EE583 - Pattern Recognition Term Project
Self-Organizing Map Oversampling (SOMO) for Imbalanced Dataset Learning

Okyanus Oral 2305134

Due Date: 09.02.2022

Contents

1 Introduction

Imbalanced class distributions come-up in many real world applications and fields such as medical diagnosis from limited data, identification of rare particles in high energy physics, detection of fraud transactions in finance. In such settings due to the inherit importance of detection of minority class instances, development of a successful classifier is directly dependent on the class imbalance. On such applications, not only the cost of misclassifying minority samples are extremely high, but also learning from an imbalanced data set is a challenging task for standard algorithms since most of the algorithms are designed to work with balanced class distributions. There are several approaches to handle the class imbalance problem. In general, class imbalance is alleviated either by undersampling the majority class or oversampling the minority class. If data is not abundant, generation of synthetic minority samples is more favourable over undersampling [1].

In this project, Self-Organizing Map Oversampling (SOMO) presented in [1] is experimented with different data sets. The results of the SOMO are compared with off the shelf over sampling algorithms such as random sampling, synthetic minority oversampling (SMOTE), borderline SMOTE and adaptive oversampling (ADASYN).

The proceeding chapters explain the oversampling algorithms and SOMO in detail, present the simulation results, give comparisons and conclusions.

1.1 Remarks

All of the relevant codes can be found in Appendix. I have written SOMO oversampler class (which was not available as an off the shelf algorithm) in Python and utilized *imblearn.over_sampling* package for the other oversampling algorithms mentioned in this report.

The YouTube link for the presentation can also be found in Appendix.

2 Theoretical Overview

The class with the most number of samples is defined as the majority class and the rest is called minorty classes. If the number of samples in majority class overly exceeds that of minority class, then the dataset is called imbalanced. In accordance, imbalance is quantitatively defined as the ratio of majority class samples to the minority class samples [1].

Machine learning models that are designed with the assumption that data is balanced, perform poorly with imbalanced datasets. In machine learning community, there are three approaches to improve performance on imbalanced datasets. The first one is creating and utilising machine learning algorithms that put emphasis on minority class data, the second method is applying cost sensitive learning schemes and the third approach is under-oversampling the data in order to balance the datatset [2].

The first two methods try to alleviate the problem on algorithmic level and therefore they are in general not applicable to wide variety of pattern recognition problems where one might have limited alternatives of classifiers due to nature of the task at hand. On the contrary under-oversampling is independent of the deployed system and therefore constitutes a more general approach to the class imbalance problem [1].

Undersampling is removing majority class instances from the dataset, and oversampling is generating artificial/syntehtic samples and ading the generated samples to the dataset. Since undersampling excludes data/information, it can negatively affect classifier performance especially when the dataset is small [3].

The focus of this work is relative effectiveness of oversampling algorithms on imbalanced data. In the following subsections used oversampling algorithms are explained.

2.1 Random Oversampling: In random oversampling minority class instances are selected at random and duplicated with replacement [4]. Due to identical replication of data samples, see Fig. 1, random oversampling may cause overfitting [5].

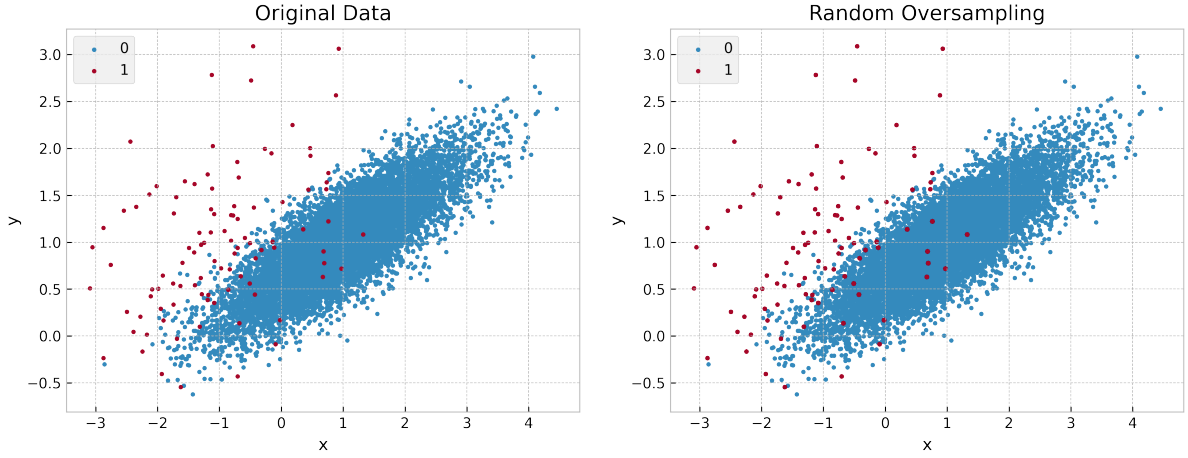


Figure 1: Random Oversampling for the example dataset; majority class label: 0, minority class label: 1.

2.2 Synthetic Minority Oversampling - SMOTE: In SMOTE, new minority samples are synthetically generated and added to the dataset until the desired imbalance ratio is reached. Overview of the algorithm is given in Alg.1.

Algorithm 1: Overview of SMOTE

while *Imbalance ratio is not satisfactory* **do**
  $x_1 \leftarrow$ Randomly select a minority class instance;
  $X_{kNN} \leftarrow$ Find the k nearest minority class neighbours of, x_1;
  $x_2 \leftarrow$ Radomly select an instance from X_{kNN};
  $x_{new} = (\alpha) \cdot x_1 + (1 - \alpha) \cdot x_2$, $\alpha \sim Unif(0, 1)$ generate a synthetic data using x_1 and x_2 as a random convex combination;
end
return Oversampled Dataset

In Fig. 2 the original and oversampled dataset with SMOTE are shown. A possible drawback of using SMOTE is generation of noise samples. Noise samples are the data instances that do not necessarily fit the distribution of its respective class. If, a noise instance is used to generate a new sample, or the line segment that connects the selected samples pass through the majority class instances then SMOTE might generate noisy instances, which might deprecate the classifier's performance.

