Knowledge-Based Systems 213 (2021) 106689

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A novel progressively undersampling method based on the density peaks sequence for imbalanced data

Xiaoying Xie [a], Huawen Liu [b,*], Shouzhen Zeng [c,*], Lingbin Lin [d], Wen Li [e]

[a] College of Economics and Management, Zhejiang Normal University, Jinhua 321004, China
[b] College of Mathematics and Computer Science, Zhejiang Normal University, Jinhua 321004, China
[c] School of Business, Ningbo University, Ningbo 315211, China
[d] Student Management Office, Zhejiang Normal University, Jinhua 321004, China
[e] Department of Mathematics & Statistics, Curtin University, Perth WA 6845, Australia

ARTICLE INFO

Article history:
Received 3 August 2020
Received in revised form 13 December 2020
Accepted 15 December 2020
Available online 27 December 2020

Keywords:
Progressive undersampling
Density peaks sequence
Importance degree
Optimal undersampling size
Imbalanced data

ABSTRACT

Undersampling is a widely used resampling technique for imbalanced data. As traditional undersampling techniques, typically making majority and minority classes in imbalanced data into the same scale, tend to miss valuable information, many strategies like clustering have been developed. However, two essential problems still remain and require more efforts to be put; that is, which and how many instances should be extracted in undersampling. To alleviate these two problems, in this paper we propose a novel undersampling method for imbalanced data. It exploits a sequence of density peaks to progressively extract instances from the majority classes of the imbalanced data. Specifically, two factors are introduced to measure the importance degree of each instance in the majority classes. With these two factors, we generate a sampling sequence based on the importance of instances for classification. Furthermore, the optimal undersampling size of the majority classes is automatically determined by progressively extracting the important instances from the sequence. To evaluate the effectiveness of the proposed method, a series of experiments comparing to six popular undersampling methods were conducted on 40 public benchmark datasets. The experimental results show that the performance of the proposed undersampling method is superior to the state-of-the-art undersampling methods.

1. Introduction

Learning from imbalanced data is still a hot topic and a challenging problem in supervised learning. Imbalanced data are ubiquitous in reality, especially in medical diagnoses, credit fraud and network intrusion, where the quantities of instances within classes are different significantly [1–3]. Generally, the rarity instances of the minority classes contains more discriminant information and attract more interests. For example, in a medical diagnosis of a rare disease where there is critical need to identify such a rare medical condition among the normal populations because the physicians could not afford any incorrect diagnosis. However, conventional classification models usually treat the instances indiscriminately, and do not take the misclassification costs of the minority classes into consideration. Besides, the nature of data, such as distribution overlap, class inseparability and small disjuncts [4–6], increases the data complexity and deteriorate the classification performance [7].

The imbalanced problem has been extensively studied and a great number of techniques have been witnessed. Roughly speaking, they can be grouped into three categories, i.e., model-adaptive, cost-sensitive and data-driven methods [8]. The model-adaptive techniques tailor conventional classification models to adapt imbalanced scenes [9,10]. Typical algorithms include AWELM (adaptive weighted extreme learning machine) [11] and NFMID (neurofuzzy model for imbalanced data) [12]. The second kind usually treat instances in imbalance classes with different misclassification costs [13,14], where the misclassification cost of the minority class is often penalized as the imbalanced ratio. The data-driven ones, e.g., resampling, generate or extract instances from the original data spaces such that their class distributions toward balanced [15–17]. Undersampling and oversampling are two representative resampling techniques. The former shrinks the majority classes by removing instances within, whereas the latter enlarges the minority classes by copying or producing new instances. It is worthy of noting that undersampling techniques seem to be more popular than oversampling ones, which incline to be overfitting [18,19].

* Corresponding authors.
E-mail addresses: hwliu@zjnu.edu.cn (H. Liu), zengshouzhen@nbu.edu.cn (S. Zeng).

https://doi.org/10.1016/j.knosys.2020.106689
0950-7051/

Clustering is of great interest to the undersampling, for it can effectively removing boundary and noise instances of the majority classes [20,21]. Up to now, a wide number of clustering-based undersampling algorithms have been developed [22,23]. They usually take K-means to cluster the instances of the majority classes and then pick the central ones as representative. For example, Tsai et al. [24] put the instances of the majority class into 'subclasses', and then filtered unimportant data out from each of the 'subclasses'. However, the K-means algorithm can only handle spherical shapes of clusters, resulting in extracting poor representative instances. Another challenging issue of the K-means algorithm is how to set an optimal value to K. Lin et al. [23] assigned it as the size of the minority class. Thus, the size of the representative instances extracted is the same as the minority class. Visa [25] argued that a half-to-half rebalanced ratio between the minority class and the majority class might not lead to an optimal performance.

For the undersampling techniques, two essential problems still remain and require more efforts to be put; that is, which and how many representative instances should be extracted from the majority classes. In this paper, we try to cope with these two problems by introducing a novel undersampling method called PUMD (Progressive Undersampling Method with Density). The framework of PUMD is presented as Fig. 1. Given a two-class classification problem, two factors, i.e., the density ρ and the distance δ, are exploited to evaluate the importance degrees of the instances of the majority class. With the evaluation factors, a sampling sequence is generated according to the importance degrees. Then, representative instances are progressively extracted from the sampling sequence to construct a new data collection coupled with the instances of the minority class. The experimental study conducted on 40 public imbalanced datasets from the KEEL dataset repository [26] shows the effectiveness of the proposed method.

The main contributions of this paper can be summarized as follows:

- The instances of the majority classes are organized into a sampling sequence with the importance degrees, which are measured by the densities and distances of the instances to others, in a descending order.
- The representative instances are extracted from the sampling sequence in a progressive manner, so that the optimal undersampling size can be automatically determined in terms of data on hand.
- The representative instances extracted from the sampling sequence are often the core members of the majority classes. To some extent, they can be taken as delegations of the data distributions for classification.

The rest of this paper is organized as follows. Section 2 briefly discusses the related work about the undersampling methods. Section 3 introduces the proposed progressive undersampling method in detail. The experimental data and experimental setup are given in Section 4, followed by the discussion of experimental results in Section 5. Finally, Section 6 concludes the paper.

2. Related work

In this paper, we place our main focuses on the undersampling techniques for imbalanced data. Here we only briefly review the state-of-the-art about the undersampling techniques. For other techniques, interesting readers can consult good books or surveys, e.g. [27–29], and references therein to get more details.

By now, a great number of the undersampling methods have been proposed. According to the techniques used, they can be roughly divided into two major groups: KNN(nearest neighbors)-based methods [30–32] and K-means based methods [21–23].

A large percentage of the existing undersampling techniques remove instances according to the information of the nearest neighbors. The main purpose is to delete instances which are redundant, and instances in border regions, or noise. Kubat and Matwin [30] designed the One Side Selection (OSS) method, which is an undersampling method resulting from the application of Tomek links [33]. Instances in majority classes (also named as negative instances) participating in Tomek links are looked as either borderline or noise. Meanwhile, the negative instances are considered redundant if they have the same class label with their one nearest neighbor (1NN), which is determined by the application of the Condensed Nearest Neighbor (CNN) method [34]. The OSS undersampling method deletes the negative instances which are believed to be the borderline, noise or redundancy. The drawback of OSS is that too many negative instances are deleted, which may deteriorate the performance of the classifier.

In contrast to the OSS algorithm, which uses the 1NN to identify redundant instances in majority classes, Laurikkala [31] proposed the Neighborhood Cleaning Rule (NCL) to remove negative instances. The NCL algorithm applies the Edited Nearest Neighbor (ENN) method [35] to remove the negative instances whose class label differs from at least two of its three nearest neighbors (3NN), and the negative instances who is one of the three nearest neighbors of positive instances (the instances in minority classes). Owing to the role of data cleaning in minority classes, the NCL undersampling method works better than OSS.

Unlike NCL, which just deletes negative instances, Kang et al. [32] also detects and filters positive instances noise. It proposes a framework to deal with the noise in minority classes via a noise filter combined with undersampling. It divides positive instances into extremely useful instances, relatively useful instances, and noisy instances according to the proportion of positive and negative instances in their K nearest neighbors. After noise filtering of positive instances, the classifier performs better. But there are two aspects left to discuss. One is how to fix the K value, and the other is the fact that when positive instances are rare, the noise filter may fail to construct a classifier.

Clustering is another technique widely used during the undersampling procedure to get a reasonable training set [36]. Chen and Shyu [21] utilized the K-means clustering method to cluster the instances of majority classes into K different negative groups. Then each negative group is combined with the positive instances to construct K new data groups which are more balanced, and each data group trains a subspace model which is integrated to predict new instances. However, the algorithm does not address how to fix the K value.

Yen and Lee [22] proposed an undersampling method based on clustering to select the representative instances of majority classes as new training data. They first cluster all the training data into K clusters, given a ratio of negative instances and positive instances in the new training data, e.g. $1 : m$, the negative instances will be selected from each cluster, combined with all of the positive instances, a new training set is produced. The unsolved part of the algorithm is how to determine the values of K and m.

