the original POS method perturbs a single feature of an instance, the newer versions perturb multiple features.

Due to this difference in perturbation intensity, POS 1.0 and POS 2.0 utilize '*p*' values that are not as low. In this context, a guideline has been established: the POS method employs a '*p*' value of 0.3, while POS 1.0 and POS 2.0 adopt values of 0.7 and 0.9 respectively, for datasets with a higher number of minority class instances.

This approach aligns with the recognition that datasets containing a greater number of minority class instances necessitate a more pronounced level of perturbation. This is because as the number of instances increases, the variations within the minority class become more apparent, requiring a stronger perturbation to amplify these distinctions effectively and improve classification performance.

Upon meticulous examination of the extensive array of experiments and comparative analyses conducted in comparison with existing class balancing methodologies, a resounding affirmation emerges: the POS method exhibits remarkable efficacy and robustness. Its adeptness in mitigating class imbalance issues is apparent across a spectrum of datasets and scenarios.

The new approaches proposed in this study, POS 1.0 and POS 2.0, borne out of an intention to refine and clarify certain nuances in the original POS method, have demonstrated substantial promise. Notably, when subjected to rigorous performance evaluations across a diverse range of scenarios using both KNN and CART classifiers, the new methods consistently outshined the original POS approach in terms of overall performance. The majority of comparisons revealed that these novel methods show superior classification outcomes, sustaining their potential for providing robust solutions to class imbalance. This remarkable consistency shows the potential breakthrough that the POS 1.0 and POS 2.0 methodologies bring to the realm of class imbalance mitigation, offering new avenues for tackling this challenge with effectiveness.

6. References:

[1] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Editorial: special issue on learning from imbalanced data sets. SIGKDD Explor. Newsl. 6, 1 (June 2004), 1–6. https://doi.org/10.1145/1007730.1007733

[2] Amit Singh, Ranjeet Kumar Ranjan & Abhishek Tiwari (2022) Credit Card Fraud Detection under Extreme Imbalanced Data: A Comparative Study of Data-level Algorithms, Journal of Experimental & Theoretical Artificial Intelligence, 34:4, 571-598, DOI: 10.1080/0952813X.2021.1907795

[3] Han, X., Cui, R., Lan, Y., Kang, Y., Jia, N.: A gaussian mixture model based combined resampling algorithm for classification of imbalanced credit data sets. International Journal of Machine Learning and Cybernetics 10, 3687–3699 (2019)

[4] Ng, W.W., Zeng, G., Zhang, J., Yeung, D.S., Pedrycz, W.: Dual autoencoders features for imbalance classification problem. Pattern Recognition 60, 875–889 (2016)

[5] Kamalov, F.: Kernel density estimation-based sampling for imbalanced class distribution. Information Sciences 512, 1192–1201 (2020)

[6] Bellinger, C., Drummond, C., Japkowicz, N.: Manifold-based synthetic oversampling with manifold conformance estimation. Machine Learning 107(3), 605–637 (2018)

[7] J. A. Prieto, «Iaradar.com,» 26 March 2023. [En línea]. Available: "https://iaradar.com/conceptos/descubriendo-auc-un-indicador-clave-en-la-evaluacion-de-modelos-de-clasificacion/," [Online]

[8] «Amirhessam Tahmassebi» 17 Sep 2019. [En línea]. Available: "https://amirhessam88.github.io/roc-vs-pr/," [Online]

[9] Noorhalim, N., Ali, A., Shamsuddin, S.M. (2019). Handling Imbalanced Ratio for Class Imbalance Problem Using SMOTE. In: Kor, LK., Ahmad, AR., Idrus, Z., Mansor, K. (eds) Proceedings of the Third International Conference on Computing, Mathematics and Statistics (iCMS2017). Springer, Singapore. https://doi.org/10.1007/978-981-13-7279-7_3

[10] Schapire, R.E. (2013). Explaining AdaBoost. In: Schölkopf, B., Luo, Z., Vovk, V. (eds) Empirical Inference. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-41136-6_5

[11] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[12] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[13] He, H., & Ma, Y. (Eds.). (2013). Imbalanced learning: foundations, algorithms, and applications.

[14] G. Lemaitre, F. Nogueira, D. Oliveira, C. Aridas., 2016. [En línea]. Available: "http://glemaitre.github.io/imbalanced-learn/auto_examples/under-sampling/plot_random_under_sampler.html#random-under-sampling," [Online]

[15] I. Tomek, "Two Modifications of CNN", IEEE Transactions on Systems Man and Communications SMC-6, 1976, pp. 769-772

[16] Rafael.A. Available: "https://www.kaggle.com/code/rafjaa/resampling-strategies-for-imbalanced-datasets?cellIds=31&kernelSessionId=1756536," [Online]

[17] Hart, P. (1968). The condensed nearest neighbor rule (corresp.). IEEE transactions on information theory, 14(3), 515-516.