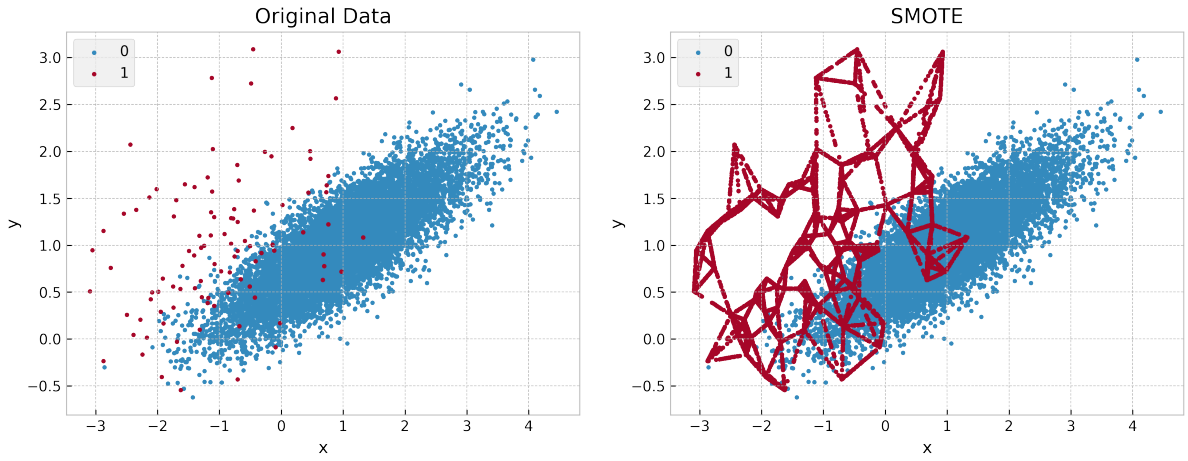


Figure 2: SMOTE for the example dataset; majority class label: 0, minority class label: 1.

2.3 Borderline SMOTE: Data near to the class boundaries are more apt to be misclassified than the ones far away, thus they are more important. Borderline SMOTE applies SMOTE to minority class instances near the borderline. Therefore, the algorithm also counts for the relative distribution of classes [6]. Overview of Borderline SMOTE is given in Alg.2.

Algorithm 2: Overview of Borderline SMOTE

while *Imbalance ratio is not satisfactory* **do**
  $x_1 \leftarrow$ Randomly select a minority class instance;
  $X_{kNN} \leftarrow$ Find the k nearest neighbours of, x_1;
  $N_{min.}, N_{maj.} \leftarrow$ Count the number of minority and majority class instances in X_{kNN};
  if $N_{min.} = 0$ **then**
    x_1 is considered as a noise instance. It is encircled by majority class instances, go to line 2;
  else if $N_{min.} > N_{maj.}$ **then**
    x_1 is not near of the separation border. Its neighbours are mostly minority samples, go to line 2;
  else
    $x_2 \leftarrow$ Randomly select a minority class instance from X_{kNN};
    $x_{new} = (\alpha) \cdot x_1 + (1 - \alpha) \cdot x_2,\ \alpha \sim Unif(0, 1)$ generate a synthetic data using x_1 and x_2 as a random convex combination;
  end
end
return Oversampled Dataset

Original and oversampled dataset with Borderline SMOTE are shown in Fig. 3.

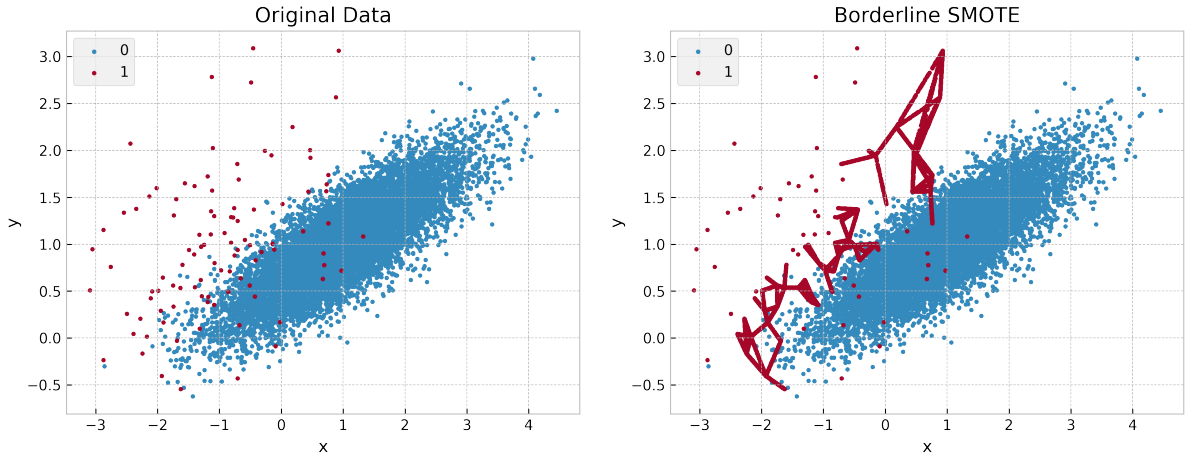


Figure 3: Borderline SMOTE for the example dataset; majority class label: 0, minority class label: 1.

2.4 Adaptive Oversampling - ADASYN Although Borderline SMOTE takes into consideration of class distributions via detection of noise and borderline samples, it does not take into account of local imbalance ratio of minority samples in the feature space. That is with Borderline SMOTE or SMOTE, number of generated samples for a border instance does not vary with local imbalance ratio. ADASYN calculates the imbalance ratio of each minority sample in the dataset and generates synthetic samples proportional to local imbalance ratios [7]. Overview of the ADASYN is given in Alg.3.

Algorithm 3: Overview of ADASYN

$N_{syn.} \leftarrow$ Calculate the number of synthetic samples needed to satisfy the desired imbalance ratio.
foreach *minority sample,* x_i **do**
  $X_{kNN} \leftarrow$ Find the k nearest neighbours of, x_i;
  $N_{min.}, N_{maj.} \leftarrow$ Count the number of minority and majority class instances in X_{kNN};
  $r_i = \frac{N_{maj.}}{N_{min.}+N_{maj.}}$ Compute the ratio of majority samples encircling x_i;
end
foreach *minority sample,* x_i **do**
  $N_{syn.\,i} = N_{syn.} \cdot \frac{r_i}{\sum_j r_j}$ Compute the proportional number of synthetic samples to be generated for x_i;
  while *the number of generated synthetic samples did not reach* $N_{syn.\,i}$ **do**
    $X_{kNN} \leftarrow$ Find the k nearest neighbours of, x_i;
    $x_2 \leftarrow$ Randomly select a minority class instance from X_{kNN};
    $x_{new} = (\alpha) \cdot x_i + (1-\alpha) \cdot x_2,\ \alpha \sim Unif(0,1)$ generate a synthetic data using x_1 and x_2 as a random convex combination;
  end
end
return Oversampled Dataset

Original and oversampled datasets with ADASYN are shown in Fig. 3. Although, emphasis is again put on the borderline, with ADASYN synthetic samples are also generated in the regions where they can be considered as noise.