Lin et al. [23] proposed a clustering-based undersampling algorithm which clusters the majority class with the K-means method, and sets the K equal to the number of positive instances. Then, the centroids of the K clusters are selected to replace the entire majority class instances to form the new training set together with minority class instances.

Apart from the above, the K-means clustering method is used to determine whether a negative instance is noise, border or redundancy before undersampling is carried out [37–39]. Most of

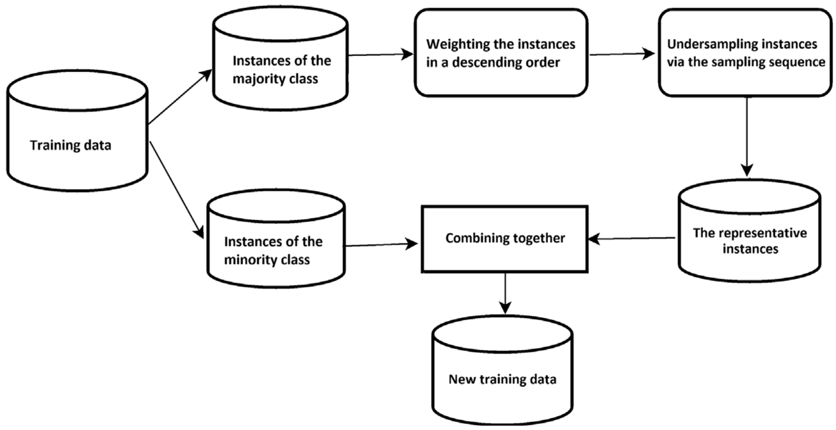


Fig. 1. The schema of the proposed undersampling method.

these methods manage to combat some weaknesses of existing undersampling algorithms, but two essential problems, which and how many instances should be extracted, still need to be considered.

3. The progressive undersampling method

As we know, a small size of instances can effectively improve both the efficiency and the performance of learning models, if the instances were carefully chosen from the original data spaces [40]. It should be pointed out that the quantity of the chosen instances should be sufficient great and they should also be representative, otherwise the performance of the learning models would be deteriorated [41,42]. How to weight the importance degrees of the instances is still an open issue. Indeed, the criteria of importance should be diverse for different models and learning tasks.

From the perspective of statistical distributions, the densities of the instances are very crucial to classification or clustering tasks. A typical example is the density peaks clustering algorithm (DPC) [43], which takes the local densities, defined as Eq. (1), of the instances as their weights of becoming cluster centers.

$$\rho_i = \sum_{j \in I_S \setminus \{i\}} e^{-\left(\frac{d_{ij}}{d_c}\right)^2} \tag{1}$$

In Eq. (1), ρ_i is the local density of the ith instance. d_{ij} is the distance between the ith and jth instances, and d_c is a cutoff distance. To achieve the separably effects, the distance of the ith instances to other potential centers is also used and represented as Eq. (2). If ρ_i is the highest, $\delta_i = \max_j(d_{ij})$.

$$\delta_i = \min_{j:\rho_i < \rho_j} (d_{ij}) \tag{2}$$

With these two factors, the instances would be considered as density peaks if their densities and distances are large enough. It is noticeable that the density peaks often locate at the centers of clusters.

It is natural to take the density ρ_i and the distance δ_i as metrics to weight the importance degree of the ith instance to its corresponding data distribution. As a matter of fact, the instances within the core of dense distributions usually have higher densities than those of sparse distributions, and the instances along the boundaries between the classes or disjunct areas within a class have low densities and relative short distances to the cluster centers. To some extent, the data distributions can be approximately represented by the instances with high local densities, making learning models have better performance, if a sufficient number of the instances with high densities are chosen.

Based on the above assumption, we exploit the local density ρ and the distance δ to construct a sampling sequence of the instances within the majority classes. Specifically, we adopt the function $f(\rho,\delta) = \rho^2 * \delta$ to evaluate the importance degrees of the instances. Then, we construct a corresponding sequence of importance degrees of the instances in a descending order. With this sequence, we can pick those important instances as representative ones to undersample a new training data collection.

Another challenging problem is how many the representative instances of the majority classes should be chosen to train models. If the size of the representative instances is too small, the original data distributions cannot be approximated perfectly by the chosen instances, resulting in poor robustness of the learning models. On the contrary, if the size is too large, many unimportant instances will be mistaken as representative ones, leading to poor performance of the models.

To address the problem above, we adopt a progressive manner to select an appropriate number of the representative instances. Thus, the optimal size of the instances can be dynamically determined through an iterative process [44]. Specifically, we first pick a small size s_0 of the instances with the highest weights out from the sequence, and then train the learning models on them. Subsequently, a large size, e.g., $s_i = s_0 * a$ or $s_i = s_0 + a$ (a is a function of the size), of the instances with high weights are extracted from the sequence. If the performance of the models can be improved further, a larger size, s_i ($i = 1, 2, \ldots$), of the instances are needed. This iterative process is repeated, until the performance of the models cannot be improved further.

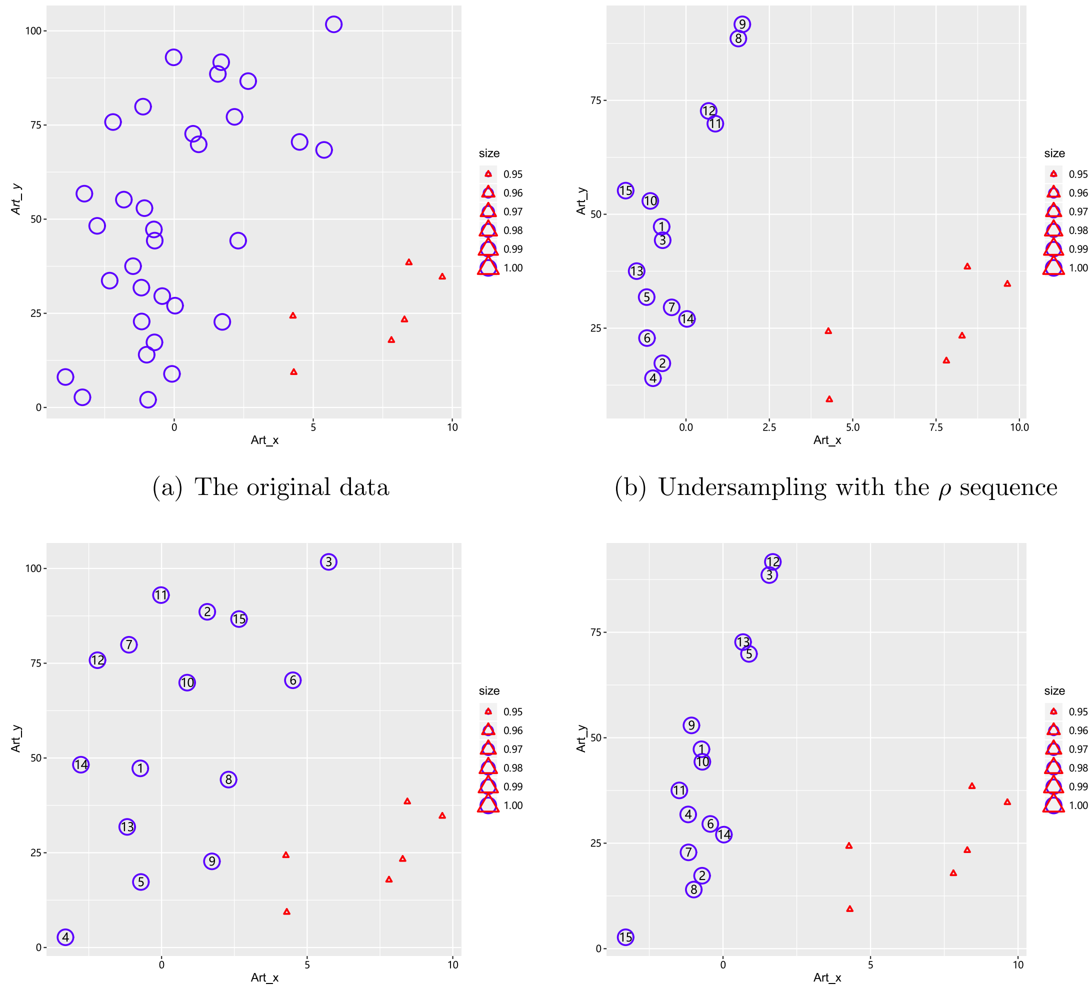
For the progressive sampling procedure, we have the following property.