[18] Kubat, M., & Matwin, S. (1997, July). Addressing the curse of imbalanced training sets: one-sided selection. In Icml (Vol. 97, No. 1, p. 179).

[19] Effective Class-Imbalance learning based on SMOTE and Convolutional Neural Networks - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Illustration-of-One-Sided-Selection_fig5_363269818 [accessed 23 Aug, 2023]

[20] Laurikkala, J. (2001). 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 (pp. 63-66). Springer Berlin Heidelberg.

[21] G. Lemaitre, F. Nogueira, D. Oliveira, C. Aridas., 2016. [En línea]. Available: "http://glemaitre.github.io/imbalanced-learn/auto_examples/over-sampling/plot_random_over_sampling.html," [Online]

[22] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 321-357.

[23] «Rich Data » 2023. [En línea]. Available: "https://rikunert.com/smote_explained," [Online]

APPENDIX

ADDITIONAL RESULTS FOR THE CONDUCTED EXPERIMENTS

The appendix contains tables with performance results that are similar to those already presented in the main report. Due to their similarity, these tables have been excluded from the main report for the sake of brevity.

"p" values	glass1.dat	glass0.dat	glass-0-1-2-3_vs_4-5-6.dat	new-thyroid1.dat	glass6.dat	ecoli-0-6-7_vs_3-5.dat	glass2.dat	glass4.dat	shuttle-6_vs_2-3.dat
0.1	0.7755	**0.8180**	0.8958	0.9338	0.8578	0.8651	0.2558	0.8462	0.9414
0.2	0.7791	0.8142	0.9017	0.9499	0.8578	0.8887	0.3712	0.8436	0.9414
0.3	0.7858	0.8148	0.8986	0.9499	0.8933	0.8834	0.3682	0.8415	0.9414
0.4	0.7825	0.8074	0.9031	0.9796	0.8908	**0.9022**	0.3639	0.8768	0.9414
0.5	0.7825	0.8140	0.9136	0.9768	0.8908	0.8979	0.3624	0.8768	0.9414
0.6	**0.7878**	0.8140	0.9337	0.9768	**0.9078**	0.8883	0.4598	0.8742	0.9414
0.7	0.7771	0.7970	0.9337	**0.9916**	**0.9078**	0.8654	0.4970	0.8690	0.9414
0.8	0.7574	0.7974	0.9548	**0.9916**	0.9050	0.8606	0.4909	0.8690	0.9414
0.9	0.7672	0.7924	0.9548	**0.9916**	0.9050	0.8606	0.4894	0.8690	0.9414
1.0	0.7636	0.7887	0.9548	**0.9916**	0.9050	0.8606	0.4872	0.8690	0.9414
1.5	0.7645	0.8040	**0.9579**	0.9768	0.9050	0.8624	**0.5711**	**0.9243**	**0.9824**
2.0	0.7645	0.8040	0.9478	0.9888	0.9050	0.8646	**0.5711**	**0.9243**	**0.9824**
3.0	0.7645	0.8040	0.9478	0.9888	0.9050	0.8646	**0.5711**	**0.9243**	**0.9824**
4.0	0.7645	0.8040	0.9478	0.9888	0.9050	0.8646	**0.5711**	**0.9243**	**0.9824**
5.0	0.7645	0.8040	0.9478	0.9888	0.9050	0.8646	**0.5711**	**0.9243**	**0.9824**
IR	1.82	2	3.2	5.14	6.38	9.09	11.59	15.47	22

Table 30: Performance results for low number of instances (200 instances) and varied values for IR. Classifier: KNN, Evaluation metric: GM, Method: POS 1.0, arranged in an ascending order of IR.

"p" values	yeast1.dat	yeast3.dat	yeast4.dat	winequality-red-4.dat	yeast5.dat	yeast6.dat	winequality-white-3-9_vs_5.dat	poker-8-9_vs_6.dat	poker-8_vs_6.dat
0.1	0.6214	0.8478	0.5814	0.2795	0.9271	0.7509	**0.3769**	0.9914	0.9640
0.2	0.6423	0.8552	0.5924	0.4571	0.9410	0.7718	0.3760	0.9904	0.9640
0.3	0.6589	0.8750	0.6242	0.4795	0.9396	0.8234	0.3751	0.9904	0.9637
0.4	0.6757	0.8832	0.6640	0.5270	0.9386	0.8210	0.3747	0.9904	0.9643
0.5	0.6837	0.8871	0.7125	0.5235	0.9520	0.8202	0.3747	0.9914	0.9640
0.6	0.6934	0.8962	0.7242	0.5341	0.9514	0.8199	0.3747	0.9917	0.9646
0.7	**0.6969**	**0.8984**	0.7340	0.5479	**0.9620**	0.8176	0.3747	**0.9935**	0.9657
0.8	0.6949	0.8932	0.7321	**0.5487**	0.9616	0.8345	0.3747	0.9735	**0.9671**
0.9	0.6963	0.8910	0.7330	0.5460	0.9616	0.8502	0.3744	0.9528	0.9405
1.0	0.6944	0.8892	**0.7482**	0.5344	0.9616	**0.8574**	0.3747	0.9292	0.9409
1.5	0.6959	0.8873	0.7152	0.4652	0.9503	**0.8574**	0.3388	0.9093	0.8752
2.0	0.6947	0.8880	0.7015	0.4678	0.9503	**0.8574**	0.3392	0.9102	0.8752
3.0	0.6959	0.8880	0.7015	0.4686	0.9506	0.8256	0.3394	0.9105	0.8396
4.0	0.6959	0.8880	0.7015	0.4686	0.9506	0.8256	0.3394	0.9105	0.8396
5.0	0.6959	0.8880	0.7015	0.4686	0.9506	0.8256	0.3394	0.9105	0.8396
IR	2.46	8.1	28.1	29.7	32.73	41.4	58.28	58.4	85.88