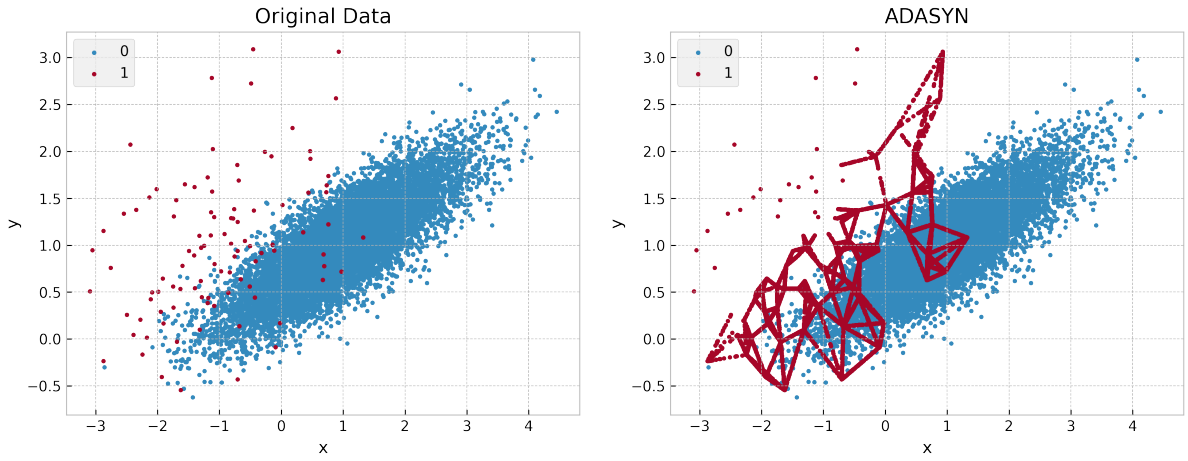


Figure 4: ADASYN for the example dataset; majority class label: 0, minority class label: 1.

2.4 Self Organising Map Oversampling - SOMO In [1] it is stated that solely relying on euclidean distance to generate synthetic samples makes the algorithms prone to noisy instances and interwoven class distributions. It is also stated that although there are heuristic oversampling methods that assume a certain manifold structure, these algorithms are not applicable or valid to many situations. Authors of SOMO try to utilize self organising maps in order to capture the topologically neighbouring minority class instances and to generate data without noise [1]. In SOMO, synthetic samples are generated using the clusters obtained via a self organising map (SOM). Also, in order to avoid using noise clusters, SOMO takes each cluster's imbalance ratio into account and filters clusters accordingly. Generated synthetic samples are convex combinations of minority samples from the filtered intra-clusters and topologically neighbouring filtered inter-clusters [1]. Overview of the SOMO algorithm is given in Alg. 4.

Algorithm 4: Overview of SOMO

$N_{itra} \leftarrow$ Number of intra-cluster synthetic samples to generate;
$N_{inter} \leftarrow$ Number of inter-cluster synthetic samples to generate;
$SOM \leftarrow$ Cluster all samples using a self organising map;
foreach *cluster* c_i *in* SOM **do**
  $N_{min.}, N_{maj.} \leftarrow$ Count the number of minority and majority class instances in c_i;
  $r_i = \frac{N_{maj.}+1}{N_{min.}+1}$ Calculate within cluster imbalance ratio;
  if $r_i > 1$ **then**
    Cluster is dominated by majority samples, generating synthetic data using this cluster may produce noise instances, go to line 2;
  else
    $c_i^* \leftarrow$ Mark the cluster, c_i, as a minority cluster;
    $d_i \leftarrow$ Calculate sum of euclidean distances between within cluster minority samples;
    $\rho_i = \frac{N_{min}}{d_i^2}$ Calculate intra-cluster density;
  end
end
$(C_i^* \times C_j^*) \leftarrow$Find topologically neighbouring minority clusters in SOM.
foreach *topologically neighbouring minority cluster pair* $(c_i^* \times c_j^*)$ *in* $(C_i^* \times C_j^*)$ **do**
  $\rho_{i,j} = \rho_i + \rho_j$ Calculate inter-cluster density;
end
foreach *minority cluster* c_i^* **do**
  $X_{intra,new} \leftarrow$ Generate $N_{intra} \cdot \frac{\rho_i}{\sum_j \rho_j}$ number of synthetic minority intra-cluster samples using SMOTE.
end
foreach *topologically neighbouring minority cluster pair* $(c_i^* \times c_j^*)$ *in* $(C_i^* \times C_j^*)$ **do**
  $N_{inter:i,j} = N_{inter} \cdot \frac{\rho_{i,j}}{\sum_{i,j} rho_{i,j}}$ Calculate the number of synthetic inter-cluster samples to be generated;
  $c_1^* \leftarrow$ find the the minority cluster with higher number of minority samples from c_i^* and c_j^*;
  $c_2^* \leftarrow$ find the the minority cluster with fewer number of minority samples from c_i^* and c_j^*;
  while *Total number of synthetic inter-cluster samples did not reach* $N_{inter:i,j}$ **do**
    $x_1 \leftarrow$ Randomly select a minority class instance from c_1^*
    $X_{kNN} \leftarrow$ Find the k nearest minority class neighbours of x_1 in c_2^*;
    $x_2 \leftarrow$ Randomly select an instance from X_{kNN};
    $x_{inter,new} = (\alpha) \cdot x_1 + (1-\alpha) \cdot x_2,\ \alpha \sim Unif(0,1)$ generate a synthetic data using x_1 and x_2 as a random convex combination;
    $X_{inter,new} \leftarrow$ Generate synthetic minority inter-cluster samples as a convex combination;
  end
end
return Oversampled Dataset

Original dataset and the output of SOMO (with a 20×20 SOM) is shown in Fig. 5.

