

# Density-Weighted Support Vector Machines for Binary Class Imbalance Learning

April 28, 2024

Priyansh Jaseja - 200001063

Kirtika Zanzan - 200002085

Atharva Mohite - 200003016

Tanishq Selot - 200003076

Bhavya Gupta - 200004008

## Introduction

In the proposal, we looked at the problem of class-imbalanced learning and how we can use the SVM versions described in [Hazarika et al.](#) to tackle it. We planned on performing various methods like undersampling and oversampling and comparing them with the algorithms proposed in the paper.

In this report, we solve the primal problems associated with Density-weighted SVM for binary class imbalance learning (DSVM-CIL) and Improved density-weighted least squares SVM for binary class imbalance learning (IDLSSVM-CIL) by writing their dual forms. Following this, we will compile the results obtained after training the various approaches on the [Keel dataset](#) - binary class imbalanced dataset. We will compare the performance of SVM, LSSVM, and IDLSSVM-CIL both with and without the utilization of oversampling and undersampling techniques. Lastly, we will evaluate the effectiveness of the various sampling methods discussed.

## Dual form(s) and their solutions

### I. Density-weighted SVM for binary class imbalance learning (DSVM-CIL):

This variant involves multiplying the density weightage obtained from the KNN distance technique used by [Cha et al.](#)<sup>[5]</sup> directly to the slack variable. The primal problem can hence be given as:

$$\begin{aligned} \min & \frac{1}{2} \|w\|^2 + Cd\psi, \\ \text{s.t., } & Y(\phi(X)w + eb) \geq e - \psi, \psi \geq 0. \end{aligned}$$

$\phi(x)^t w + eb = 0$  is the hyperplane

Here,  $e$  and  $\Psi$  are vectors of ones and slack variables.  $C > 0$  is the trade-off parameter.  $d$  is the density weight derived using KNN. The unknown parameters  $w$  and  $b$  can be found by converting the above objective function to a dual problem using the Karush-Kuhn-Tucker condition:

$$\begin{aligned} \min & \frac{1}{2} \partial^t Y \phi(X) \phi(X)^t Y \partial - e^t \partial, \\ \text{s.t., } & 0 \leq \partial \leq Cd, \quad y^t \partial = 0. \end{aligned}$$

Where  $\delta \geq 0$  is the lagrangian multiplier, and  $Y$  is the diagonal matrix of  $y$ . This expression can be rewritten using the kernel function  $K(X, X^T)$  as:

$$\begin{aligned} \min \quad & \frac{1}{2} \partial^T Y K(X, X^T) Y \partial - e^T \partial, \\ \text{s.t.,} \quad & 0 \leq \partial \leq C d, \quad y^T \partial = 0. \end{aligned}$$

The unknown variables  $w$  and  $b$  can be found by deriving the above expression with respect to  $\delta$  and then equating it to zero. The final function of this approach can be written as:

$$f(x) = \text{sign}(\phi(x^T)w + b).$$

The  $\text{sign}()$  denotes the 'signum' function here.

## II. Improved density-weighted least squares SVM for binary class imbalance learning (IDLSSVM-CIL):

To reduce the computational cost of the proposed DSVM-CIL, we propose a least squares version of DSVM-CIL as IDLSSVM-CIL. The proposed IDLSSVM-CIL searches for a classifying hyperplane:

$$K(x^t, X^T)w + eb = 0$$

Here,  $K$  is a nonlinear Kernel function.

which can be obtained by solving the primal problem:

$$\begin{aligned} \min \quad & \frac{1}{2} (\|w\|^2 + b^2) + \frac{C}{2} (D\psi)^T (D\psi), \\ \text{s.t.,} \quad & Y(\phi(X)w + eb) = e - \psi, \end{aligned}$$

where  $D$  is a diagonal matrix obtained from affinity and class probabilities discussed in [Tao et al.](#)<sup>[6]</sup>

Now, substituting the constraint from Least squares SVM for  $\Psi$ :

$$Y(\phi(X)w + eb) = e - \psi.$$

with the non-linear kernel function  $K(X, X^T)$ , we can rewrite the equation as:

$$\begin{aligned} \min \quad & \frac{1}{2} (\|w\|^2 + b^2) + \frac{C}{2} (D(e - Y(K(X, X^T)w \\ & + eb)))^T (D(e - Y(K(X, X^T)w + eb))). \end{aligned}$$

Now, considering  $G = [K(X, X^T) \ e]$  and  $v = [w \ b]^T$  and then deriving the obtained expression with respect to  $v$  and equating it to zero, we can obtain the following:

$$v = \begin{bmatrix} w \\ b \end{bmatrix} = \left( \frac{I}{C} + G^t Y^t D D^t Y G + \varepsilon I \right)^{-1} G^t Y^t D D^t e$$

Here,  $I$  is the identity matrix of suitable dimensions. Again, the same decision classifier (final function) as DSVM-CIL can be used for inference.

IDLSSVM-CIL simply solves the system of linear equations by using the equality constraints and considering the 2-norm squared of the slack vector. Because of this, IDLSSVM-CIL is computationally faster compared to DSVM-CIL.