Proposition 1. *The empirical distribution of the representative instances progressively extracted from the sequence generated by the function $f(\rho,\delta) = \rho^2 * \delta$ converges to the population distribution.*

Proof. Assume that a data collection consists of two classes, i.e., one is the majority and another is the minority. For each instance x_i of the majority class, we get its density ρ_i and distance δ_i. Let the function $f(\rho,\delta) = \rho^2 * \delta$ has k different values, and it partitions the instances of the majority class into k parts. We progressively sample each part with a weight of w_t ($t = 1, 2, \ldots, k$) shown as follows.

$$w_t = \frac{(\rho^2 * \delta)_t}{\sum_{t=1}^{k} (\rho^2 * \delta)_t} \tag{3}$$

w_t highlights the weight of the tth part of the majority class. Let X be the instances of the majority class and X_i ($i = 1, 2, \ldots, n$) be a sampling with n instances from the majority class with



(a) The original data (b) Undersampling with the ρ sequence

(c) Undersampling with the δ sequence (d) Undersampling with the $\rho^2 * \delta$ sequence

Fig. 2. Scatter plots of the top 15 representative instances, marked with blue circles, undersampled with different sequences from the majority class.

replacement according to w_t. We have $X \sim F(x)$, where $F(x)$ is the data distribution of the majority class. The empirical distribution function of X_i is defined as

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I(X_i \leq x) \tag{4}$$

where $I(\cdot)$ is an indicator function to determine whether $X_i \leq x$ or not. According to the strong law of large numbers and the Glivenko–Cantelli theorem [45], the following assertion holds.

$$P\{lim_{n\to N}\sup_{x\in R}|F(x) - F_n(x)| = 0\} = 1 \tag{5}$$

Here, N is the size of the majority class. From this assertion, one can observe that $F_n(x)$ converges to $F(x)$ by probability, that is, the data distribution of the progressive sampling is a reasonable estimation of the population distribution. ■

In extreme cases, each instance of the majority class has different value of the function $f(\rho, \delta) = \rho^2 * \delta$, then, the instance is extracted one by one from the sampling sequence and $F_n(x)$ converges to $F(x)$ when n→ N.

A roughly appropriate quantity of the representative instances can be obtained by the progressive sampling process. However, the more the iterative steps, the larger the quantity of the chosen instances. To alleviate this issue, we further exploit the relationship of the size to the performance to obtain the optimal size of the representative instances. Generally, the performance can be improved greatly along with the size of the representative instances at the beginning, and then enhanced gradually as the size increased. After the size reaches a point, the performance

Table 1
A toy synthetic dataset with imbalanced classes.

No	Art_x	Art_y	class	No	Art_x	Art_y	class
1	−0.94	2.06	0	20	−3.23	56.77	0
2	−3.3	2.7	0	21	5.39	68.39	0
3	−0.08	8.92	0	22	4.51	70.51	0
4	−3.91	8.09	0	23	0.88	69.88	0
5	−0.99	14.01	0	24	0.68	72.68	0
6	−0.71	17.29	0	25	2.17	77.17	0
7	1.73	22.73	0	26	−2.2	75.8	0
8	−1.17	22.83	0	27	−1.12	79.88	0
9	0.03	27.03	0	28	2.66	86.66	0
10	−0.43	29.57	0	29	1.57	88.57	0
11	−1.18	31.82	0	30	1.69	91.69	0
12	−2.32	33.68	0	31	−0.02	92.98	0
13	−1.48	37.52	0	32	5.74	101.74	0
14	2.3	44.3	0	33	4.3	9.3	1
15	−0.7	44.3	0	34	7.81	17.81	1
16	−0.73	47.27	0	35	8.28	23.28	1
17	−2.77	48.23	0	36	4.27	24.27	1
18	−1.07	52.93	0	37	9.64	34.64	1
19	−1.81	55.19	0	38	8.44	38.44	1

Table 2
A summary of the imbalanced datasets used in our experiments.

ID	Datsets	#feat.	#minority	#majority	#ratio
1	abalone208	8	26	804	30.92
2	cleverlanddata	12	68	235	3.45
3	ecoli1	8	77	259	3.36
4	ecoli2	8	52	284	5.46
5	ecoli3	8	35	301	8.6
6	ecoli4	8	20	315	15.75
7	ecoli014756	5	25	307	12.28
8	ecoli0345	3	20	180	9
9	ecoli01472356	4	29	307	10.59
10	glass0	5	70	144	20.57
11	glass0123456	10	51	163	3.20
12	glass1	10	76	138	1.82
13	glass2	10	50	579	11.59
14	glass4	10	13	201	15.46
15	hungerrindata	14	69	225	3.26
16	kddcup_buffer	14	30	2203	73.43
17	new_thyrodi1	6	35	180	5.14
18	page_blocks_134	9	28	444	15.86
19	pima	6	268	500	1.87
20	page_block0	11	559	4913	8.79
21	segment0	20	329	1979	6.02
22	shuttle25	10	49	3267	66.67
23	vehicle1	19	217	627	2.89
24	vehicle21	7	218	606	2.78
25	vehicle31	12	212	634	2.99
26	vowel	11	90	898	9.98
27	wisconsin	10	239	444	1.86
28	winequalityred4	12	53	1546	29.17
29	winequalityred86	12	18	285	15.86
30	winequalitywhite	12	25	1457	58.28
31	yeast035978	9	50	579	11.59
32	yeast1	9	429	1055	2.46
33	yeast17	8	30	429	14.3
34	yeast12897	9	30	917	30.57
35	yeast24	9	51	463	9.08.
36	yeast056794	9	49	458	9.35
37	yeast3	9	163	1320	8.1
38	yeast4	9	51	1433	28.09
39	yeast5	9	44	1440	32.73
40	yeast6	6	35	1449	4.14

may not be improved further and even be deteriorated. Based on this fact, we can extract a small size of the representative instances, as the performance is not improved greatly.

To better understand the idea, let us examine a toy example. Given a dataset D consists of 38 instances in two dimension spaces. They are grouped into two classes (see Table 1), where the majority class labeled with '0' contains 32 instances, while the minority class, labeled with '1 ', has 6 instances. The scatter plot of D is presented as Fig. 2(a).

Weighting the importance degrees of instances plays a key role in extracting representative instances from the data. For each instance (x_i, y_i) in the majority class of D, we calculate its density ρ_i, the distance δ_i, and their joint function $f(\rho, \delta) = \rho^2 * \delta$. Fig. 2 (b)-(d) show the scatter plots of the top 15 representative instances, marked with blue circles, extracted by our progressive sampling method from the sequences respectively. From Fig. 2, one can observe that the density ρ can effectively evaluate those important instances with respect to the classification problem, while the distance δ can preserve the distribution of the original data.

We take use of the relationship between the size of the chosen instances and the performance of an appropriate classification algorithm to determine the optimal number of the representative instances. Fig. 3 describes the relationship of the size of the selected instances to the area under the ROC curve (AUC) on the sampling sequence of $f(\rho, \delta) = \rho^2 * \delta$, where the horizontal means sampling size and the vertical means AUC of the learning model with a 5-fold cross-validation manner, respectively.

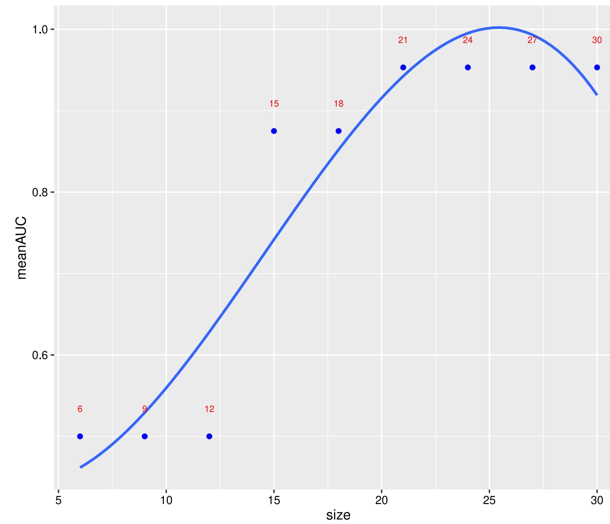


Fig. 3. The relationship between the size of selected instances and the mean AUC of the decision tree algorithm on the sampling sequence of $\rho^2 * \delta$.

The higher the mean AUC, the better the performance. The dots denote the size of the undersampling instances. From Fig. 3, we can conclude that the performance of the learning model did not approve any more on the sampling sequence after the size of the undersampling instances reached to 21.

Based on the analysis above, we propose a novel undersampling method called PUMD (Progressive Undersampling Method with Density). The detailed procedure of PUMD is presented in Algorithm 1.

Algorithm 1 Progressive undersampling method with density sequences.

Input: The training data $\mathbf{D} = \mathbf{M}_a \cup \mathbf{M}_i \in \mathcal{R}^{n \times p}$;
Output: The representative instances $\mathbf{R}$;
1: For $x \in \mathbf{M}_a$, calculate ρ and δ according to Eq. (1) and (2);
2: Obtain the sampling sequence of $f(\rho, \delta)$ in a descending order;
3: $s = \log(|\mathbf{M}_i|)$;
4: **Repeat**
5: $\mathbf{R}$=$\{x|x \in$ the stratified random sampling s instances of the sequences$\}$;
6: Get the classification performance pc on $\mathbf{R}$;
7: $s = s + \log(|n|)$;
8: **Until** $pc \geq threshold$ or $s \geq |\mathbf{M}_a|$
9: Reduce $\mathbf{R}$ through the relation of $|\mathbf{R}|$ and pc further;
10: Return $\mathbf{R}$

The proposed undersampling algorithm mainly consists of four stages, where the first one (i.e., step 1) is to calculate the density ρ and distance δ for each instance x of the majority class $\mathbf{M}_a$. Then the sampling sequence of $f(\rho, \delta) = \rho^2 * \delta$ is derived in terms of ρ and δ. The third stage (from step 3 to step 8) aims to progressively extract the representative instances $\mathbf{R}$ from the sampling sequence. Note that at each iteration, the size s of the sampling instances increases $\log(|n|)$, until the performance of the classification model on $\mathbf{R}$ reaches a given threshold. After the progressively undersampling process, the representative instances $\mathbf{R}$ may contain some unimportant instances. Therefore, the final stage (step 9) is to remove those not so important instances from $\mathbf{R}$.