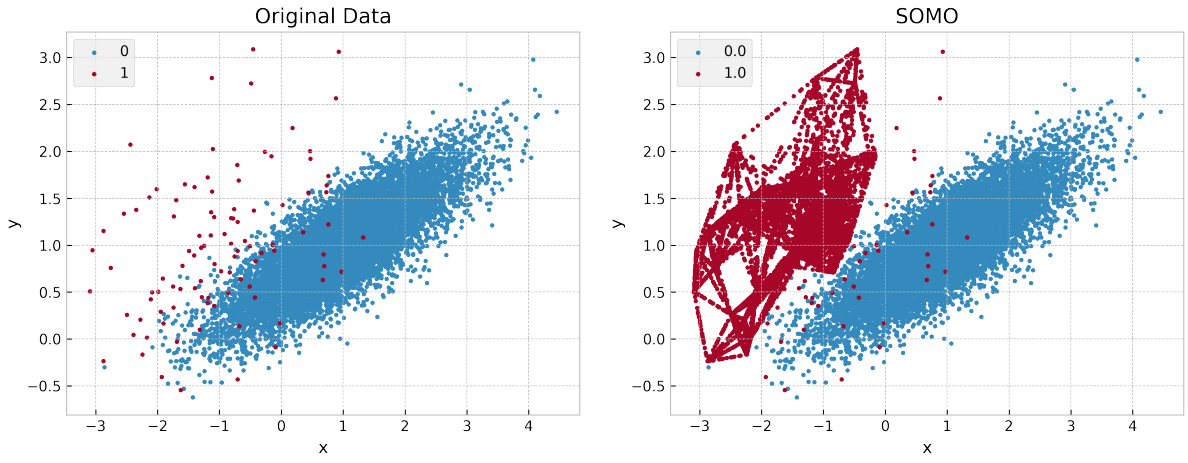
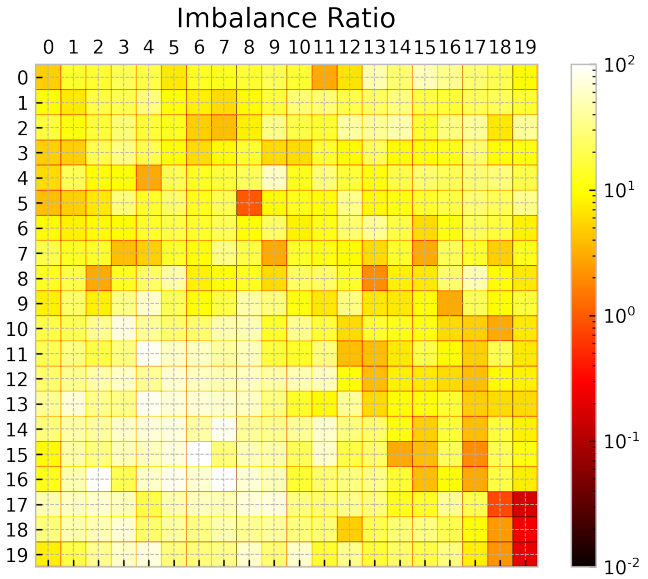


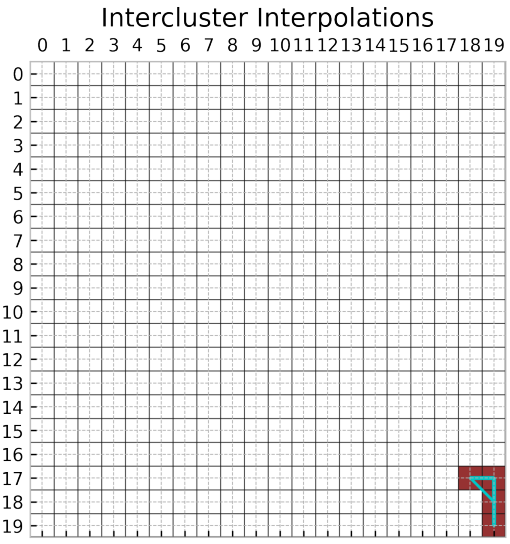
Figure 5: SOMO for the example dataset; majority class label: 0, minority class label: 1.

In order to understand the data generation process it is also important to investigate the distribution of

data inside the self organising map. For the results in Fig. 5 the corresponding SOM, filtered clusters and interpolations are shown in Fig. 6.



(a) Within cluster imbalance ratio.



(b) Filtered clusters (brown) and interpolations (blue).

Figure 6: SOMO for the example dataset, SOM grid: 20×20, resulting in 4 incidence clusters.

As it will be discussed later in detail, grid size of SOM is an important parameter for SOMO and it significantly effects the algorithm's performance.

3 Simulation

Not every classification metric is appropriate for imbalanced datasets. For example, a classifier that only classifies data as majority class would have a high accuracy although the classifier did not learn anything significant. Therefore, suitable metrics are needed to be selected. As stated and used in the original paper, oversampling algorithms' performance is assessed based on the following 3 metrics; F1-score, G-score and Area Under the ROC Curve (AUC) [1]. F1 score is the harmonic mean of precision and recall, hence high F1 score puts equal emphasis on correct positive class predictions relative to the total number of positive class predictions and total number of positive class observations. Let, TP stand for correct positive class predictions, FP stand for incorrect positive class predictions, TN stand for correct negative class predictions and FN stand for incorrect negative class predictions. Then,

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

$$Recall = \frac{TP}{TP + FN}$$

$$F1Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Similarly, G-mean is the geometric mean of the True Positive Rate and the True Negative Rate [1], i.e.

$$G_{mean} = \sqrt{\frac{TP}{TP + FN} \cdot \frac{TN}{TN + FP}}$$

Furthermore, AUC is the area under the ROC curve, which is for the worst classification performance equal to 0.5 and for the best classification performance equals to 1.

3.1 Procedure

In experiments 8 separate datasets are used, all of which corresponds to a binary classification problem, see Tab. 1. All datasets are chosen to be different from the original paper. Furthermore, some datasets' imbalance

ratio is further decreased to test for extreme settings as in [1]. Important note: Some of the datasets' sizes are also shrinked. See Appendix for all the datasets.

Table 1: Datasets

Dataset	Imbalance Ratio	Number of Features	Number of Minority Samples	Number of Majority Samples	Minority Class
Digits [8]	8.8196	64	183	1614	3
Frogs_MFCCs [9]	104.8082	22	68	7127	Bufonidae
Parkinsons [9],[10]	3.0625	48	48	147	Healthy
Vertebral Column [9]	4.1666	6	60	250	Hernia
Thyroid [8]	6.1666	5	30	185	3
CTG [9]	11.0965	39	176	1950	Pathologic
Accelerometer [9], [11]	1.8412	4	359	661	1
Wifi_localization [9],[12],[13]	3	7	500	1500	1
CTG-2 [9]	55.7142	39	35	1950	Pathologic
Accelerometer-2 citeDatasets, [11]	9.3098	4	71	661	1
Wifi_localization-2 [9],[12],[13]	15	7	100	1500	1

Oversampling method's performances are tested using two different classifiers, namely, Logistic Regression (LR) [14] and Gradient Boosting Machine (GBM) [15] as in the original paper [1]. The intention of the authors on using two different classification algorithms was to show algorithm independent performance of the oversampling methods [1]. Furthermore LR does not have any hyper-parameters and the *sklearn*'s GBM allows early stopping, which decreases overfitting [1]. As the focus of the project is to demonstrate the oversampling algorithms, these two classifiers are not explained in detail. However logistic regression is covered in lectures and GBM is an ensemble of weak prediction models similar to random forest algorithm [15].

