Journal of Data Intelligence, Vol. 2, No. 1 (2021) 030–46

AN ENSEMBLE FRAMEWORK OF MULTI-RATIO UNDERSAMPLING-BASED IMBALANCED CLASSIFICATION

TAKAHIRO KOMAMIZU
Nagoya University Nagoya, Japan
taka-coma@acm.org

YASUHIRO OGAWA
Nagoya University Nagoya, Japan

KATSUHIKO TOYAMA
Nagoya University Nagoya, Japan

Class imbalance is commonly observed in real-world data, and it is problematic in that it degrades classification performance due to biased supervision. Undersampling is an effective resampling approach to the class imbalance. The conventional undersampling-based approaches involve a single fixed sampling ratio. However, different sampling ratios have different preferences toward classes. In this paper, an undersampling-based ensemble framework, MUEnsemble, is proposed. This framework involves weak classifiers of different sampling ratios, and it allows for a flexible design for weighting weak classifiers in different sampling ratios. To demonstrate the principle of the design, in this paper, a uniform weighting function and a Gaussian weighting function are presented. An extensive experimental evaluation shows that MUEnsemble outperforms undersampling-based and oversampling-based state-of-the-art methods in terms of recall, gmean, F-measure, and ROC-AUC metrics. Also, the evaluation showcases that the Gaussian weighting function is superior to the uniform weighting function. This indicates that the Gaussian weighting function can capture the different preferences of sampling ratios toward classes. An investigation into the effects of the parameters of the Gaussian weighting function shows that the parameters of this function can be chosen in terms of recall, which is preferred in many real-world applications.

Keywords: imbalanced classification, resampling, undersampling, ensemble

1. Introduction

Class imbalance is a crucial problem in real-world applications that degrades classification performance, especially with minority classes. Class imbalance refers to a situation with datasets in which the number of examples in a class is much larger than that in other classes. The large difference in terms of the numbers of examples causes classifiers to be biased toward the majority class. Class imbalance has been observed and dealt with in various domains, such as the clinical domain [7], economic domain [28] and agricultural domain [32], and in software engineering [29] and computer networks [13].

Resampling is an effective solution to class imbalance, and it has been widely studied [8, 22, 5, 33]. Resampling techniques can be roughly classified into two categories: oversampling (e.g., SMOTE [8] and SWIM [33]) and undersampling (e.g., EasyEnsemble [22] and

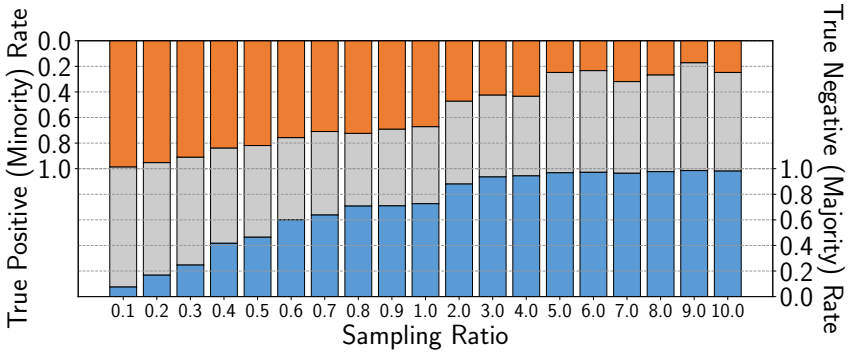

Fig. 1. Small sampling ratios prefer minority, and vice versa.

RUSBoost [31]). Undersampling is a simple and powerful resampling technique for dealing with class imbalance [11]. Not only using single-shot undersampling but also combining multiple undersampled datasets in an ensemble manner have done for the problem [22, 17, 31].

A preliminary survey of various datasets on the effects of different undersampling ratios, shown in Figure 1, indicated that different undersampling ratios have different preferences toward classes. Sampling ratio refers to the ratio of the sampled majority size over the minority size. In this paper, the minority class and majority class are regarded as the positive class and negative class, respectively. A sampling ratio of 1.0 means that the majority examples are randomly selected so that the number of sampled examples equals that of the minority examples. A ratio below 1.0 means that the number of sampled majority examples is below that of the minority examples. In this paper, this is called *excessive undersampling* and its antonym is *moderate undersampling*. The figure depicts the true positive and negative ratios for different sampling ratios in the Abalone dataset. It indicates that classifiers learned with excessively undersampled datasets favor the minority class and those by moderately undersampled datasets favor the majority class, so 1.0 may not be best balanced ratio.

This paper is an attempt to combine classifiers in different sampling ratios by using an ensemble framework, **MUEnsemble**. First, the majority examples are undersampled with different sampling ratios. To utilize a large portion of majority examples, multiple undersampled datasets are generated for each sampling ratio, and a base ensemble classifier is learned by using the multiple datasets. Then, the overall classifier is an ensemble of the base classifiers. To capture the preferences of sampling ratios toward classes, MUEnsemble introduces weighting functions on the basis of the voting strategy in ensemble learning. In this paper, a uniform weighting function and *Gaussian* function are designed as the weighting functions. The former is for equivalent voting, while the latter weights classifiers in a sampling ratio by using a Gaussian function. The Gaussian function is flexible at weighting on a central sampling ratio and on the other sampling ratios.

MUEnsemble was evaluated with 31 publicly available real-world datasets in terms of standard evaluation metrics for the class imbalance problem, that is, precision, recall, gmean, f-measure, and ROC-AUC. The datasets feature a wide range of the dimensionalities, numbers of records, and imbalance ratios (the dataset statistics are shown in Section 4). MUEnsemble was compared with the state-of-the-art methods, and it outperformed these methods in terms

of recall-preferred metrics (i.e., recall, gmean, and f-measure).

The contributions of this paper are abridged as follows.

- **Multi-ratio Undersampling-based Ensemble Framework**: In this paper, an ensemble framework, MUEnsemble, for imbalanced data is proposed. MUEnsemble employs multiple undersampling ratios including excessive undersampling. It learns an ensemble classifier for each sampling ratio and combines the ensemble classifiers among sampling ratios in an ensemble manner. It has two weighting functions, uniform and Gaussian, for ensemble classifiers corresponding to sampling ratios.
- **Extensive Experimentation on Various Public Datasets**: MUEnsemble is compared with the state-of-the-art resampling methods by using various datasets that are publicly available. The datasets include various imbalance ratios in a wide range of domains. Standard evaluation metrics for the class imbalance problem are used, namely, precision, recall, gmean, f-measure, and ROC-AUC. The experiment demonstrates that MUEnsemble outperforms the state-of-the-art methods in terms of recall-preferred metrics. To show the flexibility of the Gaussian weighting function, the possible combinations of parameters are investigated.

The rest of this paper is organized as follows. Section 2 introduces related resampling-based approaches for imbalanced data. Section 3 explains the proposed framework, MUEnsemble, and four balancing functions. Section 4 demonstrates the effectiveness of MUEnsemble over the state-of-the-art methods, and the effects of the excessive undersampling, balancing functions, and parameter optimization are discussed. Finally, Section 5 concludes this paper.