Let n be the number of instances in $\mathbf{D}$. The computation costs of the first and the second stages are $O(n^2)$ and $O(n \log n)$, respectively. The computation cost to get the classification performance

Table 3
The confusion matrix of binary classification.

		Prediction	
		Positive	Negative
GroundTruth	Positive	*TP*	*FN*
	Negative	*FP*	*TN*

Table 4
The mean precision of the classifier with the undersampling models.

Dataset	Original	RUS	NCL	OSS	CNN	ENN	CBU	PUMD
1. abalone20_8	0.3	0.848	0.5	0.667	0.4	0.6	0.689	**0.916**
2. cleverlanddata	0.167	0.413	0.348	0.314	0.271	0.28	0.528	**0.769**
3. ecoli1	0.775	0.832	0.83	0.782	0.769	0.765	0.767	**0.898**
4. ecoli2	0.8	0.861	0.815	0.755	0.76	0.86	0.861	**0.909**
5. ecoli3	0.655	0.765	0.547	0.576	0.699	0.65	0.758	**0.886**
6. ecoli4	0.727	0.788	0.758	0.712	0.818	0.727	0.879	**0.909**
7. ecoli0_147vs56	0.841	0.78	0.841	0.833	0.795	0.841	0.803	**0.909**
8. ecoli034_5	0.833	0.803	0.818	0.833	0.864	0.818	0.803	**0.909**
9. ecoli01472356	0.727	0.765	0.758	0.773	0.712	0.742	0.75	**0.909**
10. glass0	0.513	0.671	0.604	0.595	0.513	0.593	0.673	**0.83**
11. glass0123456	0.788	0.599	0.82	0.814	0.829	0.802	0.828	**0.894**
12. glass1	0.579	0.715	0.688	**0.735**	0.711	0.687	0.644	0.694
13. glass2	0.667	0.8	0.333	0.25	0.57	0.5	0.712	**0.900**
14. glass4	0.813	**0.909**	0.771	0.778	0.816	0.786	0.85	**0.909**
15. hungerrindata	0.547	0.751	0.611	0.59	0.526	0.545	0.632	**0.885**
16. kddcup_buffer	0.151	0.886	0.818	0.795	0.796	0.808	0.886	**0.909**
17. new_thyrodi1	0.78	0.782	0.818	0.805	**0.859**	0.856	0.805	0.818
18. page_blocks_134	0.879	0.833	0.879	0.878	**0.909**	0.818	0.864	**0.909**
19. pima	0.621	0.673	0.679	0.718	0.631	0.664	0.676	**0.859**
20. page_block0	0.797	0.853	0.817	0.807	0.798	0.834	0.843	**0.907**
21. segment-0	0.873	0.872	0.879	0.882	0.892	0.879	0.89	**0.904**
22. shuttle2–5	**0.909**	**0.909**	**0.909**	**0.909**	**0.909**	**0.909**	0.894	**0.909**
23. vehicle1	0.506	0.695	0.602	0.64	0.513	0.518	0.665	**0.731**
24. vehicle2–1	0.738	0.804	0.73	0.72	0.753	0.733	0.827	**0.893**
25. vehicle3–1	0.54	0.704	0.655	0.636	0.578	0.578	0.702	**0.901**
26. vowel	0.815	0.847	0.812	0.856	0.82	0.808	0.891	**0.893**
27. wisconsin	0.856	0.842	0.865	0.851	0.866	0.843	0.848	**0.909**
28. winequal-ityred4	0.035	0.622	0	0	0	0	0.63	**0.909**
29. winequalityred86	0	0.652	0.537	0.333	0.333	0.194	0.652	**0.818**
30. winequalitywhite	0.25	0.5	0.333	0.6	0.333	0.167	0.689	**0.909**
31. yeast035978	0.704	0.695	0.771	0.722	0.812	0.685	0.614	**0.909**
32. yeast1	0.556	0.622	0.626	0.664	0.556	0.624	0.65	**0.861**
33. yeast17	0.667	0.788	0.690	0.643	0.643	0.685	0.75	**0.910**
34. yeast12897	0.5	0.757	0.714	0.571	0.7	0.5	0.706	**0.909**
35. yeast24	0.763	0.839	0.822	0.830	0.81	0.827	0.87	**0.919**
36. yeast056794	0.685	0.791	0.63	0.698	0.644	0.743	0.754	**0.909**
37. yeast3	0.632	0.875	0.773	0.771	0.717	0.746	0.866	**0.904**
38. yeast4	0.5	0.75	0.569	0.714	0.667	0.333	0.721	**0.876**
39. yeast5	0.735	0.858	0.789	0.774	0.754	0.755	0.849	**0.909**
40. yeast6	0.583	0.739	0.661	0.542	0.787	0.611	0.759	**0.909**

on **R** depends on the chosen classification algorithm, taking the decision tree algorithm as an example, whose computation time is $O(pn \log n)$ (p is the number of features), the time complexity of progressively sampling stage can be summarized as $O(pn \cdot [\frac{n}{log^n}])$. Hence, the total time complexity of PUMD is $O(pn^2)$.

4. Experiments

To validate the effectiveness of the proposed method, we carried out a comprehensive experiment on 40 two-class imbalanced datasets from the KEEL dataset repository [26]. These datasets were frequently used to verify the performance of undersampling models in the literature. Their imbalance ratios are varied from 1.82 to 73.43. Table 2 briefly summaries the descriptions of the experimental datasets, where *#feat.*, *#minority*, *#majority* and *ratio* denote the numbers of features, instances in the minority class and the majority class, and the imbalance ratios, respectively.

In our experiments, six typical and popular undersampling algorithms, including random undersampler (RUS), one side selection (OSS) [30], neighborhood cleaning rule (NCL) [31], clustering-based undersampler (CBU) [23], CNN [34] and ENN [35], were

Table 5
The mean recall of the classifier with the undersampling models.

Dataset	Original	RUS	NCL	OSS	CNN	ENN	CBU	PUMD
1. abalone20_8	0.06	0.773	0.167	0.182	0.136	0.191	0.697	**0.818**
2. cleverlanddata	0.018	0.457	0.362	0.227	0.145	0.212	0.498	**0.831**
3. ecoli1	0.719	0.815	0.794	0.776	0.768	0.734	0.826	**0.89**
4. ecoli2	0.652	0.624	0.664	0.661	0.624	0.612	0.755	**0.839**
5. ecoli3	0.495	0.72	0.561	0.432	0.508	0.58	0.765	**0.864**
6. ecoli4	0.667	0.682	0.636	0.636	0.682	0.636	0.818	**0.909**
7. ecoli0147vs56	0.667	0.788	0.697	0.712	0.727	0.652	**0.833**	**0.833**
8. ecoli034_5	0.682	0.773	0.727	0.727	0.682	0.682	**0.818**	0.773
9. ecoli01472356	0.5	0.788	0.545	0.545	0.5	0.516	0.788	**0.879**
10. glass0	0.429	0.779	0.61	0.571	0.455	0.572	0.779	**0.831**
11. glass0123456	0.745	0.571	0.733	0.682	0.727	0.712	0.764	**0.873**
12. glass1	0.51	**0.75**	0.681	0.639	0.606	0.534	0.704	0.731
13. glass2	0.1	0.818	0.191	0.145	0.145	0.165	0.773	**0.836**
14. glass4	0.545	0.864	0.591	0.591	0.773	0.455	0.773	**0.894**
15. hungerrindata	0.466	0.649	0.478	0.424	0.418	0.524	0.621	**0.829**
16. kddcup_buffer	0.667	0.849	0.606	0.667	0.636	0.636	0.818	**0.909**
17. new_thyrodi1	0.74	0.796	0.705	0.78	0.864	0.667	0.856	**0.871**
18. page_blocks_134	0.742	**0.909**	0.773	0.848	0.848	0.788	**0.909**	**0.909**
19. pima	0.512	0.628	0.583	0.607	0.516	0.539	0.627	**0.862**
20. page_block0	0.773	0.862	0.759	0.753	0.73	0.748	0.844	**0.905**
21. segment0	0.881	0.876	0.87	0.873	0.857	0.878	**0.898**	0.895
22. shuttle2–5	**0.909**	0.891	**0.909**	**0.909**	**0.909**	**0.909**	**0.909**	**0.909**
23. vehicle1	0.276	0.69	0.599	0.581	0.411	0.454	**0.721**	0.718
24. vehicle2–1	0.7	0.85	0.704	0.697	0.73	0.718	0.838	**0.897**
25. vehicle3–1	0.429	0.743	0.575	0.566	0.438	0.464	0.729	**0.905**
26. vowel	0.829	0.889	0.788	0.808	0.828	0.808	0.889	**0.909**
27. wisconsin	0.831	0.849	0.844	0.875	0.89	0.875	0.867	**0.902**
28. winequalityred4	0.287	0.6	0	0	0	0	0.588	**0.867**
29. winequalityred86	0	0.772	0.318	0.227	0.136	0.136	0.727	**0.909**
30. winequalitywhite	0.03	0.111	0.09	0.106	0.04	0.03	0.636	**0.909**
31. yeast035978	0.182	0.691	0.182	0.182	0.182	0.218	0.655	**0.891**
32. yeast1	0.397	0.642	0.504	0.518	0.384	0.409	0.754	**0.869**
33. yeast17	0.212	0.788	0.273	0.273	0.152	0.242	0.696	**0.818**
34. yeast12897	0.121	0.636	0.182	0.182	0.121	0.121	0.636	**0.878**
35. yeast24	0.673	0.873	0.764	0.691	0.667	0.682	0.891	**0.919**
36. yeast056794	0.267	0.697	0.394	0.458	0.279	0.376	0.715	**0.849**
37. yeast3	0.781	0.826	0.742	0.752	0.708	0.707	0.87	**0.886**
38. yeast4	0.178	0.709	0.233	0.171	0.133	0.2	0.712	**0.839**
39. yeast5	0.688	0.891	0.655	0.677	0.65	0.646	0.891	**0.909**
40. yeast6	0.447	0.747	0.378	0.292	0.278	0.446	0.78	**0.864**