For each dataset, each classifier, each oversampling algorithm and the hyperparameter of the oversampling algorithm; k-fold cross validation is applied with k=5. Before training, at each of the k stages of k-fold cross validation, k-1 folds of the dataset is oversampled using the respective oversampling algoritms until imbalance ratio of the training dataset became 1. Then, the classifier performance is tested with the remaining validation fold, which was not oversampled. Furthermore, during the training of GBM, k-1 training folds is further split into training/validation datasets (again 5 fold) where validation performance of GBM is maximized. In summary, for each hyperparameter trial, 5 runs are made with 5-fold cross validation each. These simulation settings are identical to the original paper [1]. However, in the original paper, hyperparameter search for the oversampling algorithms is unfair. For SMOTE, Borderline SMOTE and ADASYN number of k nearest neighbours is tested only for $k \in \{2, 3, 4, 5\}$ meanwhile grid-size of SOMO is searched starting from $\sqrt{\#\ minority\ samples}$ to $\sqrt{\#\ majority\ samples}$ with steps of 5. This creates an unfair adventage on SOMO. Therefore, deviating from the original paper, in order to make the assessments fair, only 4 values of SOM gridsize is tested, which start from $\sqrt{\#\ minority\ samples}$ up to $\sqrt{\#\ majority\ samples}$ with equal spacing.

3.2 Results and Comments

The performance of classifiers on the oversampled data are presented in Appendix, Tab. 3. Presented performances are for the best hyperparameters of oversampling algorithms for the respective settings. Using the classification performances, the final score of the oversampling algorithms are determined. For each dataset, oversampling algorithms are ranked. Best ranking algorithm is scored 1 and the worst is scored 6. Then the mean of the rankings is taken and given as the performance of the oversampling algorithms. Performances, of oversampling algorithms are summarised in Tab. 2. This is the same ranking procedure described in [1]. Moreover, the datasets with artificially increased imbalance ratio are intentionally excluded from ranking since none of the oversampling algorithms had any success.

Table 2: Mean Rankings of Oversampling Algorithms

Classifier	Metric	None	Random	SMOTE	Borderline SMOTE	ADASYN	SOMO
Logistic Regression	AUC	3.222222	2.555556	2	3	2.666667	3.555556
	F	3.333333	3.555556	2.666667	2.777778	3.222222	2.333333
	G	4.777778	2	2.111111	2.777778	2.111111	3.888889
Gradient Boosting Machine	AUC	4.5	3.5	2.666667	3.5	3.5	2.833333
	F	4.833333	2.333333	2.166667	2.333333	3	3.5
	G	5.166667	3	2.166667	2.666667	2.833333	3.166667

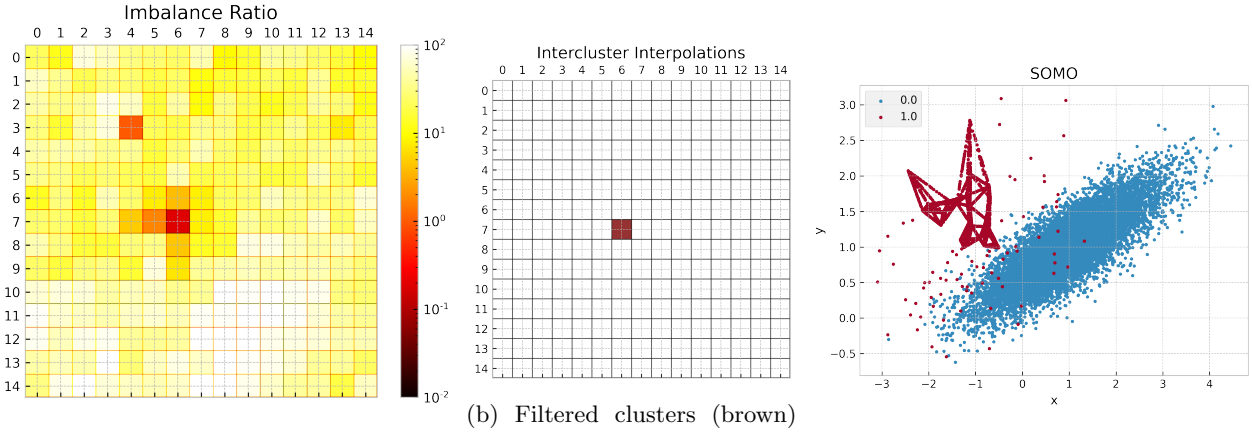
SOMO ranks 1^{st} for F1 score using LR and 2^{nd} for AUC using GBM. For all other settings it ranks worse than all other oversampling algorithms except from random oversampling and None (not sampling the dataset). This contradicts the findings of the original paper, where SOMO was the top ranking oversampling algorithm for all cases.

The results can be explained as follows;

1. In original paper, there was unfair advantage for SOMO where number of parameters searched for SOMO was not constant and was more than other algorithms.
2. Grid size of SOM affects algorithm performance significantly,
3. In the original paper, classifiers' performances were not saturated, however for most of the datasets in Tab.1 classifier results are near perfect without any oversampling. 3.

Nevertheless, it is for certain that other oversampling algorithms perform better for the conducted experiments.

From the experiment results and visual inspection of filtered clusters, see Fig. 7,8,9, the most prevalent reason why SOMO had failed seems to be its over sensitivity to SOM grid size. With increasing grid size clusters become topologically more sparse, this is also stated in [1]. Furthermore, notice that for the same grid size, 20×20, SOMO outputs significantly different oversampled data distributions, see Fig 5 and 8

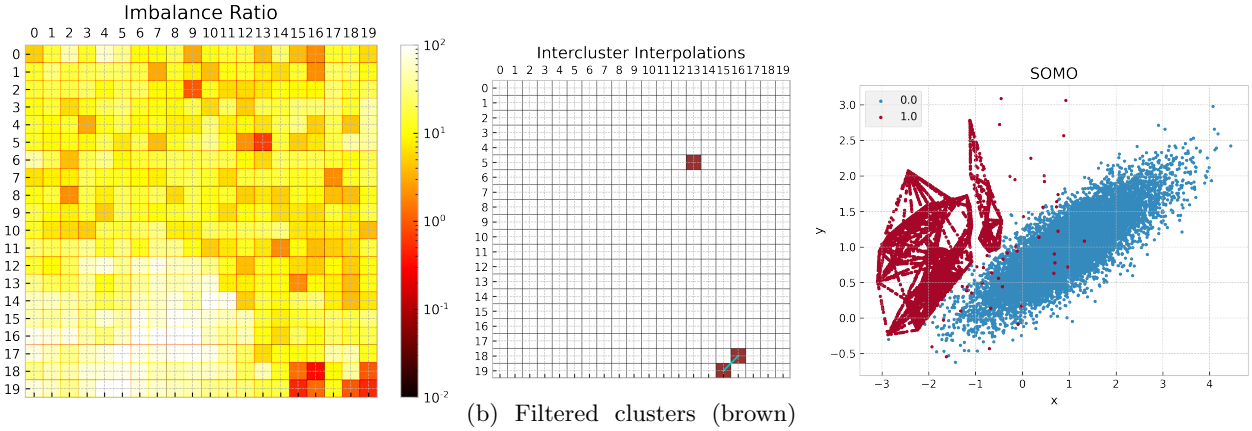


(a) Within cluster imbalance ratio.