## Sampling Methods

### I. Oversampling:

Oversampling is a technique used to address imbalanced datasets in machine learning. This imbalance can trick machine learning models into prioritizing the majority class and performing poorly on the minority class. Oversampling specifically tackles this by increasing the number of data points in the minority class. The goal of oversampling is to create a more balanced dataset where the model pays fairer attention to all the classes. Some oversampling techniques are -

- a. **SMOTE (Synthetic Minority Oversampling Technique)**<sup>[2]</sup>: SMOTE identifies existing data points in the minority class and creates new points along the line segment between them and their nearest neighbors. This helps generate data that are similar to real-world examples (schematic representation below).

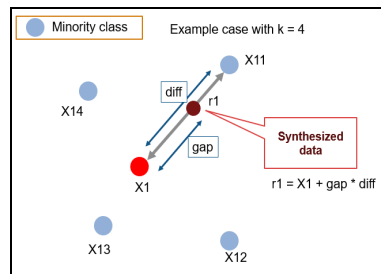


Fig 1: SMOTE oversampling

- b. **Borderline-SMOTE**<sup>[7]</sup>: This variant focuses on oversampling data points from the minority class that lie on the "borderline" between the classes. These borderline points are crucial for good model performance as they are often the most confusing for the model.
- c. **KMeans-SMOTE**<sup>[8]</sup>: This is another oversampling technique that combines K-Means clustering with SMOTE. It works in three main steps:
  - i. **Cluster**: Cluster the data using K-Means.
  - ii. **Filter**: Focus on clusters with a high minority class presence.
  - iii. **Oversample**: Apply SMOTE within these clusters to generate synthetic minority class data points.
- d. **ADASYN**<sup>[9]</sup> (**Adaptive Synthetic Minority Oversampling Technique**): This variant builds on SMOTE by focusing on "difficult" or "hard" examples in the minority class. These are data points closer to the majority class and pose a challenge for the model. ADASYN assigns a higher probability of synthetic data generation to these hard examples, aiming to improve the model's handling of borderline cases.

But the difference from SMOTE here is it considers the density distribution, which decides the no. of synthetic instances generated for samples which are difficult to learn. Due to this, it helps in adaptively changing the decision boundaries based on the samples difficult to learn. This is the major difference compared to SMOTE.

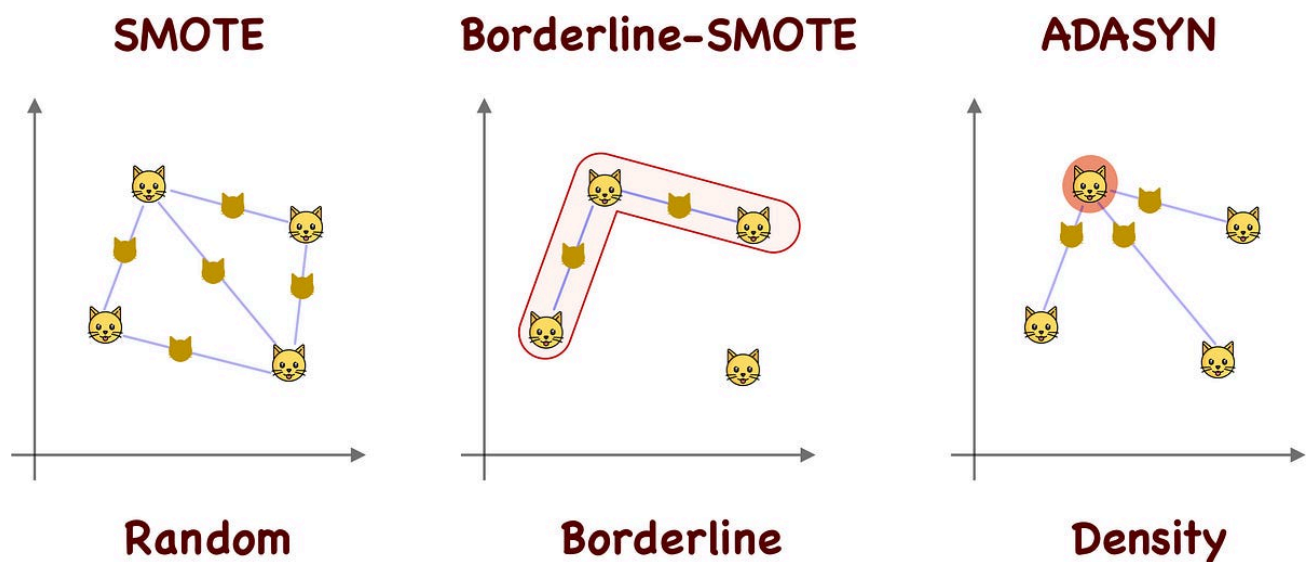


Fig 2: SMOTE vs Borderline-SMOTE vs ADASYN

## II. Undersampling:

Undersampling tackles this by reducing the number of data points in the majority class, essentially shrinking it to a size comparable to the minority class. This helps to create a more balanced dataset where the model is forced to pay attention to all classes.

- a. **NearMiss:** This technique focuses on removing majority class data points that are closest to the minority class in terms of feature similarity. The idea is to remove points that are likely to need to be clarified for the model and have minimal impact on learning the true decision boundary.



Fig 3: NearMiss undersampling

- b. **Condensed Nearest Neighbour:** CNN, or CondensedNN, aims to find a subset of samples from a dataset called a minimum consistent set, maintaining model performance. It iterates through the dataset, adding samples to the group only if they can't be correctly classified. This method addresses the high memory demands of the K-Nearest Neighbors (KNN) algorithm.
- c. **TomekLinks:** The main goal of using Tomek Links is to improve the performance of classifiers by making the decision boundary between classes more clear. Also, we can use Tomek Links to reduce noises even if our data is balanced.