used as baselines to make a comparison with the proposed method. It should be pointed out that RUS requires to predetermine the undersampling size in advance, while CBU undersamples the same size of the majority class instances to the number of the minority ones.

Since the misclassification cost of the minority class is often higher than that of the majority class in the imbalanced issues, the evaluation metrics which are sensitive to the minority class are more preferable. Following this routine, we also adopted four evaluation metrics, i.e., precision, recall, the area under the ROC curve (AUC) and F1 score, to validate the classification performance of the chosen algorithm on the new datasets resampled by the undersampling models in our experiments.

Assume Table 3 be the confusion matrix of binary classification. Precision refers to the percentage of positive instances correctly predicted to the predicted results of the classification algorithm, i.e.,

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Recall is the percentage of positive instances correctly classified to the ground truth. Generally, a perfect model can not only capture all positive instances, making $Recall = 1$, but also predict them correctly, making $Precision = 1$.

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

Table 6
The mean AUC of the classifier with the undersampling models.

Dataset	Original	RUS	NCL	OSS	CNN	ENN	CBU	PUMD
1. abalone20_8	0.483	0.803	0.536	0.545	0.521	0.5	0.682	**0.864**
2. cleverlanddata	0.458	0.411	0.507	0.483	0.449	0.484	0.495	**0.722**
3. ecoli1	0.793	0.819	0.837	0.808	0.796	0.799	0.783	**0.892**
4. ecoli2	0.771	0.763	0.778	0.763	0.75	0.756	0.806	**0.874**
5. ecoli3	0.688	0.712	0.709	0.627	0.672	0.691	0.719	**0.879**
6. ecoli4	0.784	0.704	0.767	0.755	0.791	0.767	0.841	**0.909**
7. ecoli0147vs56	0.785	0.773	0.80	0.806	0.812	0.776	0.803	**0.871**
8. ecoli034_5	0.79	0.773	0.813	0.803	0.789	0.79	0.796	**0.841**
9. ecoli01472356	0.696	0.757	0.719	0.718	0.694	0.704	0.757	**0.894**
10. glass0	0.59	0.695	0.667	0.612	0.598	0.645	0.701	**0.75**
11. glass0123456	0.805	0.656	0.807	0.775	0.793	0.794	0.793	**0.887**
12. glass1	0.623	0.711	0.717	**0.718**	0.704	0.674	0.643	0.651
13. glass2	0.5	0.795	0.497	0.472	0.478	0.477	0.682	**0.773**
14. glass4	0.725	**0.886**	0.745	0.744	0.823	0.68	0.795	**0.886**
15. hungerrindata	0.646	0.694	0.637	0.617	0.611	0.659	0.62	**0.851**
16. kddcup_buffer	0.764	0.864	0.757	0.784	0.771	0.772	0.848	**0.909**
17. new_thyrodi1	0.78	0.787	0.794	0.819	**0.864**	0.783	0.814	0.818
18. page_blocks_134	0.825	0.864	0.84	0.875	0.879	0.847	0.878	**0.909**
19. pima	0.648	0.66	0.672	0.701	0.653	0.667	0.656	**0.729**
20. page_block0	0.835	0.856	0.829	0.825	0.813	0.824	0.842	**0.907**
21. segment0	0.892	0.889	0.887	0.888	0.881	0.891	0.89	**0.90**
22. shuttle2–5	**0.909**	**0.909**	**0.909**	**0.909**	**0.909**	**0.909**	0.894	**0.909**
23. vehicle1	0.556	0.692	0.686	0.688	0.603	0.619	0.679	**0.753**
24. vehicle2–1	0.775	0.819	0.774	0.769	0.791	0.783	0.832	**0.878**
25. vehicle3–1	0.621	0.709	0.693	0.686	0.631	0.638	0.709	**0.893**
26. vowel	0.865	0.864	0.842	0.853	0.854	0.853	0.889	**0.904**
27. wisconsin	0.852	0.86	0.864	0.737	0.786	0.872	0.856	**0.905**
28. winequalityred4	0.52	0.60	0.454	0.454	0.45	0.45	0.617	**0.889**
29. winequalityred86	0.443	0.636	0.609	0.563	0.519	0.518	0.636	**0.882**
30. winequalitywhite	0.469	0.5	0.49	0.507	0.476	0.469	0.681	**0.909**
31. yeast035978	0.542	0.682	0.553	0.533	0.544	0.561	0.618	**0.843**
32. yeast1	0.602	0.624	0.645	0.662	0.596	0.618	**0.681**	0.661
33. yeast17	0.588	0.773	0.587	0.585	0.528	0.573	0.682	**0.818**
34. yeast12897	0.513	0.697	0.544	0.545	0.615	0.514	0.677	**0.888**
35. yeast24	0.783	0.847	0.831	0.794	0.78	0.792	0.876	**0.892**
36. yeast056794	0.563	0.735	0.665	0.662	0.591	0.626	0.73	**0.879**
37. yeast3	0.823	0.851	0.816	0.821	0.794	0.798	0.866	**0.893**
38. yeast4	0.419	0.727	0.53	0.513	0.4	0.348	0.721	**0.861**
39. yeast5	0.796	0.866	0.780	0.789	0.773	0.775	0.859	**0.909**
40. yeast6	0.621	0.715	0.604	0.519	0.582	0.638	0.754	**0.886**

Table 7
The mean F1 score of the classifier with the undersampling models.

Dataset	Original	RUS	NCL	OSS	CNN	ENN	CBU	PUMD
1. abalone20_8	0.3	0.774	0.473	0.527	0.556	0.375	0.675	**0.855**
2. cleverlanddata	0.143	0.411	0.507	0.483	0.203	0.250	0.504	**0.796**
3. ecoli1	0.734	0.818	0.807	0.767	0.762	0.746	0.791	**0.893**
4. ecoli2	0.704	0.714	0.721	0.685	0.656	0.700	0.795	**0.868**
5. ecoli3	0.539	0.702	0.537	0.462	0.537	0.530	0.724	**0.866**
6. ecoli4	0.695	0.694	0.661	0.641	0.727	0.667	0.830	**0.909**
7. ecoli0147vs56	0.717	0.772	0.738	0.752	0.735	0.703	0.798	**0.861**
8. ecoli034_5	0.724	0.767	0.758	0.755	0.727	0.727	0.797	**0.818**
9. ecoli01472356	0.558	0.749	0.597	0.603	0.567	0.588	0.746	**0.891**
10. glass0	0.457	0.715	0.597	0.574	0.465	0.569	0.718	**0.825**
11. glass0123456	0.754	0.566	0.765	0.728	0.758	0.744	0.781	**0.881**
12. glass1	0.526	**0.719**	0.669	0.672	0.633	0.589	0.659	0.692
13. glass2	0.445	**0.833**	0.5	0.333	0.333	0.335	0.703	0.767
14. glass4	0.75	0.879	0.767	0.833	0.813	0.738	0.833	**0.9**
15. hungerrindata	0.522	0.675	0.517	0.480	0.447	0.524	0.618	**0.853**
16. kddcup_buffer	0.239	0.859	0.682	0.705	0.746	0.739	0.842	**0.909**
17. new_thyrodi1	0.725	0.785	0.723	0.764	**0.848**	0.731	0.819	0.840
18. page_blocks_134	0.794	0.865	0.803	0.861	0.873	0.88	0.883	**0.909**
19. pima	0.558	0.647	0.626	0.656	0.565	0.593	0.646	**0.859**
20. page_block0	0.784	0.857	0.787	0.778	0.762	0.786	0.843	**0.906**
21. segment0	0.877	0.877	0.874	0.877	0.873	0.878	0.894	**0.899**
22. shuttle2–5	0.909	0.910	0.909	**0.911**	0.907	0.909	0.900	0.909
23. vehicle1	0.338	0.685	0.599	0.603	0.452	0.475	0.690	**0.72**
24. vehicle2–1	0.713	0.824	0.714	0.706	0.739	0.723	0.832	**0.894**
25. vehicle3–1	0.473	0.716	0.609	0.597	0.495	0.503	0.713	**0.903**
26. vowel	0.813	0.865	0.787	0.829	0.821	0.802	0.889	**0.9**
27. wisconsin	0.842	0.845	0.853	0.862	0.877	0.857	0.856	**0.905**
28. winequalityred4	0.681	0.599	0	0	0	0	0.596	**0.886**
29. winequalityred86	0	0.636	0.557	0.5	0.417	0.325	0.648	**0.854**
30. winequalitywhite	0.2	0.5	0.444	0.417	0.333	0.167	0.706	**0.909**
31. yeast035978	0.363	0.678	0.366	0.375	0.375	0.412	0.618	**0.867**
32. yeast1	0.463	0.628	0.554	0.576	0.451	0.49	0.697	**0.864**
33. yeast17	0.494	0.773	0.505	0.448	0.152	0.446	0.671	**0.806**
34. yeast12897	0.38	0.673	0.4	0.433	0.38	0.38	0.656	**0.877**
35. yeast24	0.695	0.85	0.787	0.736	0.715	0.736	0.876	**0.908**
36. yeast056794	0.452	0.72	0.512	0.434	0.435	0.429	0.736	**0.891**
37. yeast3	0.694	0.848	0.754	0.757	0.705	0.720	0.867	**0.894**
38. yeast4	0.244	0.718	0.305	0.272	0.222	0.250	0.704	**0.854**
39. yeast5	0.682	0.871	0.678	0.713	0.658	0.683	0.866	**0.91**
40. yeast6	0.485	0.736	0.418	0.368	0.391	0.473	0.738	**0.883**