(b) Filtered clusters (brown) and interpolations (blue).

(c) Oversampled example dataset.

Figure 7: SOMO for the example dataset, SOM grid: 15×15, resulting in 1 clusters.

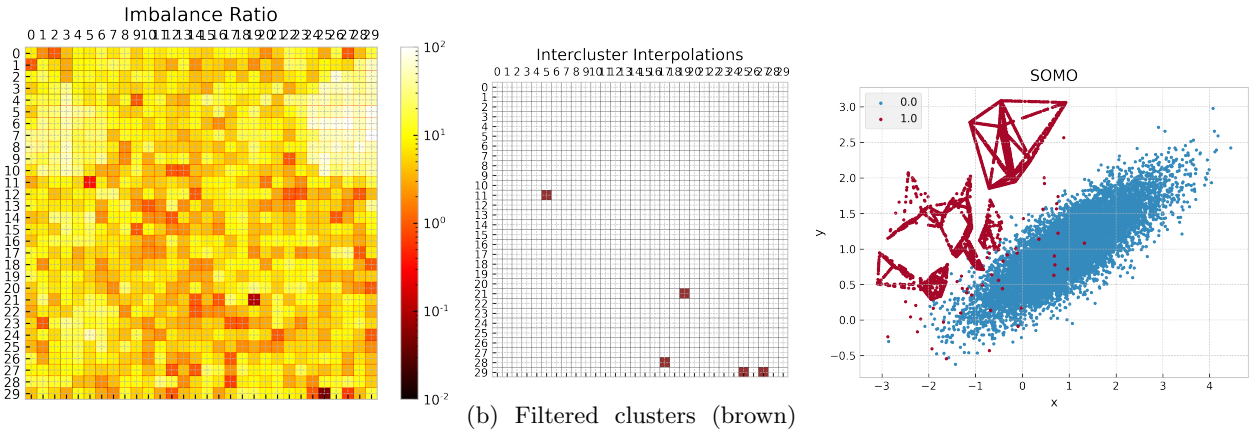


(a) Within cluster imbalance ratio.

(b) Filtered clusters (brown) and interpolations (blue).

(c) Oversampled example dataset.

Figure 8: SOMO for the example dataset, SOM grid: 20×20, resulting in 3 clusters (2 incidence).



(a) Within cluster imbalance ratio.

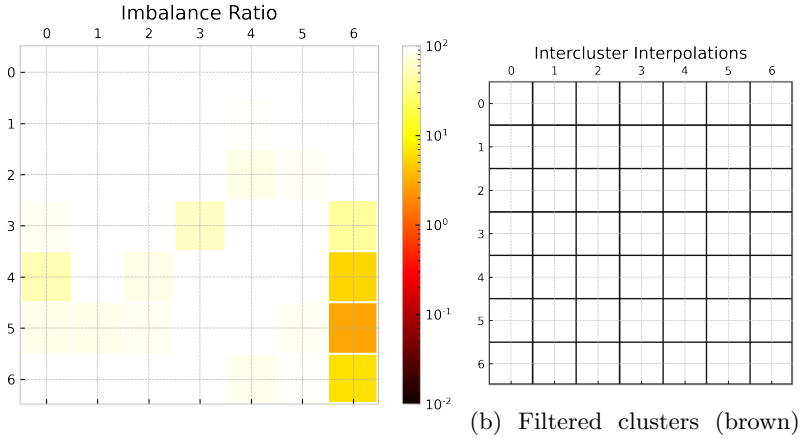
(b) Filtered clusters (brown) and interpolations (blue).

(c) Oversampled example dataset.

Figure 9: SOMO for the example dataset, SOM grid: 30×30, resulting in 5 clusters.

Also notice that SOM grid size must be set in accordance with number of minority and majority samples in the dataset [1]. With increasing number of samples, required grid size increases due to overcrowding of SOM cells with majority class instances. Realize that if there are not any clusters where minority class instances dominate, then data generation is not possible.

Correctly estimating the number of clusters required for successful oversampling make SOMO a cumbersome algorithm. Extensive tuning of its parameters are required for good performance therefore its solutions are not reliable. In Tab.3 for the last 3 datasets, SOMO performs the same with None (not oversampling the dataset). Although SOMO is tested with the suggested upper gridsize [1], the grid size is not satisfactory. This is also illustrated with the example dataset, where with grid size of 7×7, SOMO fails to generate any synthetic samples, see Fig. 10.



(a) Within cluster imbalance ratio.

(b) Filtered clusters (brown) and interpolations (blue).

Figure 10: SOMO for the example dataset, SOM grid: 7×7, resulting in 0 clusters.

4 Conclusion

In this project, class imbalance problem is explained, some of the imbalance learning approaches are discussed and oversampling methods are demonstrated.

Performance of a novel oversampling method SOMO is compared with other off the shelf oversampling algorithms. Caveats of SOMO is explained in detail and the statements about its ineffectiveness are supported with experimental results.

Appendix

Click for codes: https://www.dropbox.com/sh/vnsouf3fpbr7pvb/AAAI6QMmyedp8N4FxI7QT4_qa?dl=0