2. Related Work: Resampling Approaches

To deal with class imbalance, there are basically three groups of approaches, namely, resampling, cost-adjustment [10, 20], and algorithm modification [2, 37]. Resampling is the most commonly used because it has been shown to have robust performance and is applicable to any classifiers. Resampling approaches are roughly classified into two categories, namely, oversampling and undersampling.

2.1. *Oversampling-based Approaches*

A simple oversampling approach is to randomly copy minority examples so that the numbers of minority and majority examples become the same. This approach easily causes overfitting. To cope with the overfitting problem, oversampling approaches generate synthetic minority examples that are close to the minority. SMOTE [8] is the most popular synthetic oversampling method. It generates synthetic minority examples on the basis of the nearest neighbor technique. Since SMOTE does not take majority examples into consideration, the generated examples can easily overlap with majority examples. This degrades the classification performance. To overcome the weakness of SMOTE, more recent approaches have incorporated majority examples into the resampling process. SMOTE-Tomek [4] and SMOTE-ENN [5] employ data cleansing techniques including the removal of Tomek links [35] and Edit Nearest Neighbours [38]. Along this line, there are more advanced approaches (e.g., ADASYN [15], borderline-SMOTE [14], and SVM-SMOTE [26]). One of the state-of-the-art synthetic oversampling approaches is SWIM [33, 6]. SWIM utilizes the density of each minority example

with respect to the distribution of majority examples in order to generate synthetic minority examples. Comprehensive experiments by [18] were conducted to investigate a large number of SMOTE variants and to compare these variants with diverse kinds of datasets. In this investigation, PolyFitSMOTE [12] and ProWSyn [3] showed the best performances.

2.2. *Undersampling-based Approaches*

Undersampling-based approaches can be classified into three categories: example selection and boosting and bagging ensembles. Example selection is an approach to choosing majority examples that are expected to contribute to better classification. Major approaches choose majority examples hard to distinguish from minority examples. NearMiss [25] samples majority examples close to minority examples. Instance hardness [34] is a hardness property that indicates the likelihood that an example will be misclassified.

Boosting ensemble is a learning method that gradually changes majority examples. For each iteration, boosting approaches remove a part of the majority examples. BalanceCascade [22] is a boosting approach that removes correctly classified majority examples. RUSBoost [31] is a weighted random undersampling approach for removing majority examples that are likely to be classified correctly. EUSBoost [23] introduces a cost-sensitive weight modification and an adaptive boundary decision strategy to improve model performance. Trainable Undersampling [27] is the state-of-the-art in this category. It trains a classifier by reinforcement learning.

Bagging ensemble combines multiple weak classifiers, each of which is learned on individual pieces of undersampled training data in a voting manner. Ensemble of Undersampling [17] is one of the earlier bagging approaches using undersampled training data. EasyEnsemble [22] is an ensemble-of-ensemble approach that ensembles AdaBoost classifiers for each piece of undersampled training data in a bagging manner. In [11], a comprehensive experiment on boosting and bagging approaches is reported. It shows that RUSBoost and EasyEnsemble are the best performing approaches, and they outperform oversampling-based approaches.

MUEnsemble is classified in the bagging ensemble category. The distinctions of MUEnsemble over existing undersampling approaches are as follows. First, MUEnsemble is the first ensemble approach combining multiple sampling ratios. Second, it includes excessive undersampling, which has not been considered among these approaches. Third, it gives different weights to weak classifiers on the basis of a Gaussian function.

3. MUEnsemble

MUEnsemble is an ensemble framework that involves multiple sampling ratios. Figure 2 is an overview of MUEnsemble. It consists of two phases: the *sampling phase* and *ensemble phase*. In the sampling phase, first, given that the user specified the number n of sampling ratios, a set of sampling ratios with size n is determined. Second, for each sampling ratio, the input data are undersampled several times to make a batch of undersampled datasets. In the ensemble phase, each batch of undersampled datasets is used for learning a *base* ensemble classifier. Last, the base ensemble classifiers for all sampling ratios are combined into a classifier in an ensemble manner. In MUEnsemble, there are two major parameters: the set of sampling ratios and the weights on the base classifiers. In this paper, the weights are considered to be dominant factors in classification performance, in contrast with a set of sampling ratios.

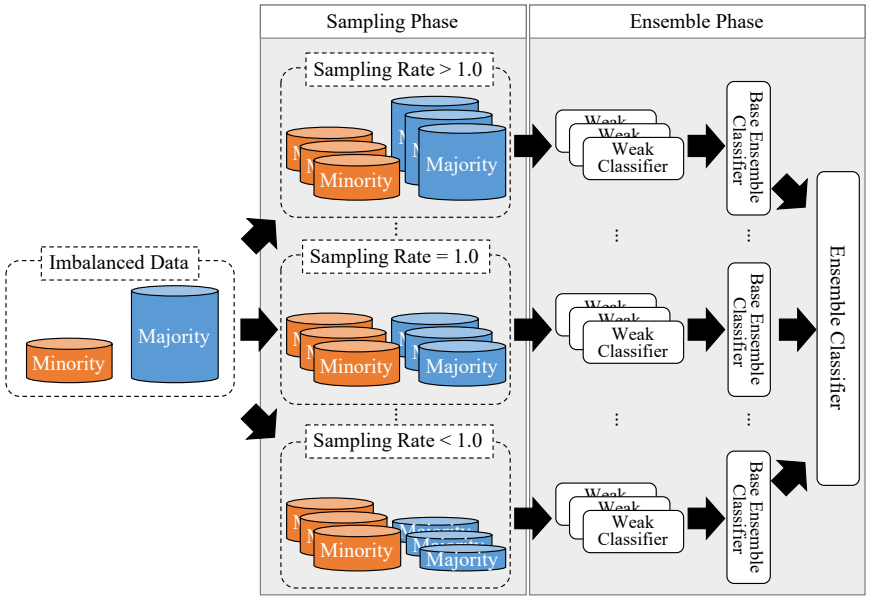

Fig. 2. Overview of MUEnsemble

Therefore, to determine a set of sampling ratios, MUEnsemble employs a simple splitting strategy. For the weights of the base classifiers, in this paper, two functions are designed; one is a uniform weighting function, and the other is a Gaussian weighting function.

Algorithm 1 displays the pseudo-code of MUEnsemble. Let $\mathcal{P}$ and $\mathcal{N}$ denote sets of minority examples and majority examples, respectively. At sampling step c, $\mathcal{N}$ is undersampled to $\mathcal{N}'$ so that $\frac{|\mathcal{N}'|}{|\mathcal{P}|} = R(c, n_{\mathcal{P}}, n_{\mathcal{N}})$, where $R(\cdot)$, $n_{\mathcal{P}}$, and $n_{\mathcal{N}}$ are a sampling ratio and the numbers of the sampling ratios below and above 1, respectively, in the splitting strategy. For $\mathcal{N}'$, MUEnsemble trains the base ensemble classifier H^c, which consists of w weak classifiers, by AdaBoost learning as $H^c = sgn\left(\sum_{j=1}^{w} \alpha_j^c h_j^c\right)$, where sgn is the sign function, h_j^c is the j-th weak classifier, and α_j^c is its weight.

$$H = sgn\left(\sum_{c=1}^{n_{\mathcal{P}}+n_{\mathcal{N}}+1} \sum_{j=1}^{w} W(c)\alpha_j^c h_j^c\right),$$

where $W(\cdot)$ is a weighting function for the sampling ratio.