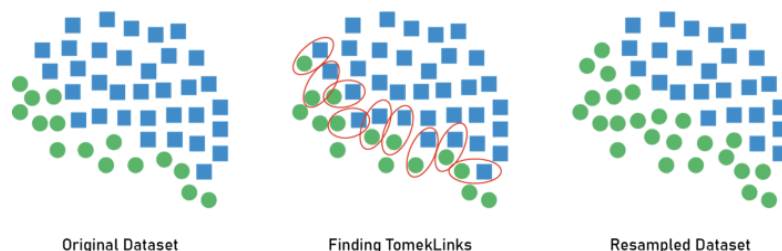


Fig 4: TomekLinks undersampling

### III. Oversampling-Undersampling Combined:

- a. **SMOTEENN**: Introduced by Batista et al. in 2004, this approach combines the SMOTE technique, which generates synthetic examples for the minority class, with the ENN method, which removes observations from both classes identified as having a different class compared to their K-nearest neighbor majority class.

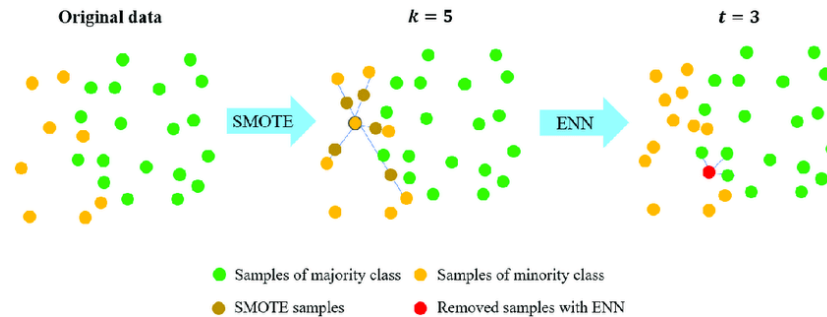


Fig 5: Illustration of SMOTE-ENN combined sampling method

- b. **SMOTE-Tomek Links**: This variant combines oversampling with undersampling. It identifies "Tomek Links" - pairs of closest neighbors where one belongs to the majority class and the other to the minority class. These points represent decision boundaries. SMOTE-Tomek Links oversamples the minority class data point and undersamples the majority class data point in the Tomek Link pair.

## Methodology

Please find the inference script to train and test SVM variants with oversampling [here](#). Below is a walkthrough of the sections in this notebook.

### 1. Initialization

- The code begins by importing pandas, matplotlib, and numpy for data handling, visualization, and numerical operations, along with various imbalanced-learn techniques such as oversampling (SMOTE, BorderlineSMOTE, KMeansSMOTE, ADASYN) and undersampling (NearMiss, CondensedNearestNeighbour, TomekLinks), as well as combined techniques (SMOTEENN, SMOTETomek). It also imports SVM and confusion\_matrix from scikit-learn for classification tasks and evaluation.
- It defines lists for datasets, sampling techniques, and models, providing a structured reference for different options. Parameters like dataset\_no, over\_or\_under\_no, and model\_no are set to select specific combinations for experimentation. The split\_ratio determines data split for training and

testing, and `optimized_params` store the optimized hyperparameters as prescribed in [1].

- Finally, the code prints the selected dataset, sampling technique, and model to verify the chosen configurations for further operations.

## 2. Loading the dataset

- The code constructs a file path to a CSV file within a specified directory. The CSV file is read into a pandas DataFrame, assuming it has no header row and specifying column names as 'feature1', 'feature2', and 'class'.
- The DataFrame is processed to prepare it for analysis. It skips the first 7 rows of the DataFrame, which may contain metadata or header information. The index of the DataFrame is reset after skipping rows, ensuring a clean, sequential index. All values in the DataFrame are converted to integers, likely to ensure consistency and facilitate numerical analysis.
- Basic EDA determines class distribution by calculating the frequency of each unique value in the 'class' column. The imbalance ratio between classes is computed, indicating class imbalance, with values  $>1$  favoring the class labeled as 0. Counts of each class are printed, providing a detailed breakdown of label distribution.
- In essence, the code prepares the dataset for analysis by cleaning and formatting it appropriately and provides initial insights into the class distribution and imbalance ratio

## 3. Normalisation (min-max scaling)

- The minimum and maximum values are computed for each feature (feature1 and feature2) in the dataset (df). These minimum and maximum values represent the range of each feature across the dataset.
- For each instance in the dataset, the normalization formula is applied individually to feature1 and feature2. The formula subtracts the minimum value of the respective feature and divides it by the difference between the maximum and minimum values of that feature. This process ensures that all values of feature1 and feature2 are scaled proportionally between 0 and 1, maintaining the relative relationships among data points.
- Normalization is crucial in machine learning and data analysis to ensure that features with different scales do not bias the model or analysis. It prevents features with larger numerical values from dominating the calculation or influencing the results disproportionately. By normalizing the data, the model's performance can improve, and the interpretation of results becomes more meaningful and fair across all features.