Table 31: Performance results for 'p', for high number of instances (1500 instances) and varied values for IR. Classifier: KNN, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of IR.

"p" values	glass1.dat	glass0.dat	glass-0-1-2-3_vs_4-5-6.dat	new-thyroid1.dat	glass6.dat	ecoli-0-6-7_vs_3-5.dat	glass2.dat	glass4.dat	shuttle-6_vs_2-3.dat
0.1	0.7708	**0.8255**	0.8727	0.9056	0.8195	0.7809	0.1140	0.5370	0.6828
0.2	0.7646	0.8154	0.8671	0.9056	0.8385	0.8702	0.1140	0.5822	0.6828
0.3	0.7547	0.8117	0.8671	0.9217	0.8601	**0.8967**	0.1140	0.6267	0.6828
0.4	0.7663	0.8045	0.9174	0.9527	0.8769	0.8684	0.1140	0.6253	0.7414
0.5	0.7605	0.8120	0.9139	0.9527	0.9078	0.8924	0.1140	0.7629	0.7414
0.6	**0.7853**	0.8049	0.9447	0.9649	**0.9268**	0.8847	0.2265	0.7566	0.7414
0.7	0.7685	0.8112	0.9447	**0.9824**	0.9242	0.8728	0.2265	0.8032	0.9414
0.8	0.7721	0.8116	0.9548	0.9796	0.9242	0.8683	0.2716	0.8690	0.9414
0.9	0.7613	0.7899	0.9481	0.9768	0.9050	0.8864	0.2701	0.8664	0.9414
1.0	0.7584	0.7860	0.9378	0.9739	0.9024	0.8837	0.2686	0.9190	**1.0000**
1.5	0.7585	0.7811	**0.9582**	0.9739	0.9050	0.8829	**0.5782**	0.9189	**1.0000**
2.0	0.7586	0.7811	**0.9582**	0.9739	0.9050	0.8646	0.5644	**0.9243**	**1.0000**
3.0	0.7586	0.7811	**0.9582**	0.9739	0.9050	0.8646	0.5711	**0.9243**	0.9414
4.0	0.7586	0.7811	**0.9582**	0.9739	0.9050	0.8646	0.5711	**0.9243**	0.9414
5.0	0.7586	0.7811	**0.9582**	0.9739	0.9050	0.8646	0.5711	**0.9243**	0.9414
IR	1.82	2	3.2	5.14	6.38	9.09	11.59	15.47	22

Table 32: Performance results for low number of instances (200 instances) and varied values for IR. Classifier: KNN, Evaluation metric: GM, Method: POS 2.0, arranged in an ascending order of IR.

"p" values	yeast1.dat	yeast3.dat	yeast4.dat	winequality-red-4.dat	yeast5.dat	yeast6.dat	winequality-white-3-9_vs_5.dat	poker-8-9_vs_6.dat	poker-8_vs_6.dat
0.1	0.6134	0.8193	0.2785	0.0000	0.8178	0.6864	0.0000	0.6887	0.4307
0.2	0.6127	0.8193	0.3046	0.0000	0.8178	0.6864	0.0893	0.8144	0.7145
0.3	0.6175	0.8255	0.3243	0.0000	0.8308	0.6853	0.0893	0.8833	0.7950
0.4	0.6251	0.8424	0.3756	0.0000	0.8418	0.7151	0.0893	0.9085	0.8241
0.5	0.6648	0.8659	0.4788	0.0630	0.8522	0.7128	0.1263	**0.9153**	0.9080
0.6	0.6815	0.8739	0.6102	0.2704	0.8783	0.7088	**0.5195**	0.9115	**0.9247**
0.7	0.6889	**0.9005**	0.6905	0.3919	0.9593	0.7880	0.4938	0.8965	0.9167
0.8	0.6936	0.8868	0.7721	0.4903	**0.9796**	0.8319	0.4903	0.9071	0.8421
0.9	**0.6958**	0.8864	**0.7925**	0.5243	0.9779	0.8609	0.5160	0.8944	0.8478
1.0	0.6919	0.8868	0.7801	**0.5434**	0.9674	**0.8724**	0.5141	0.8996	0.8563
1.5	0.6879	0.8877	0.7094	0.4656	0.9626	0.8518	0.4267	0.9077	0.8720
2.0	0.6879	0.8880	0.7012	0.4679	0.9499	0.8577	0.3392	0.9105	0.8752
3.0	0.6891	0.8880	0.7015	0.4686	0.9506	0.8256	0.3394	0.9105	0.8396
4.0	0.6891	0.8880	0.7015	0.4686	0.9506	0.8256	0.3394	0.9105	0.8396
5.0	0.6891	0.8880	0.7015	0.4686	0.9506	0.8256	0.3394	0.9105	0.8396
IR	2.46	8.1	28.1	29.17	32.73	41.4	58.28	58.4	85.88