Click for the online presentation (YouTube link): https://youtu.be/kIhhMNTSI_E

Table 3: Performance of Classifiers

Dataset	Classifier	Metric	None	Random	SMOTE	Borderline SMOTE	ADASYN	SOMO
Accelerometer	Logistic Regression	AUC	0.449951 ± 0.04492	0.463108 ± 0.05398	0.467389 ± 0.05676	0.500431 ± 0.05449	0.479191± 0.05941	0.486530 ± 0.06114
		F	0.044642 ± 0.02595	0.340305 ± 0.06444	0.347256 ± 0.06032	0.401726 ± 0.07242	0.430278 ± 0.10060	0.255555 ± 0.07245
		G	0.146226 ± 0.04180	0.457541 ± 0.05258	0.466805 ± 0.05027	0.496571 ± 0.04105	0.460889 ± 0.03622	0.393545 ± 0.06809
Accelerometer	Gradient Boosting Machine	AUC	0.726787 ± 0.02957	0.737509 ± 0.02435	0.738416 ± 0.01899	0.736898 ± 0.02024	0.732926 ± 0.02180	0.740321 ± 0.01784
		F	0.346358 ± 0.05738	0.565366 ± 0.02632	0.577235 ± 0.03306	0.588118 ± 0.02805	0.592828 ± 0.03289	0.445231 ± 0.04853
		G	0.472220 ± 0.05668	0.657587 ± 0.01991	0.666839 ± 0.02677	0.675905 ± 0.02552	0.671252 ± 0.02565	0.556759 ± 0.04474
CTG	Logistic Regression	AUC	0.983337 ± 0.00378	0.986848 ± 0.00518	0.987053 ± 0.00582	0.984954 ± 0.00552	0.985447 ± 0.00622	0.981378 ± 0.00612
		F	0.803926 ± 0.04593	0.710865 ± 0.07115	0.751411 ± 0.07814	0.728159 ± 0.07250	0.717814 ± 0.07003	0.805171 ± 0.04598
		G	0.873734 ± 0.01625	0.934955 ± 0.01945	0.936562 ± 0.01368	0.927827 ± 0.01851	0.932722 ± 0.02354	0.887146 ± 0.02223
CTG	Gradient Boosting Machine	AUC	0.996572 ± 0.00685	0.996350 ± 0.00729	0.996350 ± 0.00729	0.996350 ± 0.00729	0.996620 ± 0.00675	0.997247 ± 0.00603
		F	0.993750 ± 0.01249	0.990575 ± 0.01252	0.993750 ± 0.01249	0.993750 ± 0.01250	0.993750 ± 0.01249	0.993750 ± 0.01249
		G	0.996590 ± 0.00680	0.996345 ± 0.00669	0.996599 ± 0.00680	0.996599 ± 0.00680	0.996599 ± 0.00680	0.996599 ± 0.00680
Digits	Logistic Regression	AUC	0.995730 ± 0.00323	0.995069 ± 0.00379	0.995159 ± 0.00388	0.994621 ± 0.00350	0.994838 ± 0.00384	0.995233 ± 0.00348
		F	0.899740 ± 0.04700	0.899432 ± 0.01771	0.907943 ± 0.02955	0.886850 ± 0.04048	0.898055 ± 0.02467	0.909061 ± 0.03086
		G	0.927701 ± 0.03956	0.963320 ± 0.01679	0.957150 ± 0.02513	0.950376 ± 0.03127	0.959659 ± 0.02580	0.953572 ± 0.02503
Digits	Gradient Boosting Machine	AUC	0.989320 ± 0.00729	0.997614 ± 0.00161	0.997524 ± 0.00126	0.998090 ± 0.00172	0.998280 ± 0.00155	0.996564 ± 0.00259
		F	0.870540 ± 0.03533	0.938718 ± 0.01487	0.946247 ± 0.01775	0.946865 ± 0.02409	0.955895 ± 0.02632	0.933228 ± 0.02472
		G	0.889200 ± 0.03191	0.962545 ± 0.00898	0.961843 ± 0.01513	0.964230 ± 0.02517	0.970106 ± 0.01985	0.948706 ± 0.02732
Frogs_MFCCs	Logistic Regression	AUC	0.956501 ± 0.00409	0.974592 ± 0.01079	0.974201 ± 0.01187	0.953896 ± 0.01635	0.970129 ± 0.00085	0.928543 ± 0.04752
		F	0.0 ± 0.0	0.211099 ± 0.02892	0.217298 ± 0.02432	0.255083 ± 0.04014	0.253927 ± 0.00617	0.352202 ± 0.04611
		G	0.0 ± 0.0	0.939722 ± 0.03402	0.935753 ± 0.03189	0.886852 ± 0.09138	0.962955 ± 0.01544	0.904780 ± 0.04518
parkinsons	Logistic Regression	AUC	0.890056 ± 0.04378	0.903516 ± 0.03973	0.902950 ± 0.03912	0.900976 ± 0.04262	0.904139 ± 0.04417	0.874008 ± 0.05142
		F	0.639653 ± 0.11388	0.619034 ± 0.11532	0.629224 ± 0.12312	0.653860 ± 0.14249	0.648946 ± 0.13430	0.617552 ± 0.13431
		G	0.727705 ± 0.10533	0.783064 ± 0.05830	0.791688 ± 0.06530	0.815053 ± 0.05432	0.813743 ± 0.05270	0.742299 ± 0.09260
Parkinsons	Gradient Boosting Machine	AUC	0.943098 ± 0.04014	0.957955 ± 0.04915	0.959634 ± 0.03955	0.955551 ± 0.05143	0.952541 ± 0.04703	0.951816 ± 0.04126
		F	0.759942 ± 0.09967	0.817603 ± 0.13934	0.794729 ± 0.14537	0.801814 ± 0.15214	0.782504 ± 0.16151	0.793552 ± 0.13214
		G	0.827930 ± 0.08371	0.882973 ± 0.07976	0.885666 ± 0.05715	0.899546 ± 0.07722	0.888280 ± 0.06519	0.855127 ± 0.08443
Thyroid	Logistic Regression	AUC	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
		F	0.951871 ± 0.04100	0.954901 ± 0.06491	0.954901 ± 0.064913	0.942501 ± 0.07510	0.954901 ± 0.06491	0.962780 ± 0.04025
		G	0.962270 ± 0.04067	0.991787 ± 0.01078	0.991787 ± 0.010781	0.983728 ± 0.02447	0.991787 ± 0.01078	0.975681 ± 0.03576
Thyroid	Gradient Boosting Machine	AUC	0.956209 ± 0.04665	0.988665 ± 0.03865	0.993155 ± 0.009918	0.994413 ± 0.00709	0.995116 ± 0.00700	0.996030 ± 0.00709
		F	0.888484 ± 0.07814	0.926849 ± 0.12237	0.926883 ± 0.078202	0.913790 ± 0.09619	0.907522 ± 0.07510	0.924584 ± 0.08643
		G	0.922689 ± 0.04348	0.949639 ± 0.08677	0.953003 ± 0.051384	0.942366 ± 0.07758	0.941537 ± 0.05952	0.958199 ± 0.05823
Vertebral Column	Logistic Regression	AUC	0.924653 ± 0.04262	0.926909 ± 0.04044	0.927243 ± 0.040028	0.927753 ± 0.03830	0.928867 ± 0.03786	0.924606 ± 0.04341
		F	0.639365 ± 0.12968	0.675242 ± 0.17216	0.676503 ± 0.167804	0.677497 ± 0.17717	0.668935 ± 0.17180	0.677069 ± 0.14175
		G	0.759409 ± 0.08708	0.847443 ± 0.08177	0.843037 ± 0.081273	0.854643 ± 0.07269	0.847972 ± 0.05131	0.821259 ± 0.07648
Wifi localization	Logistic Regression	AUC	0.999729 ± 0.00018	0.999691 ± 0.00024	0.999722 ± 0.000212	0.999445 ± 0.00051	0.999403 ± 0.00048	0.999701 ± 0.00021
		F	0.987768 ± 0.00832	0.987018 ± 0.00865	0.987003 ± 0.007750	0.976978 ± 0.01290	0.972169 ± 0.01404	0.986239 ± 0.00822
		G	0.991893 ± 0.00634	0.993546 ± 0.00428	0.992729 ± 0.004537	0.989094 ± 0.00468	0.989185 ± 0.00521	0.992663 ± 0.00537
Wifi localization	Gradient Boosting Machine	AUC	0.999319 ± 0.00114	0.999080 ± 0.00129	0.999509 ± 0.000860	0.999033 ± 0.00175	0.996847 ± 0.00402	0.999295 ± 0.00113
		F	0.986996 ± 0.00864	0.990556 ± 0.00748	0.990942 ± 0.007036	0.987926 ± 0.01148	0.985866 ± 0.01057	0.989503 ± 0.00737
		G	0.991556 ± 0.00546	0.993762 ± 0.00723	0.994166 ± 0.007050	0.992506 ± 0.00655	0.991245 ± 0.00668	0.993149 ± 0.00727
Accelerometer2	Logistic Regression	AUC	0.458661 ± 0.05640	0.450373 ± 0.07083	0.456053 ± 0.058124	0.467059 ± 0.07029	0.474751 ± 0.05149	0.458661 ± 0.05640
		F	0.0 ± 0.0	0.126902 ± 0.05349	0.133211 ± 0.050586	0.137233 ± 0.04017	0.140836 ± 0.04250	0.0 ± 0.0
		G	0.0 ± 0.0	0.421641 ± 0.09397	0.438032 ± 0.086724	0.456129 ± 0.08048	0.442915 ± 0.06297	0.0 ± 0.0
Accelerometer2	Gradient Boosting Machine	AUC	0.571757 ± 0.05057	0.524560 ± 0.06522	0.537906 ± 0.062333	0.521945 ± 0.04507	0.523765 ± 0.06034	0.571757 ± 0.05057
		F	0.0 ± 0.0	0.098548 ± 0.06647	0.154261 ± 0.044769	0.127239 ± 0.04340	0.153938 ± 0.05971	0.0 ± 0.0
		G	0.0 ± 0.0	0.322236 ± 0.16332	0.465601 ± 0.088968	0.413827 ± 0.10949	0.460542 ± 0.08392	0.0 ± 0.0
CTG2	Logistic Regression	AUC	0.598239 ± 0.07731	0.596012 ± 0.07678	0.608016 ± 0.059281	0.589342 ± 0.04327	0.605144 ± 0.06533	0.598239 ± 0.07731
		F	0.0 ± 0.0	0.042449 ± 0.00850	0.044492 ± 0.007880	0.042968 ± 0.00560	0.045561 ± 0.01021	0.0 ± 0.0
		G	0.0 ± 0.0	0.545867 ± 0.06676	0.550705 ± 0.068540	0.537601 ± 0.06838	0.555811 ± 0.05989	0.0 ± 0.0
CTG2	Gradient Boosting Machine	AUC	0.466485 ± 0.04531	0.441507 ± 0.08752	0.484450 ± 0.087435	0.495207 ± 0.07660	0.477374 ± 0.08133	0.469085 ± 0.04245
		F	0.0 ± 0.0	0.0 ± 0.0	0.005714 ± 0.027994	0.0 ± 0.0	0.012820 ± 0.04351	0.0 ± 0.0
		G	0.0 ± 0.0	0.0 ± 0.0	0.014050 ± 0.068835	0.0 ± 0.0	0.028156 ± 0.09548	0.0 ± 0.0
Wifi localization2	Logistic Regression	AUC	0.506887 ± 0.03081	0.505111 ± 0.03043	0.516174 ± 0.047739	0.524676 ± 0.06070	0.513999 ± 0.04503	0.508260 ± 0.03223
		F	0.0 ± 0.0	0.115004 ± 0.03530	0.116925 ± 0.035856	0.117018 ± 0.04494	0.119779 ± 0.03353	0.006708 ± 0.02508
		G	0.0 ± 0.0	0.513340 ± 0.03024	0.518334 ± 0.035369	0.504640 ± 0.05410	0.525240 ± 0.02706	0.024393 ± 0.08273
Wifi localization2	Gradient Boosting Machine	AUC	0.524054 ± 0.08391	0.445050 ± 0.03510	0.500497 ± 0.041231	0.500387 ± 0.06118	0.503426 ± 0.03768	0.524054 ± 0.08391
		F	0.0 ± 0.0	0.076875 ± 0.04112	0.056160 ± 0.047734	0.084038 ± 0.04635	0.058583 ± 0.04733	0.0 ± 0.0
		G	0.0 ± 0.0	0.331876 ± 0.11123	0.169977 ± 0.134276	0.254805 ± 0.09408	0.179253 ± 0.12852	0.0 ± 0.0