#### 4. Applying RBF kernel (only for visualization)

- Kernel approximation transforms data into a higher-dimensional space using kernel functions like the Radial Basis Function (RBF) kernel, capturing non-linear relationships.
- The original dataset's distribution and separability are visualized in a scatter plot based on class labels ('Negative' and 'Positive').
- The RBF kernel transformation is applied using RBFSampler with a gamma parameter ( $\gamma = \mu$ ) and creates a new dataset (trans\_df) with transformed features (RBF\_1 and RBF\_2).
- The scatter plot of the transformed dataset (trans\_df) shows how the RBF kernel alters data distribution in a higher-dimensional space, potentially enhancing class separability.
- RBF kernel transformation is effective for improving data representation, aiding machine learning models in handling non-linear relationships and improving classification performance.
- Please note that the RBF kernel is not applied here. This section is only for visualization. In practice, the RBF kernel outputs more than just two features.

#### 5. Train-test-split

- The dataset is divided into subsets based on class labels (0 and 1) to create df\_0 and df\_1.
- The training set size is calculated using a specified split ratio (split\_ratio), ensuring a balanced representation of classes in the training data (df\_train).
- The remaining data forms the test set (df\_test).
- Features (X\_train, X\_test) are extracted from the datasets, containing relevant columns.
- Labels (y\_train, y\_test) are extracted, representing class labels for training and testing.

#### 6. Oversampling or Undersampling (with visualization)

- The code creates a scatter plot to visualize the distribution of the original training set, highlighting 'Negative' and 'Positive' instances in blue and red colors, respectively.
- If oversampling or undersampling is selected (over\_or\_under\_no > 0), the code applies the chosen technique (SMOTE, BorderlineSMOTE, etc.) to balance class distribution based on specified parameters.

- After sampling, another scatter plot shows the balanced training set, indicating the effectiveness of the sampling technique in addressing class imbalance.
- These visualizations demonstrate how sampling techniques modify class distribution, crucial for improving machine learning model performance on imbalanced datasets.
- Note that because undersampling methods CondensedNN and TomekLinks are dataset methods, no `sampling_strategy` parameter can be set for them.

## 7. Defining Helper Functions

- `rbf(x_i, x_j, sigma)`: Calculates the Radial Basis Function (RBF) kernel between two sets of data points using a parameter  $\mu$ .
- `compute_density_weight(X)`: Given the `X_train` matrix, this function returns the density weight diagonal matrix with fixed  $k=5$  as mentioned in [1].
- `make_meshgrid(h)`: Generates a meshgrid for visualization purposes, spanning from 0 to 1 along both x and y axes with a specified step size. Used when visualizing the SVM non-linear hyperplane.
- Above described helper functions play critical roles in the implementation of a LSSVM and IDLSSVM-CIL model with an RBF kernel.

## 8. Training the Models

### a. Support Vector Machine (SVM) - Model No. 0:

- The current time is recorded as the start time.
- An SVM model is instantiated with specified parameters ( $C$ ,  $\mu$ ) using the Radial Basis Function (RBF) kernel and balanced class weights.
- The model is trained on the training data (`X_train`, `y_train`) and predictions are made on the test data (`X_test`).
- The current time is recorded as the end time, and the computation time is calculated as the difference between the end and start times.

### b. Least Squares SVM (LSSVM) - Model No. 1:

- `fit_lssvm(X, y, C_, mu_)`: This function fits a LSSVM model with given input data, labels, regularization parameter ( $C$ ), and kernel width ( $\mu$ ), returning the model parameters ( $\alpha$  and  $b$ ).
- `predict_lssvm(sv_X, X, y, alpha_, b_, mu_)`: This function predicts labels using the LSSVM model with given support vectors, input data, labels, model parameters, and kernel width.
- Predicts labels for the test data using the trained LSSVM model.
- Measures the computation time for training and prediction processes.

c. *IDLSSVM-CIL - Model No. 2:*

- `fit_idlssvmcil(X, y, D, C_, mu_, E_)`: This function fits an Improved Density-based Least Squares Support Vector Machine with Class Imbalance Learning (IDLSSVM-CIL) model using input data, labels, density matrix, and specified parameters (C, mu, E).
- Unlike the implementation method prescribed in [1], a different formula is used to determine the Lagrange multipliers  $\alpha$  and bias term  $b$  for IDLSSVM-CIL. The derivatives of the IDLSSVM-CIL Lagrangian  $L$  with respect to  $w$ ,  $b$ , slack variables  $e_i$ , and Lagrange multipliers  $\alpha$  are equated to zero, yielding a set of linear equations. These equations are then solved using the matrix inverse method. Find the calculations [here](#).
- `predict_idlssvmcil(sv_X, X, y, alpha_, b_, mu_)`: This function predicts labels using the IDLSSVM-CIL model, given support vectors, input data, labels, model parameters, and kernel width.
- Predicts labels for the test data using the trained IDLSSVM-CIL model.
- Measures the computation time for training and prediction processes.

## 9. Results section of the notebook

This code calculates evaluation metrics such as AUC, G-mean, and test accuracy from a confusion matrix, then prints them alongside dataset, sampling technique, model, hyperparameters, and computation time. The last code cell visualizes the non-linear SVM, LSSVM or IDLSSVM-CIL algorithm hyperplane on the test set.

## Results and Discussion

We compared three different SVM variants (SVM, LSSVM and IDLSSVM-CIL) and their multiple combinations with oversampling, undersampling, and combined sampling methods (total of nine) using three sampling strategies (desired IR of resultant training data) - 0.4, 0.6, and 1, across five test sets of the KEEL imbalanced datasets.