3.1. *Sampling Ratio Determination*

Determining sampling ratios to be ensembled is not intuitive for human users; therefore, automatically determining them is desirable. One approach is to simplify the parameter to a ratio interval; that is, the user gives an interval value to split imbalance ratio (IR) into a sequence sampling ratios at intervals of the value. For instance, suppose that the IR is 2.0 and the ratio interval is 0.2; the sampling ratios are determined as $\{0.2, 0.4, 0.6, \ldots, 2.0\}$. However, the IR differs dataset by dataset, and when it is too large, the number of sampling ratios becomes too large. This leads to the ensemble being heavy; that is, so many base

Algorithm 1 MUEnsemble

Input: $\mathcal{P}, \mathcal{N}, n_\mathcal{P}, n_\mathcal{N}, w$
Output: H

1: **for** $c \leftarrow 1$ to $n_\mathcal{P} + n_\mathcal{N} + 1$ **do**
2: **for** $j \leftarrow 1$ to w **do**
3: Randomly sample subset $\mathcal{N}'$ from $\mathcal{N}$ s.t. $|\mathcal{N}'| : |\mathcal{P}| = R(c, n_\mathcal{P}, n_\mathcal{N}) : 1$
4: Learn H_i^c using $\mathcal{P}$ and $\mathcal{N}'$
$H_i^c = sgn\left(\sum_{j=1}^{w} \alpha_j^c h_j^c\right)$
5: **end for**
6: **end for**
7: $H = sgn\left(\sum_{c=1}^{n_\mathcal{P}+n_\mathcal{N}+1} W(c)\alpha_j^c h_j^c\right)$

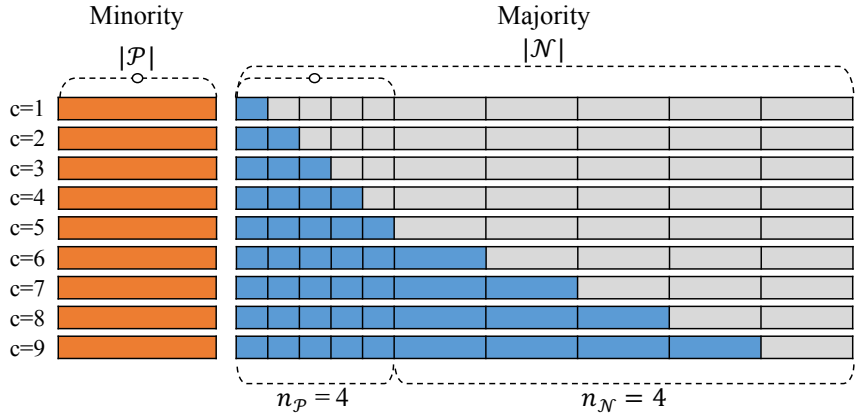

Fig. 3. Splitting Strategy

classifiers are trained, and as a result, the training cost becomes too large.

To avoid this issue, MUEnsemble employs a splitting strategy for determining the sampling ratio. It takes the numbers $n_\mathcal{P}$ and $n_\mathcal{N}$ to determine the number $|\mathcal{N}'|$ of sampled majority examples. $n_\mathcal{P}$ (resp. $n_\mathcal{N}$) is the number of blocks that equally split the minority (resp. majority) example size. Figure 3 illustrates an example of the splitting strategy. The majority example size is divided into $n_\mathcal{P} + n_\mathcal{N} + 1$ blocks ($n_\mathcal{P} = n_\mathcal{N} = 4$ in the example).

Given $n_\mathcal{P}$ and $n_\mathcal{N}$, the sampling ratio determination function R calculates the sampling ratio as a cumulative ratio of blocks as follows.

$$R(c, n_\mathcal{P}, n_\mathcal{N}) = \begin{cases} \frac{c}{n_\mathcal{P}+1} & \text{if } c \leq n_\mathcal{P} \\ 1 & \text{if } c = n_\mathcal{P} + 1 \\ 1 + \frac{IR}{n_\mathcal{N}}(c - n_\mathcal{P}) & \text{if } c > n_\mathcal{P} + 1 \end{cases}, \tag{1}$$

where $IR = \frac{|\mathcal{N}|}{|\mathcal{P}|}$ is the imbalance ratio, which is the number of majority examples over that of minority examples. In Figure 3, the sampling ratios of the corresponding iteration c are the blue-shaded areas. The amounts of blue-shaded and orange-shaded areas for the minority and majority examples are used in the training.

3.2. *Weighting Function*

The weighting function is a programmable function for deciding the weights on base classifiers of sampling ratios. To investigate the base choices for the weight function, in this paper, the aforementioned uniform weighting function, and Gaussian weighting function are introduced.

The uniform weighting function is a function that equally weights base ensemble classifiers. In particular, it provides the same weights for all base classifiers, which is defined as follows.

$$W_{uni}(c) = \frac{1}{n_{\mathcal{P}} + n_{\mathcal{N}} + 1} \tag{2}$$

The Gaussian weighting function is a more flexible function than the uniform weighting function. Different datasets can have different preferences toward sampling ratios. To capture the differences, a Gaussian function is introduced as follows.

$$W_{gauss}(c) = a \cdot \exp\left(-\frac{\left(R(c, n_{\mathcal{P}}, n_{\mathcal{N}}) - \mu\right)^2}{2\sigma^2}\right), \tag{3}$$

where μ and σ^2 are tunable parameters, and a is a normalization constant such that $a = \left(\sum_c W_{gauss}(c)\right)^{-1}$. When $\mu = 1.0$, most of the weight is on the base classifier trained using the balanced data, and the weights gradually decrease as $R(c, n_{\mathcal{P}}, n_{\mathcal{N}})$ increases and decreases from μ.

4. Experimental Evaluation

In this paper, MUEnsemble was evaluated to answer three questions:

Q1: Does MUEnsemble outperform baseline methods? — To answer this question, MUEnsemble was compared with state-of-the-art resampling-based approaches by using datasets featuring a wide range of domains, dimensionalities, numbers of records, and imbalance ratios.

Q2: How effective is taking multi-ratio undersampling into consideration for ensemble classifiers? — To answer this question, MUEnsemble was compared with EasyEnsemble [22] which is a single sampling ratio version of MUEnsemble.

Q3: Is the Gaussian weighting function potentially better than the uniform weighting function? — To answer this question, MUEnsemble with the Gaussian weighting function and that with the uniform weighting function were compared. The comparison was twofold: the best parameters were compared, and the possible parameters were compared, and this showed the relationship between the parameters of the Gaussian weighting and uniform weighting functions.