Table 33: Performance results for 'p', for high number of instances (1500 instances) and varied values for IR. Classifier: KNN, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of IR.

"p" values	glass1.dat	glass0.dat	glass-0-1-2-3_vs_4-5-6.dat	new-thyroid1.dat	glass6.dat	ecoli-0-6-7_vs_3-5.dat	glass2.dat	glass4.dat	shuttle-6_vs_2-3.dat
0.1	0.7834	0.7703	0.8890	0.9478	0.9120	0.8365	0.6816	0.8834	0.9954
0.2	0.7791	0.7635	0.8890	0.9115	0.9123	0.8053	0.6324	0.8783	0.9954
0.3	0.7877	**0.8092**	**0.9174**	0.9109	0.9120	**0.8460**	0.6504	0.8803	0.9954
0.4	0.7650	0.7823	0.8888	**0.9684**	0.9145	0.7913	0.5888	0.8425	0.9977
0.5	**0.7993**	0.7374	0.9124	0.9335	0.9094	0.8141	0.6220	0.8797	0.9977
0.6	0.7834	0.7166	0.9064	0.9201	0.9100	0.7648	0.7163	0.8854	0.9977
0.7	0.7498	0.7395	0.9115	0.9371	0.9005	0.7974	0.5876	0.9436	0.9977
0.8	0.7377	0.7618	0.9014	0.9480	0.9220	0.7729	0.6068	0.9436	**1.0000**
0.9	0.7453	0.7685	0.9014	0.9416	0.9027	0.7793	0.6609	0.8560	**1.0000**
1.0	0.7362	0.7670	0.8985	0.9416	**0.9394**	0.8065	**0.7472**	**0.9461**	**1.0000**
1.5	0.7918	0.8005	0.8773	0.9564	0.8835	0.7557	0.6858	0.7322	**1.0000**
2.0	0.7918	0.8005	0.8897	0.9618	0.8835	0.7506	0.4986	0.7762	**1.0000**
3.0	0.7918	0.8005	0.8897	0.9647	0.8835	0.7506	0.3912	0.7721	**1.0000**
4.0	0.7876	0.7899	0.8897	0.9647	0.8835	0.7506	0.3912	0.7721	**1.0000**
5.0	0.7648	0.7899	0.9029	0.9647	0.8835	0.8033	0.3912	0.7721	**1.0000**
IR	1.82	2	3.2	5.14	6.38	9.09	11.59	15.47	22

Table 34: Performance results for low number of instances (200 instances) and varied values for IR. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS, arranged in an ascending order of IR.

"p"values	yeast1.dat	yeast3.dat	yeast4.dat	winequality-red-4.dat	yeast5.dat	yeast6.dat	winequality-white-3-9_vs_5.dat	poker-8-9_vs_6.dat	poker-8_vs_6.dat
0.1	0.6405	0.8257	0.6543	0.4888	0.8915	0.6944	0.4878	0.4903	0.3747
0.2	0.6524	0.8472	0.6490	**0.5033**	0.9143	0.6692	0.4499	**0.6026**	0.1613
0.3	0.6675	0.8573	0.7264	0.4346	0.9023	0.7413	0.4219	0.4182	0.2602
0.4	0.6446	**0.8589**	0.6579	0.4989	0.9149	0.7494	0.3864	0.5732	0.4181
0.5	0.6439	0.8333	0.7333	0.4386	0.9135	0.7543	0.4243	0.4246	0.4718
0.6	**0.6684**	0.8531	**0.7648**	0.3798	0.9008	0.7774	0.3991	0.4178	**0.4722**
0.7	0.6555	0.8062	0.6664	0.4182	**0.9294**	**0.7924**	**0.5101**	0.4264	0.3733
0.8	0.6628	0.8017	0.6686	0.3843	0.9142	0.7119	0.3988	0.4182	0.3756
0.9	0.6582	0.8035	0.6949	0.3715	0.8729	0.7818	0.3989	0.2137	0.3770
1.0	0.6505	0.8064	0.6540	0.4526	0.8538	0.7499	0.4270	0.0000	0.1630
1.5	0.6505	0.7967	0.5957	0.2678	0.8207	0.6140	0.2499	0.0891	0.2633
2.0	0.6494	0.7967	0.5950	0.2083	0.8336	0.5973	0.2797	0.0891	0.2633
3.0	0.6276	0.7929	0.5950	0.2083	0.8336	0.5973	0.2517	0.0891	0.2633
4.0	0.6269	0.7846	0.5466	0.2083	0.7861	0.5973	0.2517	0.0891	0.2633
5.0	0.6269	0.7846	0.5326	0.2086	0.7893	0.5973	0.2517	0.0891	0.2633
IR	2.46	8.1	28.1	29.17	32.73	41.4	58.28	58.4	85.88