Detailed results can be accessed [here](#). Result table can be obtained by running [this](#) script. Following metrics have been used:-

### 1. AUC

In [1] Area Under Curve (AUC) (in %) is calculated as

$$AUC = \left( \left( 1 + \frac{TP}{TP + FN} - \frac{FP}{FP + TN} \right) / 2 \right) \times 100$$

where, TP is true positive, TN is true negative, FN is false negative and FP is false positive.

## 2. G-mean

In [1] G-mean is calculated as

$$G\text{-mean} = \text{square root} \left( \frac{TP}{TP + FP} \times \frac{TP}{TP + FN} \right)$$

## 3. Test Accuracy

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

In results, accuracy on the test set is expressed in %

## 4. Implementation time (training + testing time) (in s)

AUC and G-mean have been used for performance evaluation because they depict the relationship between recall and specificity.

# I. Model Comparison:

Following is the methodology to compare Algorithm-Sampling combinations (models):-

**Step 1:** On the 'Results' sheet, the best performing model metrics (minimum time and maximum AUC, G-mean, and test accuracy) are highlighted (boldened in the sheet) for each of the five datasets. The sampling technique that performs the best with a particular algorithm and for a specific dataset is underlined.

**Step 2:** Six models that have achieved the best AUC for a particular dataset are selected for ranking. If the achieved AUC for a dataset is equal, factors such as scores for other datasets and equal representation of an algorithm in the selected models are considered during selection. The no sampling variant (baseline) of each of the three algorithms is included for ranking regardless of its performance. Therefore, the total count of best performing and baseline models is nine.

**Step 3:** For each dataset, models are ranked according to the AUC scores. Models with equal scores are assigned the same rank.

**Step 4:** Average rank is calculated for each of the nine models across the five datasets.

**Step 5:** Steps 2 to 4 are repeated to determine average ranks based on G-mean and test accuracy.

Below given Figures 6,7 and 8 are bar plots of the average ranks of the nine models based on AUC, G-mean and test accuracy respectively.

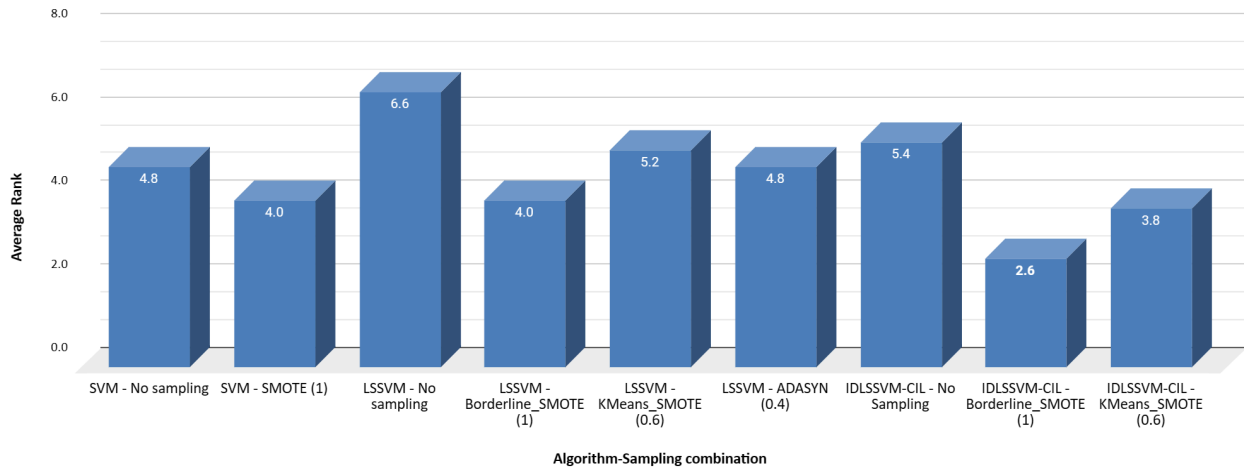


Fig 6: Average rank of the reported algorithm-sampling combinations (with sampling\_strategy) for KEEL artificial imbalanced datasets based on AUC

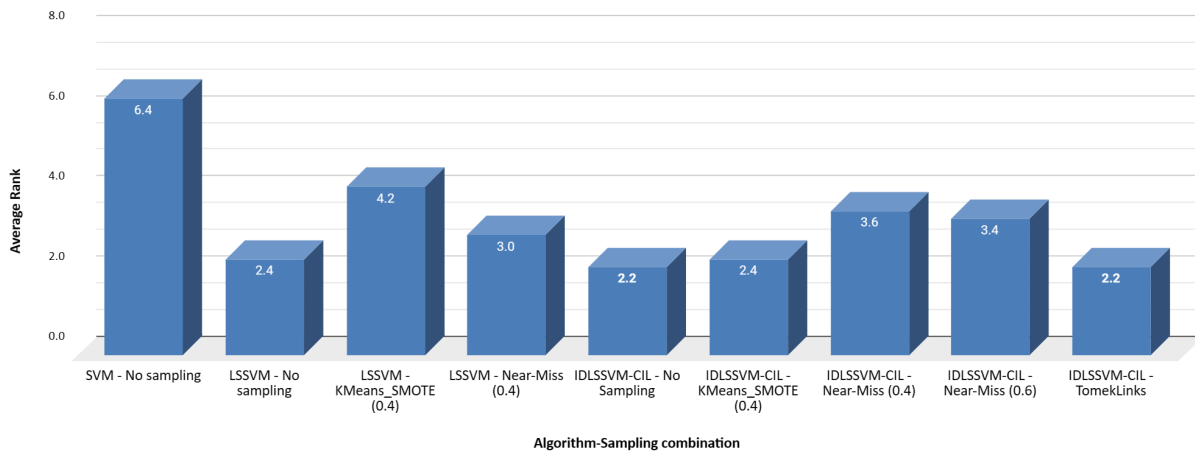


Fig 7: Average rank of the reported algorithm-sampling combinations (with sampling\_strategy) for KEEL artificial imbalanced datasets based on G-mean