4.1. *Settings*

4.1.1. *Evaluation*

The evaluation metrics were *Precision*, *Recall*, *Gmean*, F_2, and AUC. Let TP, FN, TN, and FP be the true positives, false negatives, true negatives, and false positives. *Precision* =

Table 1. Classification Datasets

ID	Name	#dim	#major	#minor	IR
D1	Abalone (9 v. 18)	8	689	42	16.4
D2	Anuran Calls (Lept. v. Bufo.)	22	4,420	68	65.0
D3	Covertype (2 v. 5)	54	283,301	9,493	29.8
D4	default of credit card clients	23	23,364	6,636	3.5
D5	HTRU2	8	16,259	1,639	9.9
D6	Online Shoppers Purchasing Intention	17	10,422	1,908	5.5
D7	Polish companies bankruptcy	64	5,500	410	13.4
D8	Spambase	57	2,788	1,813	1.5
D9	Wine Quality – Red ((3, 4) v. others)	11	1,536	63	24.4
D10	Wine Quality – White (7 v. 3)	11	880	20	44.0
D11	Churn Modelling	9	7,963	2,037	3.9
D12	Credit Card Fraud Detection	29	284,315	492	577.9
D13	ECG Heartbeat – Arrhythmia (N v. F)	187	90,589	803	112.8
D14	Financial Distress	85	3,536	136	26.0
D15	LoanDefault LTFS AV	39	182,543	50,611	3.6
D16	Mafalda Opel– Driving Style	14	9,530	2,190	4.4
D17	Mafalda Peugeot – Driving Style	14	12,559	678	18.5
D18	Rain in Australia	20	110,316	31,877	3.5
D19	Surgical	24	10,945	3,690	3.0
D20	Paysim1	10	6,354,407	8,213	773.7
D21	cm1	21	449	49	9.2
D22	kc3	39	415	43	9.7
D23	mw1	37	372	31	12.0
D24	pc1	21	1,032	77	13.4
D25	pc3	37	1,403	160	8.8
D26	pc4	37	1,280	178	7.2
D27	Yeast (1 v. 7)	7	429	30	14.3
D28	Yeast (6 v. others)	8	1,449	35	41.4
D29	Abalone (19 v. others)	8	4,142	32	129.4
D30	Wine Quality – Red (3 v. 5)	8	1,890	26	72.7
D31	Abalone (20 v. (8, 9, 10))	11	681	10	68.1

$\frac{TP}{TP+FP}$ measures how many classified positive instances are correctly classified. $Recall = \frac{TP}{TP+FN}$ measures how many positive (minority) instances are correctly classified. $Gmean = \sqrt{Recall \cdot TNR}$ is the geometric mean of the recalls of both classes, where $TNR = \frac{TN}{TN+FP}$. $F_\beta = \frac{(1+\beta^2)Recall \cdot Precision}{Recall + \beta^2 Precision}$ is the harmonic mean of recall and precision, where β determines the weight on the recall. In this experiment, β was set to 2 because higher recalls are preferred in many real-world applications. AUC is the area under the receiver operating characteristic (ROC) curve.

To accurately estimate these evaluation metric values, the experimental process was repeated 50 times. In the process, a dataset was randomly separated into 70% for training and 30% for testing, and the classifiers were trained on the training set and evaluated by using the test set. The overall metric scores were the macro average of the 50 trials.

4.1.2. *Datasets*

This evaluation was performed on publicly available datasets that have a wide range of dataset features. Table 1 summarizes the datasets. D1 - D10 were obtained from the UCI Machine Learning Repository [9], D11 - D20 were obtained from the Kaggle Dataset,[a] D21 - D26 were obtained from the OpenML dataset [36], and D27 - D31 were obtained from the KEEL repository [1]. These datasets include few categorical attributes, and when categorical attributes exist, they are dictionary-encoded to numeric attributes.[b] Some of the datasets were for the multi-class classification task, and two of the classes in the datasets were selected to be for the binary classification task, which are represented in the dataset column in brackets. The datasets had various characteristics in terms of dimensionality (#dim.), the number of majority and minority examples (#major, #minor), and the imbalance ratio (IR).

4.1.3. *Baselines*

MUEnsemble was compared with simple baselines, synthetic oversampling approaches and undersampling-based approaches, listed as follows.

- SMT: SMOTE [8] + AdaBoost Classifier [30]
- PWS: ProWSyn [3] + AdaBoost Classifier
- PFS: SMOTE with Polynomial Fitting [12] + AdaBoost Classifier
- RUS: Random undersampling with sampling ratio of 1 + AdaBoost Classifier
- RBS: RUSBoost [31]
- EE: EasyEnsemble [22]

SMT, PWS, and PFS are synthetic oversampling methods, where SMT is the most popular method, PRW and PFS have shown superior performances in [18]. RUS is a baseline approach to undersampling-based classifiers. It is a simple undersampling with a sampling ratio of 1.0. EasyEnsemble and RUSBoost are undersampling-based ensemble methods, where EasyEnsemble is a closer algorithm to MUEnsemble except for the inclusion of various sampling ratios, and RUSBoost is a boosting ensemble method. To compare the effects of resampling, the classifiers of the synthetic oversampling approaches and RUS were set to the AdaBoost classifier [30], which is the base classifier of EasyEnsemble and MUEnsemble. The implementations of SMT, PWS, and PFS were in [19], and those of RUS, EasyEnsemble, and RUSBoost were in the imbalanced-learn library [21]. Classifiers were learned using scikit-learn (v. 0.20.3).[c]

4.1.4. *Parameters*

The parameters of MUEnsemble were set as follows. $n_{\mathcal{P}} = n_{\mathcal{N}} = 10$. The parameter spaces of the Gaussian function, μ and σ^2, were set as $\mu \in \{0.2, 0.6, 1.0, 1.4, 1.8, 2.0\}$, and $\sigma^2 \in \{0.2, 0.4, 0.6, \ldots, 3.0\}$. The parameters of the baselines were the default values.