Table 35: Performance results for 'p', for high number of instances (1500 instances) and varied values for IR. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS, arranged in ascending order of IR.

"p" values	glass-0-4_vs_5	glass-0-6_vs_5	glass-0-1-5_vs_2	ecoli-0-3-4_vs_5	ecoli-0-1_vs_2-3-5	ecoli-0-3-4-7_vs_5-6	yeast-2_vs_4	yeast-0-2-5-7-9_vs_3-6-8	yeast-0-2-5-6_vs_3-7-8-9	page-blocks0
0.1	0.9815	0.97906	0.6207	0.9081	0.8613	0.8772	0.8235	0.8422	0.6860	0.9277
0.2	0.9815	0.97933	0.5527	0.9024	0.8653	0.8957	0.8244	0.8555	0.7283	0.9227
0.3	0.9877	0.97426	0.5553	0.9423	0.8563	**0.9078**	0.8240	0.8424	0.7182	**0.9374**
0.4	0.9877	0.97426	0.5108	0.9452	0.8327	0.9004	0.8563	0.8582	0.7130	0.9245
0.5	0.9877	0.97933	0.6127	0.8757	0.8109	0.9050	0.8274	0.8630	**0.7440**	0.9201
0.6	0.9692	0.97933	0.6525	**0.9477**	0.8172	0.8893	**0.8746**	**0.8640**	0.6736	0.9156
0.7	0.9873	0.97933	**0.6588**	0.9242	0.8426	0.8650	0.7965	0.8549	0.6895	0.9254
0.8	0.9873	**0.98439**	0.5119	0.9035	0.8189	0.8704	0.8659	0.8601	0.6741	0.9167
0.9	0.9873	0.96787	0.5147	0.8725	0.7937	0.8884	0.8572	0.8243	0.6837	0.9238
1.0	0.9873	0.96787	0.5659	0.8196	0.8539	0.8706	0.8459	0.8630	0.6858	0.9240
1.5	0.9873	0.97933	0.4597	0.8484	0.8452	0.8933	0.8483	0.8568	0.6623	0.9139
2.0	0.9287	0.93636	0.5756	0.8484	**0.8668**	0.8189	0.8483	0.8568	0.6623	0.9123
3.0	0.9873	0.93116	0.4203	0.8484	**0.8668**	0.8189	0.8483	0.8563	0.6872	0.9115
4.0	**0.9936**	0.93636	0.4203	0.8484	**0.8668**	0.8189	0.8483	0.8511	0.6674	0.9115
5.0	**0.9936**	0.93636	0.4203	0.8484	**0.8668**	0.7967	0.8612	0.8508	0.6424	0.9115
IR	9.22	10.00	9.12	9.00	9.17	9.28	9.08	9.14	9.14	8.79
dim	92	108	172	200	244	257	514	1004	1004	5472

Table 36: Performance results for the different values of 'p' setting IR fixed at an average value of 9 and varying the number of instances . Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS, arranged in ascending order of number of instances