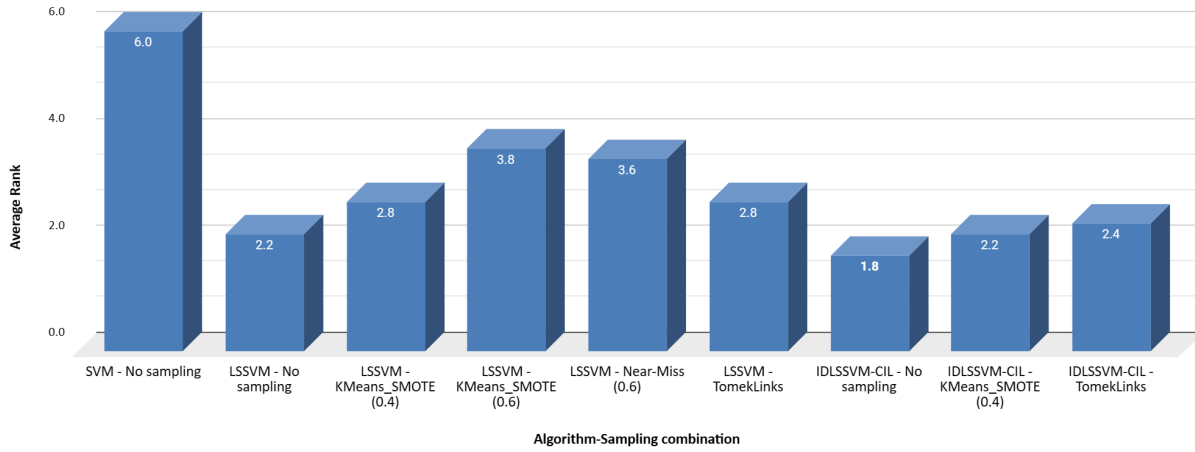


Fig 8: Average rank of the reported algorithm-sampling combinations (with sampling\_strategy) for KEEL artificial imbalanced datasets based on test accuracy

## II. Sampling Technique Comparison:

Four oversampling, three undersampling, and two combined methods are utilized. Each method, except for two undersampling methods, CondensedNN and TomekLinks, experiments with three sampling strategies - 0.4, 0.6, and 1.0. Therefore, there are a total of twenty-four models (including models with no sampling). Below is the methodology for comparing the nine sampling methods:

**Step 1:** In the 'Sampling comparison' sheet, for each of the five datasets, the percentage change in AUC after applying a sampling method, with a specific sampling strategy ratio, is calculated in a separate column.

$$\%change = \frac{AUC_{sampling} - AUC_{baseline}}{AUC_{baseline}} \times 100$$

**Step 2:** To calculate percentage change in AUC for a sampling method irrespective of the sampling strategy used, average of the '%change' is calculated (refer [this](#))

**Step 3:** To consider the percentage change in AUC for all datasets, the average (across five datasets) of the first average '%change', which we can call the 'second average of %change', is calculated.

**Step 4:** To account for the effects of sampling methods on %change in AUC for different models, the final third average is calculated for each method.

**Step 5:** Steps 1 to 4 are repeated to determine the third avg of %change in G-mean and test accuracy.

Table 1 tabulates the third averages of the %change in metrics for each of the nine sampling techniques averaged over the sampling strategy ratios, datasets and models

Sampling Technique	third avg of %change in AUC (all datasets, all sampling_strategies, all models)	third avg of %change in G-mean (all datasets, all sampling_strategies, all models)	third avg of %change in test_acc (all datasets, all sampling_strategies, all models)
SMOTE	<b>6.2829</b>	-0.8654	-1.1082
Borderline_SMOTE	5.8030	-1.9548	-2.7916
KMeans_SMOTE	2.9365	-0.4220	-0.5343
ADASYN	6.1696	-2.0580	-2.9333
Near-Miss	<u>-7.3099</u>	<u>-4.2597</u>	<u>-4.8338</u>
Condensed_NN	-0.8706	<b>0.4383</b>	<b>2.3549</b>
TomekLinks	0.8226	-0.3033	-0.3567
SMOTEENN	3.1594	-1.7045	-2.5891
SMOTETomek	5.6961	-0.8396	-1.1046

Table 1: Average %change in AUC, G-mean and test accuracy for different sampling techniques. Best performing methods are highlighted and worst performing are underlined.