References

[1] G. Douzas and F. Bacao, "Self-organizing map oversampling (somo) for imbalanced data set learning," *Expert Systems with Applications*, vol. 82, pp. 40–52, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417417302324

[2] A. Fernández, V. López, M. Galar, M. J. del Jesus, and F. Herrera, "Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches," *Knowledge-Based Systems*, vol. 42, pp. 97–110, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705113000300

[3] D. A. Cieslak and N. V. Chawla, "Start globally, optimize locally, predict globally: Improving performance on imbalanced data," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 143–152.

[4] T. R. Hoens and N. V. Chawla, *Imbalanced Learning: Foundations, Algorithms, and Applications.* John Wiley and Sons, Inc., 2013, ch. 3.2.

[5] A. Fenandez, S. Garcia, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from Imbalanced Data Sets.* Springer, 2018, p. 83.

[6] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing*, D.-S. Huang, X.-P. Zhang, and G.-B. Huang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 878–887.

[7] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322–1328.

[8] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

[9] D. Dua and C. Graff, "Uci machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[10] Little, M.A., McSharry, P.E., Roberts, S.J. et al. Exploiting Nonlinear Recurrence and Fractal Scaling Properties for Voice Disorder Detection. BioMed Eng OnLine 6, 23 (2007). https://doi.org/10.1186/1475-925X-6-23.

[11] G. S. Sampaio, A. R. de Aguiar Vallim Filho, L. S. da Silva, and L. A. da Silva, "Prediction of motor failure time using an artificial neural network," *Sensors.*

[12] R. Bhatt, *Fuzzy-Rough Approaches for Pattern Classification: Hybrid measures, Mathematical analysis, Feature selection algorithms, Decision tree algorithms, Neural learning, and Applications.* Independently, 2017.

[13] J. Rohra, B. Perumal, S. J.N., P. Thakur, and R. Bhatt, *User Localization in an Indoor Environment Using Fuzzy Hybrid of Particle Swarm Optimization Gravitational Search Algorithm with Neural Networks*, 02 2017, pp. 286–295.

[14] P. McCullagh, "Generalized linear models," *European Journal of Operational Research*, vol. 16, no. 3, pp. 285–292, 1984. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221784902820

[15] Friedman, Jerome H. "Greedy Function Approximation: A Gradient Boosting Machine." The Annals of Statistics, vol. 29, no. 5, Institute of Mathematical Statistics, 2001, pp. 1189–232, http://www.jstor.org/stable/2699986.