"p"values	glass1.dat	glass0.dat	glass-0-1-2-3_vs_4-5-6.dat	new-thyroid1.dat	glass6.dat	ecoli-0-6-7_vs_3-5.dat	glass2.dat	glass4.dat	shuttle-6_vs_2-3.dat
0.1	0.6910	0.8434	0.9256	0.9440	0.8902	0.8221	0.5453	0.6723	0.9954
0.2	0.7197	**0.8618**	0.9038	0.9157	0.9042	0.8124	0.5831	0.6113	0.9954
0.3	0.7490	0.8089	0.9191	0.9510	0.8927	0.8774	0.5049	0.8702	0.9954
0.4	0.7191	0.7994	0.9024	0.9157	0.8927	0.8378	0.5116	0.8597	0.9954
0.5	0.7153	0.7539	0.9161	0.9419	0.9212	**0.8563**	0.6097	0.6616	0.9954
0.6	**0.7820**	0.7914	0.9356	0.9452	0.9094	0.8264	0.6907	**0.9482**	0.9977
0.7	0.7236	0.8109	**0.9385**	0.9421	0.9145	0.8284	0.6969	0.8465	0.9977
0.8	0.7311	0.8273	0.9255	0.9208	0.9120	0.8284	0.6969	0.8296	0.9977
0.9	0.7536	0.7771	0.9255	0.9231	0.8930	0.8495	**0.6991**	0.8959	0.9954
1.0	0.7494	0.8049	0.9236	**0.9647**	**0.9265**	0.8513	**0.6991**	0.8648	0.9977
1.5	0.7066	0.7802	0.9013	0.9618	0.9262	0.8244	0.6510	0.8713	**1.0000**
2.0	0.6980	0.7915	0.8905	0.9618	0.9122	0.7506	0.6602	0.8088	**1.0000**
3.0	0.6990	0.7915	0.8905	0.9618	0.9122	0.7506	0.5668	0.7721	**1.0000**
4.0	0.7880	0.7969	0.8905	0.9618	0.9122	0.7506	0.5668	0.7721	**1.0000**
5.0	0.7648	0.7899	0.9029	0.9618	0.8810	0.7506	0.2988	0.7721	**1.0000**
IR	1.82	2	3.2	5.14	6.38	9.09	11.59	15.47	22

Table 37: Performance results for low number of instances (200 instances) and varied values for IR. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 1.0, arranged in an ascending order of IR.

"p" values	yeast1.dat	yeast3.dat	yeast4.dat	winequality-red-4.dat	yeast5.dat	yeast6.dat	winequality-white-3-9_vs_5.dat	poker-8-9_vs_6.dat	poker-8_vs_6.dat
0.1	**0.6496**	0.8071	0.5695	0.2624	0.8563	**0.6753**	0.3391	0.9539	0.9997
0.2	**0.6496**	0.8071	0.5695	0.1848	0.8563	**0.6753**	0.3893	**1.0000**	0.9997
0.3	**0.6496**	0.8057	0.5695	0.3266	0.8161	**0.6753**	0.3008	**1.0000**	0.9997
0.4	**0.6496**	0.8028	0.5695	0.3828	0.7598	**0.6753**	0.3874	**1.0000**	**1.0000**
0.5	**0.6496**	0.8038	0.5437	0.3276	0.7437	0.5783	0.4758	**1.0000**	**1.0000**
0.6	**0.6496**	0.7786	0.5864	**0.4924**	0.7866	0.6333	**0.5490**	0.9547	**1.0000**
0.7	**0.6496**	**0.8075**	0.5442	0.2856	0.7920	0.5957	0.3007	0.9549	**1.0000**
0.8	**0.6496**	0.7713	0.5867	0.4038	0.8019	0.5961	0.3354	0.9549	**1.0000**
0.9	**0.6496**	0.7751	0.5745	0.4797	0.8181	0.5961	0.3004	0.9777	0.8559
1.0	0.6327	0.7941	**0.6407**	0.3941	**0.8687**	0.5528	0.3644	0.8000	0.8559
1.5	0.6327	0.7867	0.6261	0.2421	0.8527	0.6015	0.4250	0.7549	0.7365
2.0	0.6327	0.7858	0.6261	0.2414	0.8527	0.6562	0.1783	0.7549	0.7365
3.0	0.6397	0.7780	0.6261	0.2659	0.8656	0.6750	0.3043	0.7549	0.7365
4.0	0.6269	0.7843	0.5162	0.2084	0.8141	0.5973	0.3043	0.6441	0.7365
5.0	0.6269	0.7846	0.5326	0.2086	0.8023	0.5973	0.2517	0.0891	0.0000
IR	2.46	8.1	28.1	29.17	32.73	41.4	58.28	58.4	85.88

Table 38: Performance results for 'p', for high number of instances (1500 instances) and varied values for IR. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of IR.