From the above results, following key observations can be noted:-

1. Variants of **IDLSSVM-CIL**, proposed in [1], achieves the **best average rank** among all of the models for all the three performance measures.
2. Best average rank (**2.6**) based on **AUC** is achieved by **IDLSSVM-CIL** trained on a training set oversampled by **Borderline\_SMOTE** with **sampling strategy 1.0**.
3. Best average rank (**2.2**) based on **G-mean** is achieved by **IDLSSVM-CIL** trained on a training set with either **no sampling** or oversampled by **TomekLinks**.
4. Best average rank (**1.8**) based on **test accuracy** is achieved by **IDLSSVM-CIL** trained on a training set with **no sampling**.
5. Oversampling using **SMOTE increases AUC** score the most (by **6.29%**). **Near-Miss** method **worsens** the AUC score the most (by **7.31%**).
6. **CondensedNN** undersampling leads to the highest **increase in G-mean**, however with a nominal gain of **0.44%**. Conversely, the **Near-Miss** method results in the most significant **decrease in G-mean**, with a decline of **4.26%**.
7. **CondensedNN** undersampling demonstrates the largest **increase in test accuracy**, albeit with a modest gain of **2.35%**. In contrast, the **Near-Miss** method again exhibits the most pronounced **decrease in G-mean**, with a decline of **4.83%**.
8. Considering all datasets and models, **oversampling** methods prove more effective than undersampling methods in **enhancing the AUC score**. Conversely, to enhance

the **G-mean and test accuracy**, the **undersampling method CondensedNN** emerges as the sole consistently beneficial approach.

9. **Sampling** methods exert a **notable effect on the AUC** score, ranging from -7.31% to 6.29% on average. However, their impact on **G-mean** and **test accuracy** is relatively **modest**, varying from -4.83% to +2.36% on average.
10. At first glance, a baseline **SVM** trained on a training set undersampled using **Near-Miss** appears to be trained and implemented (tested) the **fastest**. Following this, algorithms combined with CondensedNN undersampling show similar efficiency consistently for all the five datasets. (see 'Results' sheet [here](#))
11. After comparing the results obtained by the **baseline (no sampling) IDLSSVM-CIL** with those published by the authors of [1] (on page 4251, Table-2, and Table-3), it was observed that the test AUC scores were **1-10% higher than the reported scores**. This inconsistency may be attributed to several factors:
  - I. Discrepancies in the test sets obtained after splitting the complete KEEL datasets.
  - II. Variances in the implementation of IDLSSVM-CIL (as discussed in the Methodology section).
  - III. Differences in the selection of the regularization term ( $\epsilon$ ).

Further investigation is required.

Below given Figure 9a and Figure 9b shows the effect of an oversampling technique KMeans\_SMOTE (1) on the performance of IDLSSVM-CIL on the '04clover5z-600-5-70-BI.dat' test set. The AUC score drastically increases after training on oversampled dataset of IR equal to 1.0. More such figures can be generated using [this](#).