The area under the receiver operating characteristic curve (AUC) is commonly used to measure the performance of classification, while treating the performance of majority classes as important as the performance of minority classes.

$$AUC = \frac{1 + Recall - FPR}{2} \tag{8}$$

where *FPR* is the percentage of negative instances misclassified in the active negative instances.

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

F1 score is the harmonic average of precision and recall, which reflects the predictive capability of classifiers for positive instances, i.e.,

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{10}$$

5. Experimental results and analysis

5.1. Classification performance with the undersampling algorithms

In our experiments, we compared the proposed method to six popular undersampling algorithms at the aspect of classification performance. Specifically, we performed the undersampling algorithms on the experimental datasets to generate new and balanced training datasets, and then applied the appropriate classification models to get classification performance, notably the undersampling datasets extracted from the identical imbalanced data execute the same classification algorithm. The experiments were conducted in a 10-fold cross-validation manner, and their mean values were taken as the final performance.

Table 4 presents the mean precision of the classifiers on the experimental datasets with the undersampling models, where the best one for each dataset is highlighted in boldface. According to the experimental results in Table 4, one may observe that the proposed undersampling method, PUMD, had considerably better performance than the comparison undersampling techniques. For example, PUMD achieved the best precision on 38 out of 40 datasets. On the *glass1* and *new_thyrodi1* datasets, the mean precision were 0.694 and 0.818, respectively, which were slightly lower than those of OSS and CNN respectively. Even so, PUMD still improved the performance of the classification model. To show the difference of the undersampling algorithms clearly, we present the difference of mean precision as a chart (see Fig. 4). As shown in Fig. 4, we know that PUMD was significantly superior to others. Although OSS achieved the best performance on *glass1*, it was the worst one on *yeast6*.

On the metrics of recall, AUC and F1 score, PUMD exhibited similar properties. Tables 5–7 report the mean performance of the chosen classification model with the undersampling data generated by the undersampling algorithms with respect to recall, AUC and F1 score, respectively. As shown in Tables 5–7, we note that PUMD also had comparable performance on the metrics of recall, AUC and F1 score. For instance, PUMD had the highest recall values on 36 out of 40 datasets. It also outperformed the popular

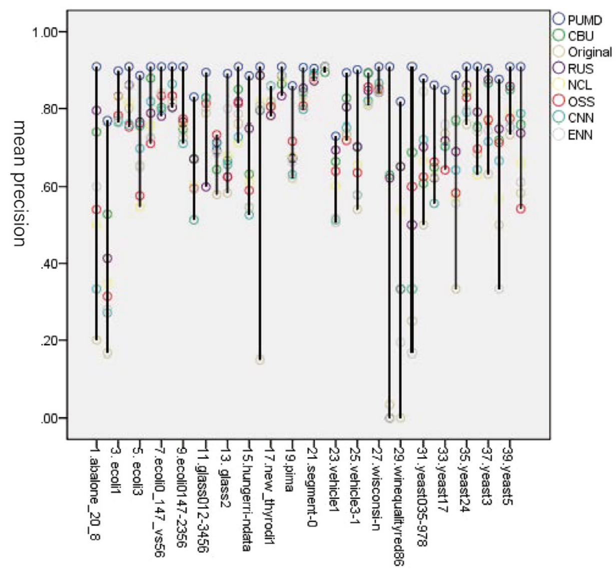


Fig. 4. The difference of mean precision with the undersampling algorithms.

undersampling algorithms on 37 out of 40 datasets with respect to the AUC measure and 36 out of 40 datasets with respect to the F1 score measure. For the recall measure, CBU achieved the best performance on *ecoli0345*, *segment0* and *vehicle1*. However, the values of AUC and F1 score corresponding to CBU were not the best ones.

5.2. The sizes of representative instances

As mentioned above, the size of representative instances sampled by an undersampling method is another important aspect which should be taken into account. Generally speaking, a small sampling size of representative instances is more preferable for an undersampling algorithm. To verify this assumption, we offered the sizes of representative instances sampled by the undersampling algorithms in Table 8, where *NumofMin* denotes the number of instances in the minority class. From the experimental results in Table 8, one may note that PUMD tends to choose a small size of representative instances, in comparing to NCL, OSS, CNN and ENN. Especially, it sampled very few representative instances, whose sizes were even less than the minority ones in some cases, such as *cleverlanddata*, *ecoli1*, *glass0*, *pima* and *wisconsin*. For example, PUMD only sampled 73 representative instances of the majority class from *vehicle1*, which are far less than the instances of the minority class. It should be pointed out that the size of instances sampled by RUS and CBU was the same as the size of the minority instances, while the sizes of chosen instances sampled from the majority classes by NCL, OSS, CNN and ENN were much larger than the minority ones.

Summarily, from the experimental results in Tables 4–8, one may conclude that:

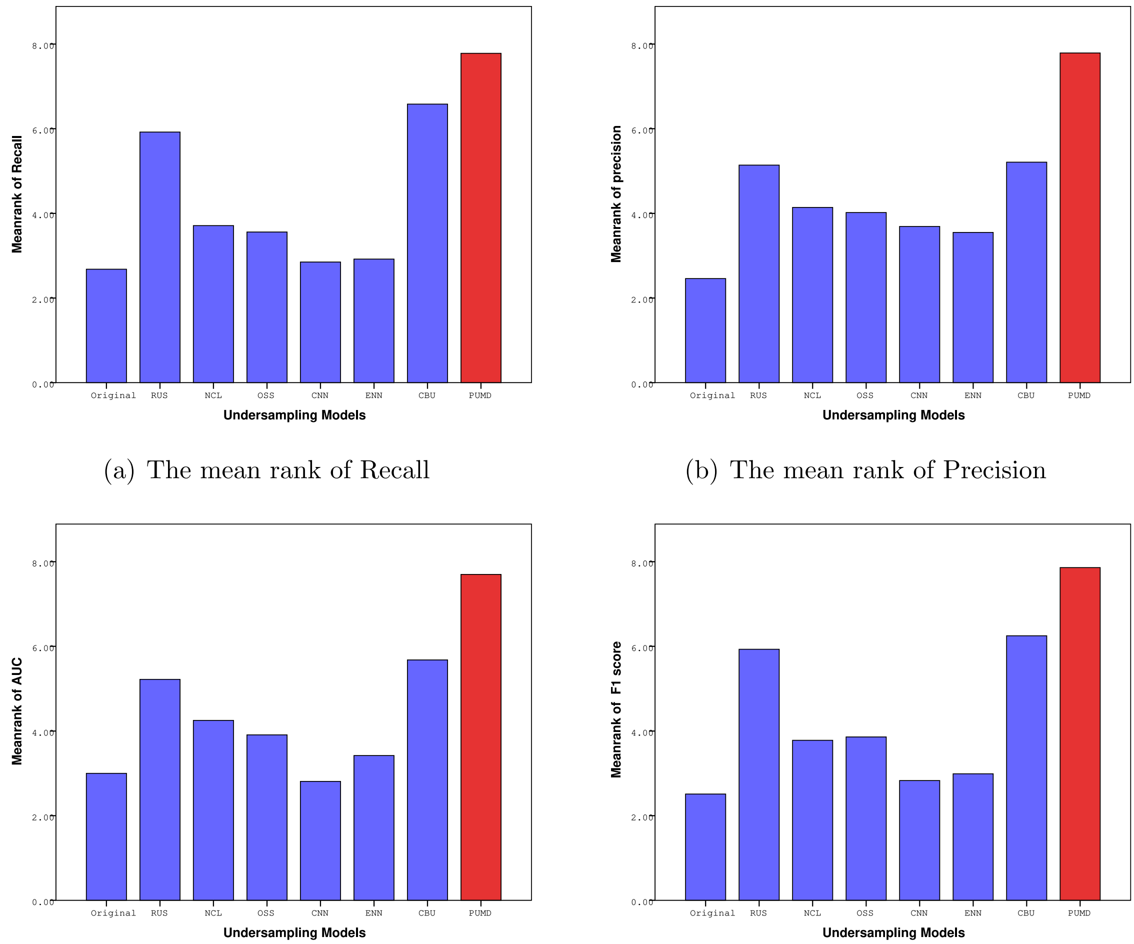
- The NCL, OSS, CNN and ENN undersampling algorithms remove those noisy or boundary instances from the majority class, which would be beneficial to the learning models for imbalanced data. However, a high imbalance ratio of the majority class to the minority class in under-sample still deteriorates learning performance.
- Contrastively, RUS and CBU extract the representative instances of the majority class with the same size to the minority class, making the learning performance comparable in general. Even so, the size of sampled instances does not concerns the diversity of data. As a result, they often fail in some imbalanced learning scenarios.
- The proposed method can automatically determine the number of representative instances of the majority class according to the inherent property of data, making it more appealing in real-world applications. Indeed, the experimental results above have shown this fact; that is, for different learning tasks, the sizes of representative instances of the majority class vary around the number of minority instances, even when the original size of majority class is very huge. From this point of view, the computational cost of PUMD should be far less than $O(pn^2)$, even in larger datasets, it is acceptable.