"p" values	glass-0-4_vs_5.dat	glass-0-6_vs_5.dat	glass-0-1-5_vs_2.dat	ecoli-0-3-4_vs_5.dat	ecoli-0-1_vs_2-3-5.dat	ecoli-0-3-4-7_vs_5-6.dat	yeast-2_vs_4.dat	yeast-0-2-5-7-9_vs_3-6-8.dat	yeast-0-2-5-6_vs_3-7-8-9.dat	page-blocks0.dat
0.1	0.9813	0.9276	0.4987	0.8465	0.7470	0.8474	**0.8935**	0.8486	**0.6797**	0.9049
0.2	0.9813	0.7077	0.3811	0.8485	0.7742	0.8679	0.8605	0.8484	**0.6797**	0.9017
0.3	0.9748	0.7114	0.6744	0.8717	0.7988	0.8509	0.8597	0.8544	**0.6797**	0.9002
0.4	0.9813	0.5633	0.6036	0.8485	0.9024	0.8694	0.8484	**0.8657**	0.6727	0.9120
0.5	0.9813	0.5688	0.6009	0.8746	0.8211	**0.8982**	0.8247	0.8479	0.6727	0.9117
0.6	0.9813	0.5077	0.5954	0.8432	**0.8847**	0.8922	0.8151	0.8591	0.6727	0.9103
0.7	0.9877	0.7847	0.6514	**0.8774**	0.8207	0.8940	0.8339	0.8420	0.6727	0.9164
0.8	0.9940	0.7051	0.5457	0.8428	0.7819	0.8848	0.8572	0.8476	0.6727	0.9159
0.9	0.9940	0.9207	0.5457	0.8147	0.7797	0.8868	0.8521	0.8346	0.6727	**0.9233**
1.0	0.9940	0.9795	**0.6518**	0.8134	0.8201	0.8534	0.8397	0.8486	0.6727	0.9171
1.5	0.9811	0.9738	0.5421	0.8195	0.8343	0.8743	0.8371	0.8481	0.6727	0.9158
2.0	**1.0000**	**0.9844**	0.5485	0.8484	0.8624	0.8306	0.8361	0.8481	0.6727	0.9132
3.0	0.9873	0.9312	0.5485	0.8484	0.8624	0.8306	0.8673	0.8611	0.6450	0.9143
4.0	0.9936	0.9364	0.5485	0.8484	0.8624	0.8498	0.8152	0.8454	0.6496	0.9143
5.0	0.9936	0.9364	0.3806	0.8484	0.8645	0.8172	0.8603	0.8508	0.6424	0.9115
IR	9.22	10.00	9.12	9.00	9.17	9.28	9.08	9.14	9.14	8.79
dim	92	108	172	200	244	257	514	1004	1004	5472

Table 39: Performance results for the different values of 'p' setting IR fixed at an average value of 9 and varying the number of instances. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 1.0, arranged in ascending order of number of instances

"p" values	glass1.dat	glass0.dat	glass-0-1-2-3_vs_4-5-6.dat	new-thyroid1.dat	glass6.dat	ecoli-0-6-7_vs_3-5.dat	glass2.dat	glass4.dat	shuttle-6_vs_2-3.dat
0.1	0.6931	0.7512	0.9191	0.9508	0.9080	0.8244	**0.7262**	0.7128	0.9977
0.2	0.6969	0.7749	0.9132	0.9312	0.9236	0.8513	0.6593	0.7107	0.9977
0.3	0.7450	0.7339	0.9132	0.9451	0.9212	0.8226	0.5316	0.6604	0.9977
0.4	0.7004	0.7904	0.9085	0.9349	0.9070	0.8796	0.6850	0.7419	0.9954
0.5	0.6762	0.7462	**0.9257**	**0.9916**	0.9045	0.8820	0.6873	0.4534	0.9931
0.6	0.7350	0.7647	0.9163	0.9539	0.9120	**0.8950**	0.6870	0.8415	0.9954
0.7	0.7245	**0.8305**	0.9032	0.9458	0.9198	0.8805	0.6969	0.8529	0.9931
0.8	0.7556	0.8228	0.9024	0.9537	0.9074	0.8730	0.6956	0.8891	0.9977
0.9	0.7471	0.8022	0.9021	0.9482	0.9170	0.8262	0.6961	0.9288	0.9977
1.0	0.7549	0.8006	0.8761	0.9536	0.9239	0.8262	0.6985	**0.9415**	**1.0000**
1.5	**0.7579**	0.8164	0.9150	0.9564	**0.9367**	0.8344	0.6103	0.7968	**1.0000**
2.0	0.7504	0.8140	0.9088	0.9430	0.8788	0.8344	0.6156	0.7707	**1.0000**
3.0	0.7564	0.8065	0.9088	0.9429	0.8788	0.8344	0.6168	0.7721	**1.0000**
4.0	0.7564	0.8028	0.9117	0.9429	0.8788	0.8344	0.6168	0.7721	**1.0000**
5.0	0.7456	0.7987	0.8995	0.9458	0.8788	0.7532	0.4607	0.7721	**1.0000**
IR	1.82	2	3.2	5.14	6.38	9.09	11.59	15.47	22

Table 40: Performance results for low number of instances (200 instances) and varied values for IR. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 1.0, arranged in an ascending order of IR.