4.2. *Results*

Table 2. *Precision* Comparison

	Oversampling			Undersampling			MUEnsemble	
data	SMT	PWS	PFS	RUS	RBS	EE	Uni	Gau
D1	0.279	0.275	**0.402**	0.136	0.200	0.158	0.118	0.149
D2	0.998	0.998	0.998	0.999	0.999	0.999	0.999	**1.000**
D3	0.142	0.246	**0.572**	0.135	0.139	0.140	0.137	0.140
D4	0.626	0.631	**0.668**	0.455	0.451	0.468	0.440	0.648
D5	0.749	0.730	**0.900**	0.690	0.491	0.735	0.698	0.728
D6	0.611	0.621	**0.666**	0.533	0.472	0.536	0.519	0.545
D7	0.619	0.633	**0.849**	0.451	0.482	0.498	0.421	0.481
D8	0.918	0.918	**0.927**	0.903	0.903	0.914	0.888	0.910
D9	0.977	0.974	0.966	0.977	0.969	0.986	0.986	**0.990**
D10	0.984	0.983	0.985	0.992	0.982	0.991	0.994	**1.000**
D11	0.635	0.616	**0.694**	0.457	0.456	0.457	0.429	0.625
D12	0.083	0.109	**0.340**	0.030	0.040	0.044	0.034	0.045
D13	0.120	0.141	**0.841**	0.078	0.097	0.093	0.080	0.093
D14	0.271	0.277	**0.412**	0.150	0.165	0.171	0.145	0.171
D15	0.295	0.285	**0.464**	0.295	0.296	0.297	0.286	0.432
D16	0.923	0.893	0.859	0.935	0.929	0.944	0.955	**0.991**
D17	0.979	0.972	0.963	0.988	0.982	0.991	0.994	**0.999**
D18	0.564	0.604	**0.694**	0.516	0.518	0.519	0.504	0.663
D19	0.734	0.756	**0.854**	0.634	0.621	0.646	0.610	0.805
D20	0.047	0.050	**0.954**	0.036	0.038	0.037	0.036	0.037
D21	0.211	**0.245**	0.227	0.164	0.181	0.177	0.153	0.184
D22	0.338	0.343	**0.439**	0.191	0.199	0.189	0.168	0.193
D23	0.265	0.272	**0.290**	0.156	0.165	0.163	0.147	0.158
D24	**0.319**	0.262	0.305	0.177	0.163	0.194	0.155	0.200
D25	0.310	0.285	**0.350**	0.228	0.217	0.241	0.205	0.232
D26	0.529	0.527	**0.616**	0.410	0.435	0.424	0.379	0.440
D27	0.241	0.178	**0.262**	0.123	0.163	0.155	0.125	0.176
D28	0.336	0.285	**0.493**	0.115	0.213	0.142	0.122	0.141
D29	0.028	**0.029**	0.005	0.017	0.022	0.020	0.016	0.021
D30	0.237	0.223	**0.400**	0.053	0.128	0.062	0.049	0.061
D31	**0.064**	0.031	0.025	0.031	0.042	0.039	0.036	0.034

4.2.1. *Q1: Does MUEnsemble outperform baseline methods?— Yes, MUEnsemble achieved the best performance in terms of recall, gmean, F_2, and AUC.*

Tables 2, 3, 4, 5, and 6 show performance comparisons in terms of precision, recall, gmean, F_2, and AUC, respectively. For the recall, gmean, F_2, and AUC metrics, MUEnsemble outperformed the other methods. For the precision metric, the oversampling-based approaches, especially SMOTE with polynomial fitting (PFS), performed the best. On the contrary, the recall scores of the oversampling-based approaches were lower than those of the undersampling-based approaches including MUEnsemble. In many real-world applications such as credit card fraud detection and arrhythmia detection, higher recall is preferred. From this perspective, the undersampling-based approaches are more preferable than the oversampling-based

[a] `https://www.kaggle.com/datasets/`
[b] Coping with categorical attributes is out of the scope of this paper.
[c] `https://scikit-learn.org/`

Table 3. *Recall* Comparison

	Oversampling			Undersampling			MUEnsemble	
data	SMT	PWS	PFS	RUS	RBS	EE	Uni	Gau
D1	0.588	0.511	0.328	0.658	0.463	0.717	0.803	**0.931**
D2	0.886	0.863	0.865	0.947	0.914	0.961	0.968	**0.971**
D3	0.872	0.521	0.133	0.896	0.887	0.905	0.921	**0.992**
D4	0.411	0.406	0.357	0.616	0.621	0.613	0.649	**0.977**
D5	0.902	0.909	0.852	0.912	0.752	0.914	0.919	**0.960**
D6	0.677	0.655	0.568	0.812	0.733	0.818	0.836	**0.957**
D7	0.853	0.828	0.732	0.901	0.811	0.917	0.934	**0.976**
D8	0.926	0.926	0.915	0.932	0.923	0.939	0.956	**0.976**
D9	0.511	0.447	0.242	0.624	0.377	0.764	0.832	**0.968**
D10	0.283	0.283	0.350	0.720	0.273	0.683	0.783	**1.000**
D11	0.553	0.538	0.457	0.733	0.723	0.734	0.769	**0.947**
D12	0.897	0.878	0.827	0.917	0.875	0.915	0.922	**0.959**
D13	0.876	0.834	0.456	0.914	0.868	0.910	0.925	**0.971**
D14	0.593	0.456	0.259	0.826	0.540	0.878	0.921	**0.983**
D15	0.181	0.121	0.015	0.631	0.628	0.634	0.705	**0.993**
D16	0.728	0.593	0.360	0.782	0.756	0.816	0.862	**0.984**
D17	0.675	0.533	0.329	0.828	0.727	0.869	0.915	**0.994**
D18	0.664	0.606	0.490	0.762	0.760	0.763	0.784	**0.951**
D19	0.708	0.703	0.666	0.817	0.813	0.814	0.832	**0.951**
D20	0.944	0.934	0.483	0.983	0.981	0.984	0.985	**0.996**
D21	0.347	0.393	0.153	0.648	0.388	0.716	0.892	**0.960**
D22	0.423	0.423	0.331	0.697	0.358	0.772	0.858	**0.962**
D23	0.320	0.390	0.260	0.686	0.308	0.708	0.784	**0.970**
D24	0.550	0.446	0.242	0.738	0.352	0.839	0.934	**0.971**
D25	0.483	0.417	0.300	0.746	0.512	0.798	0.887	**0.952**
D26	0.704	0.730	0.509	0.880	0.695	0.925	0.949	**0.980**
D27	0.467	0.467	0.344	0.640	0.398	0.756	0.809	**0.967**
D28	0.555	0.609	0.482	0.838	0.651	0.847	0.820	**0.964**
D29	0.450	0.400	0.004	0.698	0.340	0.830	0.930	**0.960**
D30	0.713	0.637	0.325	0.792	0.552	0.853	0.833	**0.963**
D31	0.167	0.067	0.067	0.613	0.193	0.693	0.740	**0.900**

approaches.

4.2.2. *Q2: How effective is taking multi-ratio undersampling into consideration for ensemble classifiers?— The effectiveness can be seen from the fact that MUEnsemble outperformed EasyEnsemble in most cases, and was comparable in the other cases.*

In the undersampling-based approaches, EasyEnsemble (EE) was the closest approach to MUEnsemble in the sense that EasyEnsemble ensembles multiple weak classifiers with a fixed undersampling ratio. The differences in recall, gmean, and F_2 scores between MUEnsemble and EasyEnsemble show the effect of involving multiple sampling ratios.