Table 8
The sizes of representative instances of the majority class chosen by the undersampling algorithms.

Dataset	NumofMin	RUS	NCL	OSS	CNN	ENN	CBU	PUMD
1. abalone20_8	26	26	600	652	732	152	26	67
2. cleverlanddata	68	68	156	187	227	233	68	38
3. ecoli1	77	77	235	175	139	255	77	57
4. ecoli2	52	52	270	190	247	283	52	52
5. ecoli3	35	35	282	159	113	297	35	65
6. ecoli4	20	20	306	214	184	315	20	30
7. ecoli0147vs56	25	24	294	201	286	306	25	20
8. ecoli0345	20	20	170	59	108	179	20	15
9. ecoli01472356	29	29	285	185	222	306	29	44
10. glass0	70	70	123	92	137	135	70	25
11. glass0123456	51	51	156	129	92	162	51	66
12. glass1	76	76	111	113	137	134	76	56
13. glass2	50	50	365	402	414	421	50	36
14. glass4	13	13	175	58	50	192	13	20
15. hungerrindata	69	69	184	194	218	219	69	49
16. kddcup_buffer	30	30	2199	489	492	2202	30	25
17. new_thyrodi1	35	35	180	107	99	180	35	45
18. page_blocks134	28	28	406	110	124	422	28	28
19. pima	268	268	357	391	500	481	268	43
20. page_block0	559	559	4713	4198	4287	4888	559	629
21. segment0	329	329	1900	1733	1491	1976	329	369
22. shuttle2–5	49	49	3267	1946	2040	3267	49	39
23. vehicle1	217	217	497	513	627	601	217	307
24. vehicle2–1	218	218	598	594	591	621	218	73
25. vehicle3–1	212	211	487	524	629	618	212	67
26. vowel	90	90	898	422	272	898	90	190
27. wisconsin	239	239	430	47	47	436	239	79
28. winequalityred4	53	53	1409	1485	1518	1546	53	58
29. winequalityred86	18	18	593	595	612	638	18	48
30. winequalitywhite	25	25	1392	1383	1328	1457	25	20
31. yeast035978	50	50	405	398	421	431	50	60
32. yeast1	429	429	793	862	1052	1016	429	189
33. yeast17	30	30	371	374	400	409	30	39
34. yeast12897	30	30	847	851	848	901	30	27
35. yeast24	51	51	403	401	413	441	51	28
36. yeast056794	49	49	419	417	438	454	49	25
37. yeast3	163	163	1241	1152	1185	1310	163	83
38. yeast4	51	51	1370	1092	1199	1430	51	126
39. yeast5	44	44	1425	839	632	1437	44	34
40. yeast6	35	35	1303	1174	1139	1444	35	140

5.3. Statistical test

To validate the fact that the performance of PUMD is significantly superior to those of the popular undersampling algorithms, the Friedman test [46] was performed further in our experiments. Before diving into the statistical test, we first derived the rank orders of the undersampling algorithms on the experimental datasets with respect to the metrics, i.e., precision, recall, AUC and F1 score. Specifically, for each undersampling algorithm, a rank value varied from one to eight was assigned according to its performance (see Tables 4–7) on each dataset individually. For the metrics of precision, recall, AUC and F1 score, the higher the rank, the better the performance. Fig. 5 records the mean rank



(a) The mean rank of Recall

(b) The mean rank of Precision

(c) The mean rank of AUC

(d) The mean rank of F1 Score

Fig. 5. The mean rank bars of the undersampling models on the metrics of recall, precision, AUC and F1 score.

values of the undersampling algorithms. As described in Fig. 5, PUMD also outperformed the popular undersampling algorithms from the perspective of the rank order.

Based on the rank orders, we further took a statistical test by using the Wilcoxon paired test [47] to verify the superiority of PUMD. Specifically, let PUMD be the control base. The hypothesis of the differences of the mean ranks of PUMD to the popular undersampling algorithms was tested at a significance level of α = 0.05, and the results were presented in Table 9. According to the results in Table 9, one can observe that all the null hypotheses were rejected for the comparison between PUMD and the others. The obtained p-value is low, which indicates that PUMD was superior to the popular undersampling algorithms.

6. Conclusion

Undersampling is an effective technique to address learning problems for imbalanced data. However, the traditional undersampling methods suffer from two essential problems; that is, which and how many instances should be extracted when undersampling. The existing undersampling methods usually keep the same size of majority classes and minority classes, which in many cases is not recommendable.

In this article, we proposed a novel undersampling method called PUMD (Progressive Undersampling Method with Density) to deal with the two aspects mentioned above. Our algorithm exploits a sequence of density peaks to progressively extract instances from the majority classes of imbalanced data. Specifically, two factors, i.e., the density ρ and the distance δ, are exploited to build a joint function $f(\rho, \delta)$ to evaluate importance degrees of the instances of majority classes. After that, a sampling sequence of majority class instances is generated according to the importance degrees and the representative instances are progressively extracted from it, while the optimal undersampling size is automatically determined.

The proposed algorithm can quickly detect the instances of majority classes that are core members of data distributions and automatically determine the optimal undersampling size, which effectively downsizes datasets and eliminates the instances of majority classes that are not important to classification tasks. Empirical results and statistical hypothesis tests show that PUMD outperforms other classical undersampling algorithms, whether in classification metric scores or sampling sizes. All of these characteristics make PUMD particularly suitable for large-scale data or even streaming data.

The proposed method provides a direction for future research to extend the progressive strategies from undersampling models to oversampling ones, to determine the optimal size of oversampling. Additionally, more elaboration will be carried out for large-scale datasets and streaming datasets, especially how to set sampling-stop rules to find the optimal sample size.

Table 9
The Wilcoxon test of PUMD to the popular undersampling algorithms.

Recall		Precision		AUC		F1 Score	
Sampler	*p*-value	Sampler	*p*-value	Sampler	p-value	Sampler	*p*-value
Original	2.73E−08	Original	2.73E−08	Original	2.73E−08	Original	2.73E−08
RUS	5.76E−08	RUS	2.95E−08	RUS	8.01E−08	RUS	2.51E−08
NCL	2.73E−08	NCL	4.02E−08	NCL	5.48E−08	NCL	2.73E−08
OSS	2.73E−08	OSS	4.03E−08	OSS	7.44E−08	OSS	2.00E−08
CNN	2.73E−08	CNN	4.35E−08	CNN	5.48E−08	CNN	2.73E−12
ENN	2.73E−08	ENN	4.03E−08	ENN	3.19E−08	ENN	2.73E−08
CBU	5.76E−08	CBU	1.85E−08	CBU	2.92E−08	CBU	1.85E−08

CRediT authorship contribution statement

Xiaoying Xie: Writing - original draft. **Huawen Liu:** Formal analysis. **Shouzhen Zeng:** Writing - review & editing. **Lingbin Lin:** Investigation. **Wen Li:** Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is funded by the National Social Science Foundation of China [grant number 17BTJ028].