"p" values	yeast1.dat	yeast3.dat	yeast4.dat	winequality-red-4.dat	yeast5.dat	yeast6.dat	winequality-white-3-9_vs_5.dat	poker-8-9_vs_6.dat	poker-8_vs_6.dat
0.1	**0.6638**	0.8040	0.5889	0.4446	0.8401	0.6692	0.2660	0.2145	0.1732
0.2	**0.6638**	0.8028	0.5889	0.3187	0.8401	0.6685	0.4426	0.1780	0.3365
0.3	**0.6638**	0.8023	0.5695	0.4995	0.8532	0.6692	0.5870	0.5975	0.7732
0.4	**0.6638**	0.8105	0.5695	0.4254	0.7571	0.6692	0.5866	0.9542	0.8991
0.5	**0.6638**	0.8089	0.5437	0.4817	0.7560	0.6680	0.6188	0.9092	0.8998
0.6	**0.6638**	0.7987	0.5604	0.4793	0.7580	0.5785	0.5976	**0.9549**	0.8986
0.7	**0.6638**	0.7979	0.5441	0.5364	0.7885	0.5352	0.5643	**0.9549**	**0.9729**
0.8	**0.6638**	0.8146	0.5862	0.5146	0.8358	0.5787	**0.7239**	**0.9549**	0.9458
0.9	**0.6638**	0.8173	0.5708	**0.5390**	0.7877	0.5520	0.6149	**0.9549**	0.8885
1.0	**0.6638**	**0.8176**	0.5715	0.5332	0.8109	0.6516	0.4830	0.9533	0.7732
1.5	**0.6638**	0.8142	**0.6418**	0.3320	**0.8958**	**0.6740**	0.3652	0.9536	0.7359
2.0	0.6415	0.7901	0.5237	0.2682	0.8952	0.6555	0.3035	0.9536	0.7359
3.0	0.6415	0.8088	0.5237	0.2683	0.8526	0.6314	0.1773	0.9536	0.7359
4.0	0.6332	0.8031	0.4246	0.1468	0.7897	0.6302	0.1773	0.8153	0.7359
5.0	0.6332	0.8031	0.4254	0.1657	0.8025	0.6441	0.1773	0.0000	0.3357
IR	2.46	8.1	28.1	29.17	32.73	41.4	58.28	58.4	85.88

Table 41: Performance results for 'p', for high number of instances (1500 instances) and varied values for IR. Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of IR.

"p" values	glass-0-4_vs_5.dat	glass-0-6_vs_5.dat	glass-0-1-5_vs_2.dat	ecoli-0-3-4_vs_5.dat	ecoli-0-1_vs_2-3-5.dat	ecoli-0-3-4-7_vs_5-6.dat	yeast-2_vs_4.dat	yeast-0-2-5-7-9_vs_3-6-8.dat	yeast-0-2-5-6_vs_3-7-8-9.dat	page-blocks0.dat
0.1	0.9748	0.9475	0.4570	0.8385	0.7248	0.8280	0.8468	0.8424	0.6521	0.9065
0.2	0.9232	0.9476	0.4456	0.8101	0.7419	0.8280	0.8511	0.8595	0.6521	0.8995
0.3	0.9499	0.9744	0.5024	0.8411	0.7237	0.8260	0.8705	0.8694	0.6581	0.9185
0.4	0.9312	0.9149	**0.6589**	0.8673	0.7237	0.8283	0.8705	0.8543	0.6651	0.9129
0.5	0.9692	0.8539	0.5087	0.8669	0.7443	0.8699	**0.8739**	0.8495	0.6651	0.9133
0.6	0.9680	0.8297	0.6145	0.8586	0.7990	0.8698	0.8693	0.8692	0.6570	0.9130
0.7	0.9626	0.8211	0.6426	**0.8985**	0.8263	0.8738	0.8708	0.8692	0.6570	0.9164
0.8	0.9688	0.8963	0.6458	0.8726	0.8231	0.8758	0.8531	0.8634	0.6570	0.9243
0.9	0.9877	0.7579	0.6445	0.8701	0.8585	**0.8920**	0.8701	0.8529	0.6570	**0.9331**
1.0	**0.9940**	0.8985	0.6089	0.8701	**0.8706**	0.8696	0.8687	**0.8730**	**0.6869**	0.9195
1.5	0.9308	0.9793	0.5412	0.8534	0.7317	0.8554	0.8274	0.8349	0.6570	0.9183
2.0	0.9748	**0.9897**	0.5485	0.8509	0.7749	0.8341	0.8274	0.8349	0.6570	0.9074
3.0	0.9351	0.9312	0.5485	0.8509	0.7749	0.8341	0.8274	0.8579	0.6762	0.9073
4.0	0.9936	0.9364	0.5681	0.8509	0.7749	0.8341	0.8382	0.8470	0.6762	0.9077
5.0	0.9936	0.9364	0.5320	0.8509	0.7751	0.7903	0.8594	0.8465	0.6714	0.9077
IR	9.22	10.00	9.12	9.00	9.17	9.28	9.08	9.14	9.14	8.79
dim	92	108	172	200	244	257	514	1004	1004	5472

Table 42: Performance results for the different values of 'p' setting IR fixed at an average value of 9 and varying the number of instances . Classifier: CART Decision Tree, Evaluation metric: GM, Method: POS 2.0, arranged in ascending order of number of instances