4.2.3. *Q3: Is the Gaussian weighting function potentially better than the uniform weighting function?— Yes, the Gaussian weighting function had clear superiority to the uniform weighting function.*

Table 4. *Gmean* Comparison

	Oversampling			Undersampling			MUEnsemble	
data	SMT	PWS	PFS	RUS	RBS	EE	Uni	Gau
D1	0.723	0.675	0.552	0.686	0.625	**0.734**	0.703	**0.734**
D2	0.940	0.927	0.928	0.947	0.952	0.967	**0.972**	0.966
D3	0.847	0.702	0.364	0.851	0.851	0.858	**0.862**	0.860
D4	0.618	0.615	0.582	0.698	0.688	0.701	**0.704**	**0.704**
D5	0.935	0.937	0.919	0.935	0.806	0.940	0.939	**0.941**
D6	0.789	0.779	0.733	0.840	0.774	0.844	0.847	**0.850**
D7	0.905	0.893	0.851	0.909	0.870	**0.924**	0.919	**0.924**
D8	0.936	0.936	0.934	0.933	0.929	0.940	0.938	**0.941**
D9	0.661	0.604	0.452	0.638	0.537	**0.711**	0.622	0.710
D10	0.478	0.464	0.537	0.713	0.440	0.732	0.760	**0.807**
D11	0.712	0.701	0.658	0.754	0.748	0.755	0.753	**0.758**
D12	0.939	0.931	0.908	0.932	0.917	**0.940**	0.938	0.939
D13	0.909	0.892	0.674	0.909	0.897	**0.915**	**0.915**	**0.915**
D14	0.745	0.658	0.502	0.821	0.686	0.856	0.852	**0.860**
D15	0.398	0.332	0.120	0.606	0.606	**0.609**	0.601	**0.609**
D16	0.736	0.680	0.569	0.747	0.742	0.762	**0.764**	0.762
D17	0.751	0.685	0.556	0.803	0.763	0.829	0.822	**0.836**
D18	0.752	0.732	0.678	0.777	0.777	0.780	0.780	**0.781**
D19	0.804	0.806	0.800	0.829	0.817	0.832	0.826	**0.834**
D20	0.960	0.955	0.695	0.974	0.974	**0.976**	0.975	**0.976**
D21	0.532	0.575	0.345	0.634	0.540	0.667	0.629	**0.695**
D22	0.610	0.611	0.554	0.687	0.533	0.708	0.686	**0.722**
D23	0.538	0.556	0.481	0.668	0.489	0.682	0.675	**0.684**
D24	0.703	0.631	0.470	0.731	0.537	0.781	0.749	**0.789**
D25	0.650	0.603	0.528	0.727	0.630	**0.753**	0.733	0.740
D26	0.800	0.813	0.696	0.849	0.777	0.872	0.860	**0.879**
D27	0.635	0.616	0.559	0.645	0.561	0.726	0.690	**0.759**
D28	0.723	0.756	0.681	0.829	0.768	**0.856**	0.831	0.844
D29	0.624	0.584	0.013	0.680	0.505	0.744	0.705	**0.758**
D30	0.828	0.782	0.558	0.785	0.709	0.830	0.797	**0.840**
D31	0.249	0.114	0.113	0.613	0.302	0.692	**0.696**	0.653

In most cases, MUEnsemble with the Gaussian weighting function was superior to that with the uniform weighting function for all metrics. This indicates that there are parameters that lead MUEnsemble to perform better than just voting from the base classifiers.

Figure 4 and 5 show the effects of parameters for the Gaussian weighting function in comparison with the uniform weighting function. To avoid redundancy in showing these effects across all datasets, D4 and D20 are shown in these figures as they demonstrate the typical tendency of the effects. In these figures, the solid lines are the changes in the performance of MUEnsemble with the Gaussian weighting function, and the dashed horizontal lines are those with the uniform weighting function. Figure 4 indicates that MUEnsemble was sensitive to the parameters for D4. For instance, the lowest $\mu = 0.2$ (blue line in the figure) made the recall of MUEnsemble higher, but it leads to low gmean scores, meaning that the low μ leads to low true negative ratios. In contrast, Figure 5 shows that MUEnsemble was not that sensitive to the parameters for D20. From the both figures, larger the σ^2, the closer the performances of

Table 5. F_2 Comparison

	Oversampling			Undersampling			MUEnsemble	
data	SMT	PWS	PFS	RUS	RBS	EE	Uni	Gau
D1	**0.478**	0.431	0.334	0.366	0.354	0.418	0.371	0.406
D2	0.905	0.886	0.887	0.956	0.929	0.968	0.974	**0.977**
D3	0.430	0.426	0.158	0.422	0.428	0.432	0.430	**0.433**
D4	0.441	0.437	0.394	0.575	0.574	0.577	0.592	**0.631**
D5	0.867	0.866	0.862	0.857	0.652	**0.871**	0.864	**0.871**
D6	0.662	0.648	0.585	0.735	0.650	0.740	0.745	**0.750**
D7	**0.792**	0.779	0.752	0.750	0.711	0.784	0.751	0.780
D8	0.925	0.924	0.917	0.926	0.919	0.934	0.941	**0.942**
D9	0.563	0.497	0.281	0.670	0.425	0.798	0.857	**0.972**
D10	0.321	0.323	0.393	0.753	0.310	0.720	0.813	**1.000**
D11	0.567	0.552	0.490	0.654	0.645	0.654	0.664	**0.670**
D12	0.303	0.364	**0.642**	0.131	0.168	0.185	0.146	0.188
D13	0.387	0.421	**0.501**	0.292	0.335	0.329	0.298	0.330
D14	0.479	0.401	0.278	0.433	0.363	**0.481**	0.444	0.478
D15	0.196	0.136	0.018	0.514	0.513	0.517	0.546	**0.593**
D16	0.760	0.636	0.407	0.809	0.785	0.839	0.880	**0.986**
D17	0.720	0.586	0.378	0.856	0.765	0.891	0.930	**0.995**
D18	0.641	0.605	0.520	0.696	0.695	0.698	0.706	**0.723**
D19	0.713	0.713	0.697	0.772	0.763	0.774	0.775	**0.777**
D20	0.197	0.207	**0.536**	0.158	0.164	0.163	0.156	0.160
D21	0.305	0.346	0.178	0.405	0.316	0.444	0.452	**0.486**
D22	0.398	0.401	0.344	0.452	0.300	0.476	0.469	**0.496**
D23	0.303	0.395	0.263	0.406	0.256	0.420	0.417	**0.421**
D24	0.479	0.387	0.248	0.448	0.276	0.503	0.464	**0.513**
D25	0.434	0.380	0.308	0.512	0.399	**0.545**	0.533	0.529
D26	0.659	0.677	0.526	0.715	0.619	0.747	0.729	**0.758**
D27	0.382	0.351	0.320	0.343	0.295	0.422	0.383	**0.461**
D28	0.478	**0.486**	0.478	0.365	0.441	0.422	0.379	0.408
D29	0.112	**0.113**	0.097	0.079	0.094	0.092	0.075	0.096
D30	**0.500**	0.461	0.334	0.206	0.320	0.238	0.198	0.236
D31	**0.301**	0.270	0.251	0.135	0.205	0.165	0.156	0.156

MUEnsemble are with the Gaussian and uniform weighting functions are. This is a natural phenomenon because larger the σ^2, the flatter distribution of the Gaussian function becomes.