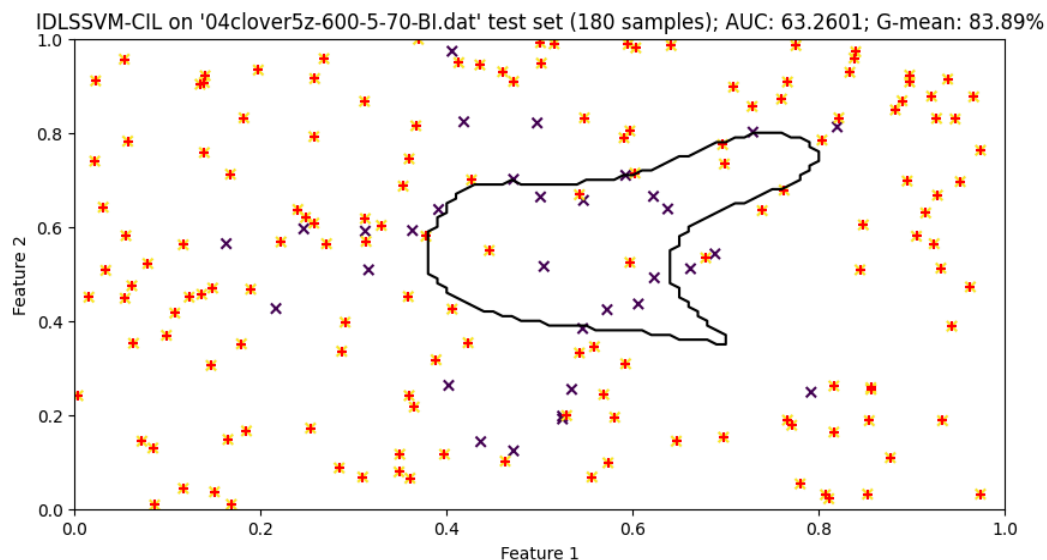


Fig 9a): IDLSSVM-CIL with no sampling



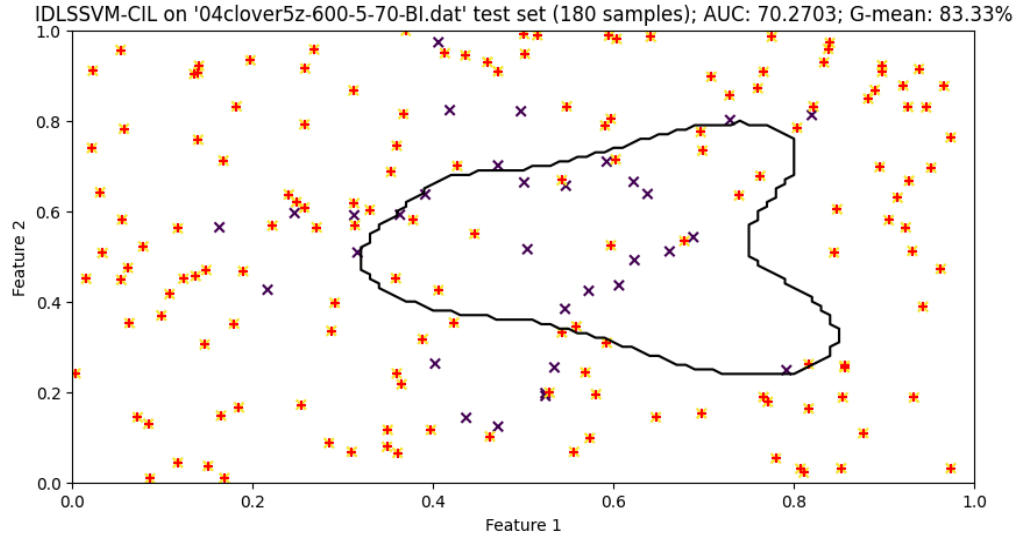


Fig 9b): IDLSSVM-CIL with KMeans\_SMOTE (1)

## Conclusion

In this work, the **SVM** and **LSSVM** models are improved with the density-weighted notion and two new approaches are proposed as **DSVM-CIL** and **IDLSSVM-CIL**. In the first model, the density weight is directly assigned to the slack variables, while in the second model the diagonal weighted matrix is multiplied to the 2-norm of slack variables. Here, the training data-points get some weights during the training phase based on their importance. A series of experiments were conducted to compare the performance of the proposed models with existing and related ones, such as SVM and LSSVM. Additionally, the influence of specific sampling methods was combined with these models for a comprehensive assessment.

Following conclusions can be drawn from the study:-

1. The proposed IDLSSVM-CIL significantly enhances classification performance in terms of AUC and G-mean for imbalanced datasets, concurrently reducing the computational complexity compared to DSVM-CIL.
2. IDLSSVM-CIL, akin to SVM and LSSVM, demonstrates robust performance when combined with oversampling and undersampling techniques.

## Future Scope

The entire process was conducted meticulously, involving the implementation of each model from scratch. This served as a valuable learning experience, as compiling the results from multiple experiments provided us with important insights, as outlined above.

Unfortunately, implementing DSVM-CIL fell outside the scope of the project's allotted time. The DSVM-CIL loss function could potentially be addressed by deriving a set of equations using the Karush–Kuhn–Tucker (KKT) conditions or by iteratively optimizing the values of  $w$  and  $b$  using Gradient Descent. Therefore, our aim is to tackle this implementation in the future.


Additionally, we plan to experiment with model hyperparameters such as  $C$  and  $\mu$ , exploring various combinations of algorithms and sampling techniques rather than relying solely on prescribed values for the baseline models. We intend to apply these combined approaches to multi-variate real-world UCI datasets. Furthermore, thorough hyperparameter tuning of the sampling techniques used is also within our future scope. We also aspire to investigate the effectiveness of other novel sampling methods. Moreover, we aim to explore additional SVM algorithms such as Fuzzy SVM (FSVM) and its variants for comparison.

Looking ahead, we also seek to implement the paper on [Density Weighted Twin Support Vector Machines for Binary Class Imbalance Learning](#), which builds upon the concept of binary class imbalance learning.

Please find the code implementation in [this](#) github repository.

## References

- [1] Hazarika, B.B., Gupta, D. Density-weighted support vector machines for binary class imbalance learning. *Neural Comput & Applic* 33, 4243–4261 (2021). <https://doi.org/10.1007/s00521-020-05240-8>
- [2] 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.
- [3] Haibo He, Yang Bai, E. A. Garcia and Shutao Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, Hong Kong, 2008, pp. 1322-1328, doi: 10.1109/IJCNN.2008.4633969.
- [4] Akira Tanimoto, So Yamada, Takashi Takenouchi, Masashi Sugiyama, Hisashi Kashima, Improving imbalanced classification using near-miss instances, *Expert Systems with Applications*, Volume 201, 2022, 117130, ISSN 0957-4174, <https://doi.org/10.1016/j.eswa.2022.117130>.

- 
- [5] Myungrae Cha, Jun Seok Kim, Jun-Geol Baek, Density weighted support vector data description, *Expert Systems with Applications*, Volume 41, Issue 7, 2014, Pages 3343-3350, ISSN 0957-4174, <https://doi.org/10.1016/j.eswa.2013.11.025>.
- [6] Xinmin Tao, Qing Li, Chao Ren, Wenjie Guo, Qing He, Rui Liu, Junrong Zou, Affinity and class probability-based fuzzy support vector machine for imbalanced data sets, *Neural Networks*, Volume 122, 2020, Pages 289-307, ISSN 0893-6080, <https://doi.org/10.1016/j.neunet.2019.10.016>.
- [7] Han, Hui, Wen-Yuan Wang, and Bing-Huan Mao. "Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning." *International conference on intelligent computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.
- [8] Douzas, Georgios, Fernando Bacao, and Felix Last. "Improving imbalanced learning through a heuristic oversampling method based on k-means and SMOTE." *Information sciences* 465 (2018): 1-20.
- [9] He, Haibo, et al. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning." *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. Ieee, 2008.