References

[1] S. Fotouhi, S. Asadi, M.W. Kattan, A comprehensive data level analysis for cancer diagnosis on imbalanced data, J. Biomed. Inform. 90 (2019) 103089.
[2] A. Zakaryazad, E. Duman, A profit-driven Artificial Neural Network (ANN) with applications to fraud detection and direct marketing, Neurocomputing 175 (JAN.29PT.A) (2016) 121–131.
[3] D.D. Roy, D. Shin, Network intrusion detection in smart grids for imbalanced attack types using machine learning models, in: 2019 International Conference on Information and Communication Technology Convergence, ICTC, 2019.
[4] S. Gupta, A. Gupta, Handling class overlapping to detect noisy instances in classification, Knowl. Eng. Rev. 33 (2018) e8.
[5] M. Hlosta, Z. Zdrahal, J. Zendulka, Are we meeting a deadline? classification goal achievement in time in the presence of imbalanced data, Knowl.-Based Syst. 160 (2018) 278–295.
[6] T. Jo, N. Japkowicz, Class imbalances versus small disjuncts, ACM SIGKDD Explorations Newsl. 6 (1) (2004) 40–49.
[7] M.R. Smith, T. Martinez, C. Giraud-Carrier, An instance level analysis of data complexity, Mach. Learn. 95 (2) (2014) 225–256.
[8] H. Guo, Y. Li, J. Shang, M. Gu, Y. Huang, B. Gong, Learning from class-imbalanced data: review of methods and applications, Expert Syst. Appl. 73 (2017) 220–239.
[9] R. O'Brien, H. Ishwaran, A random forests quantile classifier for class imbalanced data, Pattern Recognit. 90 (2019) 232–249.
[10] C. Zhang, J. Bi, S. Xu, E. Ramentol, G. Fan, B. Qiao, H. Fujita, Multi-Imbalance: an open-source software for multi-class imbalance learning, Knowl.-Based Syst. 174 (2019) 137–143.
[11] X. Gao, Z. Chen, S. Tang, Y. Zhang, J. Li, Adaptive weighted imbalance learning with application to abnormal activity recognition, Neurocomputing 173 (JAN.15PT.3) (2016) 1927–1935.
[12] G. Ming, H. Xia, C. Harris, Construction of neurofuzzy models for imbalanced data classification, IEEE Trans. Fuzzy Syst. 22 (6) (2014) 1472–1488.
[13] S. Zhang, Cost-sensitive KNN classification, Neurocomputing 391 (2020) 234–242.
[14] M.L. Wong, K. Seng, P.K. Wong, Cost-sensitive ensemble of stacked denoising autoencoders for class imbalance problems in business domain, Expert Syst. Appl. 141 (2020) 112918.
[15] M. Li, A. Xiong, L. Wang, S. Deng, J. Ye, Aco Resampling: Enhancing the performance of oversampling methods for class imbalance classification, Knowl.-Based Syst. (2020) 105818.
[16] V. Garcia, J. Sanchez, R. Mollineda, On the effectiveness of preprocessing methods when dealing with different levels of class imbalance, Knowl.-Based Syst. 25 (1) (2012) 13–21.
[17] A. Fernandez, V. Lopez, M. Galar, M.J. del Jesus, F. Herrera, Analysing the classification of imbalanced data-sets with multiple classes: binarization techniques and ad-hoc approaches, Knowl.-Based Syst. 42 (2013) 97–110.
[18] Z. Wang, Y. Li, D. Li, Z. Zhu, W. Du, Entropy and gravitation based dynamic radius nearest neighbor classification for imbalanced problem, Knowl.-Based Syst. 193 (2020) 105474.
[19] Q. Fan, Z. Wang, D. Li, D. Gao, H. Zha, Entropy-based fuzzy support vector machine for imbalanced datasets, Knowl.-Based Syst. 115 (2017) 87–99.
[20] X.S. Hu, R.J. Zhang, Clustering-based subset ensemble learning method for imbalanced data, in: Proceedings of 2013 International Conference on Machine Learning and Cybernetics, IEEE, Tianjin, China, 2013, pp. 35–39.
[21] C. Chen, M.L. Shyu, Clustering-based binary-class classification for imbalanced data sets, in: Proceedings of 2011 IEEE International Conference on Information Reuse and Integration, IEEE, Las Vegas, NV, USA, 2011, pp. 384–389.
[22] S.J. Yen, Y.S. Lee, Cluster-based under-sampling approaches for imbalanced data distributions, Expert Syst. Appl. 36 (3) (2009) 5718–5727.
[23] W.C. Lin, C.F. Tsai, Y.H. Hu, J.S. Jhang, Clustering-based undersampling in class-imbalanced data, Inform. Sci. 409–410 (2017) 17–26.
[24] C.-F. Tsai, W.-C. Lin, Y.-H. Hu, G.-T. Yao, Under-sampling class imbalanced datasets by combining clustering analysis and instance selection, Inform. Sci. 477 (2019) 47–54.
[25] S. Visa, Fuzzy Classifiers for Imbalanced Data Sets (Ph.D. thesis), University of Cincinnati, Cincinnati, OH, USA, 2007.
[26] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garcia, L. Sanchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Mult.-Valued Logic Soft Comput. 17 (2–3) (2011) 255–287.
[27] A. Fernández, S. García, M. Galar, R.C. Prati, B. Krawczyk, F. Herrera, Learning from Imbalanced Data Sets, Springer, 2018.
[28] H. Kaur, H.S. Pannu, A.K. Malhi, A systematic review on imbalanced data challenges in machine learning: Applications and solutions, ACM Comput. Surv. 52 (4) (2019) 1–36.
[29] M.S. Shelke, P.R. Deshmukh, V.K. Shandilya, A review on imbalanced data handling using undersampling and oversampling technique, Int. J. Recent Trends Eng. Res. 3 (2017) 444–449.
[30] M. Kubat, S. Matwin, Addressing the curse of imbalanced training sets: one-sided selection, in: Proceedings of the 14th International Conference on Machine Learning, Nashville, TN, USA, 1997, pp. 179–186.
[31] J. Laurikkala, Improving identification of difficult small classes by balancing class distribution, in: S. Quaglini, P. Barahona, S. Andreassen (Eds.), Artificial Intelligence in Medicine, 2001, pp. 63–66.
[32] Q. Kang, X. Chen, S. Li, M. Zhou, A noise-filtered under-sampling scheme for imbalanced classification, IEEE Trans. Cybern. 47 (12) (2017) 4263–4274.
[33] I. Tomek, Two modifications of CNN, IEEE Trans. Syst. Man Cybern. 6 (11) (1976) 769–772.
[34] P. Hart, The condensed nearest neighbor rule, IEEE Trans. Inform. Theory 14 (3) (1968) 515–516.
[35] I. Tomek, An experiment with the edited nearest-neighbor rule, IEEE Trans. Syst. Man Cybern. 6 (6) (1976) 448–452.
[36] X. Deng, W. Zhong, J. Ren, D. Zeng, H. Zhang, An imbalanced data classification method based on automatic clustering under-sampling, in: 2016 IEEE 35th International Performance Computing and Communications Conference, IPCCC, IEEE, 2016, pp. 1–8.
[37] N. Ofek, L. Rokach, R. Stern, A. Shabtai, Fast-CBUS: A fast clustering-based undersampling method for addressing the class imbalance problem, Neurocomputing 243 (2017) 88–102.
[38] X. Zhang, Q. Luo, Unbalanced data classification algorithm based on clustering ensemble under-sampling, Comput. Sci. 42 (Z11) (2015) 63–66.
[39] W.W.Y. Ng, J. Hu, D.S. Yeung, S. Yin, F. Roli, Diversified sensitivity-based undersampling for imbalance classification problems, IEEE Trans. Cybern. 45 (11) (2015) 2402–2412.
[40] M.E. Ramirez-Loaiza, M. Sharma, G. Kumar, M. Bilgic, Active learning: an empirical study of common baselines, Data Min. Knowl. Discov. 31 (2) (2017) 287–313.
[41] M. Sharma, M. Bilgic, Evidence-based uncertainty sampling for active learning, Data Min. Knowl. Discov. 31 (1) (2016) 164–202.

[42] Y. Fu, X. Zhu, B. Li, A survey on instance selection for active learning, Knowl. Inf. Syst. 35 (2) (2012) 249–283.
[43] A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, Science 344 (6191) (2014) 1492–1496.
[44] P. Christensen, A. Kensler, C. Kilpatrick, Progressive multi-jittered sample sequences, in: Computer Graphics Forum, Vol. 37, No. 4, Wiley Online Library, 2018, pp. 21–33.
[45] S. Bobkov, M. Ledoux, One-dimensional Empirical Measures, Order Statistics, and Kantorovich Transport Distances, Vol. 261, No. 1259, American Mathematical Society, 2019.
[46] C. López-Vázquez, E. Hochsztain, Extended and updated tables for the friedman rank test, Comm. Statist. Theory Methods 48 (2) (2019) 268–281.
[47] R.S.M. de Barros, J.I.G. Hidalgo, D.R. de Lima Cabral, Wilcoxon rank sum test drift detector, Neurocomputing 275 (2018) 1954–1963.