4.3. *Lessons Learned*

- **Oversampling prefers precision and undersampling prefers recall.** This is not that surprising but it is noteworthy. Oversampling methods generate examples of a minority class on the basis of the distribution of observed examples; therefore, they increase the density of the data points representing them, and thus, classifiers learn a boundary around them. This indicates that these classifiers can classify unobserved examples within the boundary in the minority class with high confidence; therefore, high precision can be achieved. In contrast, undersampling methods randomly skew examples of the majority class, while those of the minority class remain as they are.

Table 6. *AUC* Comparison

	Oversampling			Undersampling			MUEnsemble	
data	SMT	PWS	PFS	RUS	RBS	EE	Uni	Gau
D1	0.814	0.805	0.798	0.767	0.743	0.810	0.816	**0.820**
D2	0.994	0.993	0.995	0.982	**0.998**	0.997	0.997	0.996
D3	0.927	0.901	0.930	0.928	0.928	0.934	**0.936**	**0.936**
D4	0.758	0.759	0.770	0.769	0.768	**0.775**	0.774	**0.775**
D5	0.976	0.977	0.976	0.976	0.879	0.977	0.977	**0.980**
D6	0.912	0.911	0.917	0.913	0.861	0.917	**0.918**	**0.918**
D7	0.975	0.968	0.972	0.967	0.946	0.976	0.975	**0.978**
D8	0.979	0.980	0.979	0.978	0.977	0.981	**0.982**	**0.982**
D9	0.734	0.654	0.635	0.685	0.614	0.769	0.772	**0.784**
D10	0.791	0.722	0.740	0.802	0.711	0.846	0.883	**0.894**
D11	0.842	0.836	**0.843**	0.835	0.830	0.836	0.835	0.838
D12	0.976	0.971	0.960	0.973	0.961	0.978	0.980	**0.981**
D13	0.956	0.948	0.965	0.964	0.955	0.969	**0.972**	0.971
D14	0.902	0.908	0.900	0.902	0.840	0.932	0.933	**0.940**
D15	0.601	0.597	0.647	0.648	0.649	**0.652**	**0.652**	**0.652**
D16	0.819	0.789	0.790	0.826	0.825	0.841	**0.842**	0.839
D17	0.860	0.842	0.840	0.886	0.861	0.905	0.905	**0.913**
D18	0.850	0.846	0.853	0.862	0.862	0.864	0.865	**0.866**
D19	0.900	0.902	0.910	0.912	0.907	0.912	0.912	**0.913**
D20	0.993	0.993	**0.997**	0.996	0.996	**0.997**	**0.997**	**0.997**
D21	0.735	0.749	0.695	0.688	0.629	0.737	0.762	**0.785**
D22	0.763	0.741	0.728	0.745	0.618	0.805	**0.822**	0.812
D23	0.734	0.763	0.727	0.743	0.668	**0.802**	0.795	0.775
D24	0.839	0.830	0.819	0.791	0.652	0.851	0.855	**0.861**
D25	0.805	0.808	0.795	0.792	0.706	0.816	**0.826**	0.808
D26	**0.929**	0.922	0.907	0.904	0.884	0.921	0.928	0.923
D27	0.750	0.760	0.766	0.715	0.692	0.807	0.807	**0.812**
D28	0.861	0.895	0.867	0.909	0.902	0.937	0.920	**0.943**
D29	0.767	0.776	0.753	0.748	0.708	0.815	**0.823**	0.816
D30	0.882	0.854	0.845	0.856	0.877	0.923	0.911	**0.929**
D31	0.768	0.675	0.702	0.690	0.654	0.805	**0.834**	0.731

On the basis of the skewed majority examples, classifiers can learn a more relaxed boundary between minority and majority classes. This indicates that these classifiers eagerly classify examples close to the observed minority examples as the minority class. Therefore, undersampling-based approaches increase recall.

- **Undersampling with different sampling ratios has different preferences toward classes.** Ensemble learning, which is a typical approach to achieving higher performance in classification tasks, prefers to combine classifiers having diverse classification criteria. EasyEnsemble combines classifiers trained on different subsets of examples to realize divergence. In contrast, the proposed method, MUEnsemble, increases diversity by using different sampling ratios. Base classifiers trained on examples with different sampling ratios have diverse preferences toward classes. The experiment in this paper showed the superiority of MUEnsemble over EasyEnsemble. Therefore,

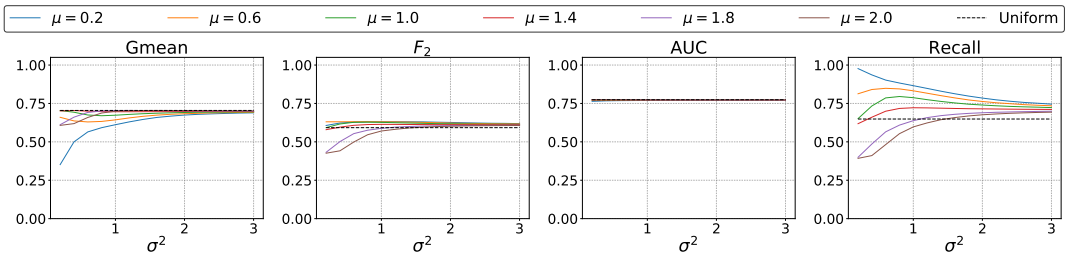

Fig. 4. Effects of Parameters for the Gaussian Weighting Scheme on D4

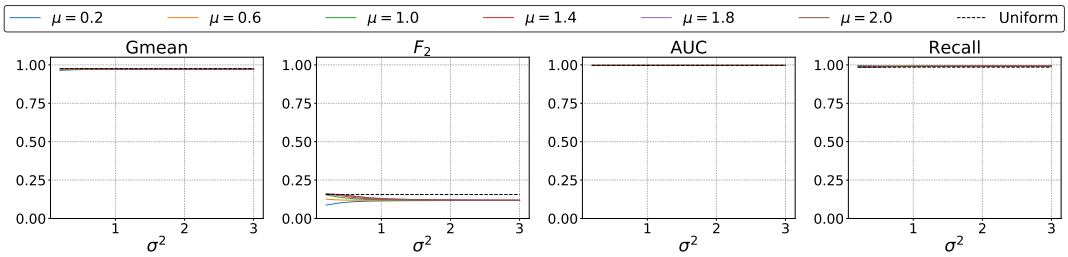

Fig. 5. Effects of Parameters for the Gaussian Weighting Scheme on D20

increasing divergence by incorporating multiple sampling ratios contributes to achieving better ensemble classifier.

- **A biased weighting scheme for ensembles can improve simple voting ensembles.** As discussed above, the combination of multiple sampling ratios increases the diversity of classifiers; however, classifiers with sampling ratios that are too low (or too high) sampling ratios are sometimes too biased toward minority class (or majority class). Classifiers taht are too biased can wrongly classify examples with quite high confidence. This can degrade the performance of ensemble classifiers. The experimental results showing that MUEnsemble with the Gaussian weighting function outperformed that with the uniform weighting function indicate that weights should be biased toward the "*balanced*" sampling ratios. "Balanced" here does not mean that the sampling ratio equals to 1.0 but rather that it is balanced for the true positive rate and true negative rate.

5. Conclusion

In this paper, an ensemble classification framework, MUEnsemble, was proposed for the class imbalance problem, by which multiple undersamplings with different sampling ratios are applied. In an experiment, the Gaussian balancing function proved its flexibility. Since there were still datasets (e.g., D15) for which none of the approaches could perform well, promising future directions would be investigating reasons, such as the possibility that *small disjuncts* [16] and the *overlapping* of example distributions of the majority and minority

classes [24] were causes, and then combining solutions to the reasons for the degradation into MUEnsemble. This evaluation also revealed a limitation of MUEnsemble and undersampling-based approaches; that is, their precision scores tended to be lower than the oversampling-based approaches. The precision-recall trade-off still remains in the class imbalance problem.

Acknowledgements

This work was partly supported by the Kayamori Foundation of Informational Science Advancement.

References

1. J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, and S. García. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. *J. Multiple Valued Log. Soft Comput.*, 17(2-3):255–287, 2011.
2. H. Bao and M. Sugiyama. Calibrated Surrogate Maximization of Linear-fractional Utility in Binary Classification. *CoRR*, abs/1905.12511, 2019.
3. S. Barua, M. M. Islam, and K. Murase. ProWSyn: Proximity Weighted Synthetic Oversampling Technique for Imbalanced Data Set Learning. In *PAKDD 2013*, pages 317–328, 2013.
4. G. E. A. P. A. Batista, A. L. C. Bazzan, and M. C. Monard. Balancing Training Data for Automated Annotation of Keywords: a Case Study. In *II Brazilian Workshop on Bioinformatics*, pages 10–18, 2003.
5. G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6(1):20–29, 2004.
6. C. Bellinger, S. Sharma, N. Japkowicz, and O. R. Zaane. Framework for extreme imbalance classification: SWIM-sampling with the majority class. *KAIS*, 2019.
7. S. Bhattacharya, V. Rajan, and H. Shrivastava. ICU Mortality Prediction: A Classification Algorithm for Imbalanced Datasets. In *AAAI 2017*, pages 1288–1294, 2017.
8. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.*, 16:321–357, 2002.
9. D. Dua and C. Graff. UCI Machine Learning Repository, 2019.
10. C. Elkan. The Foundations of Cost-Sensitive Learning. In *IJCAI 2001*, pages 973–978, 2001.
11. M. Galar, A. Fernández, E. B. Tartas, H. B. Sola, and F. Herrera. A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 42(4):463–484, 2012.
12. S. Gazzah and N. E. B. Amara. New Oversampling Approaches Based on Polynomial Fitting for Imbalanced Data Sets. In *DAS 2008*, pages 677–684, 2008.
13. S. E. Gómez, L. Hernández-Callejo, B. C. Martínez, and A. J. Sánchez-Esguevillas. Exploratory study on Class Imbalance and solutions for Network Traffic Classification. *Neurocomputing*, 343:100–119, 2019.
14. H. Han, W. Wang, and B. Mao. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In *ICIC 2005*, pages 878–887, 2005.
15. H. He, Y. Bai, E. A. Garcia, and S. Li. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *IJCNN 2008*, pages 1322–1328, 2008.
16. T. Jo and N. Japkowicz. Class Imbalances versus Small Disjuncts. *SIGKDD Explorations*, 6(1):40–49, 2004.
17. P. Kang and S. Cho. EUS SVMs: Ensemble of Under-Sampled SVMs for Data Imbalance Problems. In *ICONIP 2006*, pages 837–846, 2006.
18. G. Kovács. An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets. *Applied Soft Computing*, 83:105662, 2019. (IF-2019=4.873).
19. G. Kovács. smote-variants: a Python Implementation of 85 Minority Oversampling Techniques. *Neurocomputing*, 366:352–354, 2019. (IF-2019=4.07).

20. B. Krawczyk, M. Wozniak, and G. Schaefer. Cost-sensitive decision tree ensembles for effective imbalanced classification. *Appl. Soft Comput.*, 14:554–562, 2014.
21. 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*, 18(17):1–5, 2017.
22. X. Liu, J. Wu, and Z. Zhou. Exploratory Undersampling for Class-Imbalance Learning. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 39(2):539–550, 2009.
23. W. Lu, Z. Li, and J. Chu. Adaptive Ensemble Undersampling-Boost: A novel learning framework for imbalanced data. *Journal of Systems and Software*, 132:272–282, 2017.
24. J. Luengo, A. Fernández, S. García, and F. Herrera. Addressing data complexity for imbalanced data sets: analysis of SMOTE-based oversampling and evolutionary undersampling. *Soft Comput.*, 15(10):1909–1936, 2011.
25. I. Mani and I. Zhang. kNN approach to unbalanced data distributions: a case study involving information extraction. In *ICML'2003 Workshop on Learning from Imbalanced Datasets*, volume 126, 2003.
26. H. M. Nguyen, E. W. Cooper, and K. Kamei. Borderline over-sampling for imbalanced data classification. *IJKESDP*, 3(1):4–21, 2011.
27. M. Peng, Q. Zhang, X. Xing, T. Gui, X. Huang, Y. Jiang, K. Ding, and Z. Chen. Trainable Undersampling for Class-Imbalance Learning. In *AAAI 2019*, pages 4707–4714, 2019.
28. A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In *SSCI 2015*, pages 159–166, 2015.
29. D. Rodríguez, I. Herraiz, R. Harrison, J. J. Dolado, and J. C. Riquelme. Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction. In *EASE 2014*, pages 43:1–43:10, 2014.
30. R. E. Schapire. A Brief Introduction to Boosting. In *IJCAI 1999*, pages 1401–1406, 1999.
31. C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano. RUSBoost: A Hybrid Approach to Alleviating Class Imbalance. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 40(1):185–197, 2010.
32. A. Sharififar, F. Sarmadian, and B. Minasny. Mapping imbalanced soil classes using Markov chain random fields models treated with data resampling technique. *Computers and Electronics in Agriculture*, 159:110–118, 2019.
33. S. Sharma, C. Bellinger, B. Krawczyk, O. R. Zaïane, and N. Japkowicz. Synthetic Oversampling with the Majority Class: A New Perspective on Handling Extreme Imbalance. In *ICDM 2018*, pages 447–456, 2018.
34. M. R. Smith, T. R. Martinez, and C. G. Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.
35. I. Tomek. Two Modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, 1976.
36. J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: networked science in machine learning. *SIGKDD Explor.*, 15(2):49–60, 2013.
37. H. Wang, Y. Gao, Y. Shi, and H. Wang. A Fast Distributed Classification Algorithm for Large-Scale Imbalanced Data. In *ICDM 2016*, pages 1251–1256, 2016.
38. D. L. Wilson. Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Trans. Systems, Man, and Cybernetics*, 2(3):408–421, 1972.