Neurocomputing 329 (2019) 172–187

Contents lists available at ScienceDirect

Neurocomputing

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

Class imbalance learning using UnderBagging based kernelized extreme learning machine

Bhagat Singh Raghuwanshi, Sanyam Shukla*

Department of Computer Science and Engineering, Maulana Azad National Institute of Technology (MANIT), Bhopal, Madhya Pradesh 462003, India

ARTICLE INFO

Article history:
Received 15 June 2018
Revised 13 October 2018
Accepted 25 October 2018
Available online 1 November 2018

Communicated by Dr. G.-B. Huang

Keywords:
Kernelized extreme learning machine
Class imbalance problem
Random undersampling
Voting methods

ABSTRACT

Many real-life problems can be described as imbalanced classification problems, where the number of samples belonging to one of the classes is heavily outnumbered than the numbers in other classes. The samples with larger and smaller class proportion are referred to as the majority and the minority class respectively. Traditional extreme learning machine (ELM) and Support Vector Machine (SVM) provides equal importance to all the samples leading to results biased towards the majority class. Many variants of ELM-like Weighted ELM (WELM), Boosting WELM (BWELM) etc. are designed to solve the class imbalance problem effectively. This work develops a novel UnderBagging based kernelized ELM (UBKELM) to address the class imbalance problem more effectively. In this paper, an UnderBagging ensemble is proposed which employs kernelized ELM as the component classifier. UnderBagging ensemble incorporates the strength of random undersampling and the bagging. This work creates several balanced training subsets by random undersampling of the majority class samples. The number of training subsets depending on the degree of the class imbalance. This work uses kernelized ELM as the component classifier to make the ensemble as it is stable and has the promising generalization performance. The computational cost of UBKELM is significantly lower than BWELM, BalanceCascade, EasyEnsemble, hybrid artificial bee colony WELM. The final outcome is computed by the majority voting and the soft voting of these classification models. The proposed work is assessed by employing benchmark real-world imbalanced datasets taken from the KEEL dataset repository. The experimental results show that the proposed work outperforms in contrast with the rest of the classifiers for class imbalance learning.

1. Introduction

During the past years, ELM proposed by Huang et al. [1] has gained increasing interest among the researchers around the world. It is a single-hidden layer feedforward neural network (SLFN) with random weights between the input and the hidden layer. It computes the weights between the hidden layer and the output layer analytically using the Moore–Penrose (MP) pseudoinverse. This makes ELM fast compared to other gradient-based algorithms. Conventional ELM [2,3] does not take into account the class imbalance problem effectively. Numerous variants of ELM-like WELM [4], Boosting WELM [5], class-specific cost-sensitive boosting WELM [6], Ensemble WELM [7], Regularized Weighted Circular Complex valued ELM [8], UnderBagging reduced kernelized WELM [9], CCR-ELM [10] and CS-ELM [11] etc. have been designed to deal with the class imbalance problem effectively. As mentioned above, ELM randomly initializes the weights between the input and the hidden layer. These random weights are used to map the input data to the feature space. These weights remain unaltered amid the training phase. Due to this some of the samples usually at the classification boundary are misclassified in certain realizations. The work in [12,13] shows that ELM with Gaussian kernel is referred as kernelized ELM (KELM) outperforms ELM with Sigmoid node for the small and medium scale learning problems. ELM with Gaussian kernel is relatively slower on the larger datasets compared to the ELM with Sigmoid node. It is also mentioned in the aforementioned work that for most of the datasets (small and medium scale learning problems) ELM with Gaussian kernel runs faster than ELM with Sigmoid node.

WELM minimizes the weighted least squares error along with regularization to find the weights between the hidden and the output layer. WELM uses two weighting schemes to assign class-wise weights to the samples. These weighting schemes consider class skewness for assigning weights to the samples.

BWELM [5] enhances the performance of WELM using ensemble method. It is a variant of AdaBoost algorithm [14] with WELM as the component classifier. This AdaBoost variant differs from

* Corresponding author.
E-mail addresses: bhagat.mnit@gmail.com (B.S. Raghuwanshi), sanyamshukla@gmail.com (S. Shukla).

https://doi.org/10.1016/j.neucom.2018.10.056
0925-2312/© 2018 Elsevier B.V. All rights reserved.

AdaBoost as all the classes are assigned equal weights. This AdaBoost variant also differs in weight updating approach. It update the weights in class-wise manner. After each iteration, the weights of the samples belonging to each class are rescaled such that the total weight assigned to all the classes remains equal. The final outcome of BWELM is computed using weighted voting. It has been shown in [5], that BWELM takes about 10–50 times training and testing time compared to WELM. It enhances the performance of WELM at the cost of increases redundancy and computational overhead.

CCR-ELM [10] employs class-specific cost regularization for attaining good generalization performance. For a binary classification problem, CCR-ELM has to tune two regularization parameters to get the optimal results by performing a grid search on $(2^{-24}, 2^{-23} \cdots 2^{24}, 2^{25})$. This needs finding the accuracy of CCR-ELM on training dataset for 2500 distinct combinations of the regularization parameter, which is computationally intensive.

CS-ELM [11] sets the value of class-specific regularization in accordance with the proportion of each class. CS-ELM has to tune one regularization parameter to get the best results by performing a search on $(2^{-18}, 2^{-16} \cdots 2^{48}, 2^{50})$. This needs finding the accuracy of CS-ELM on training dataset for 50 distinct combinations of the regularization parameter.

EWELM [7], is a heterogeneous ensemble with WELM as the component classifiers with different activation function. EWELM combines the output of the component classifiers using weighted voting. For later, different weights are assigned to different component classifiers using differential evolution method. It enhances the performance of WELM at the cost of increased redundancy and computational overhead. For some datasets, the differential evolution based ensemble method may over-train the minority class samples leading to the reduced accuracy. It can be noted that all the above mentioned variants of WELM minimizes the weighted least squares error. Weights assigned to the samples determine the level of accuracy of WELM. Authors in [5,7] have worked on dynamic assignment of weights to the training samples. None of variants of ELM have focused on converting the imbalanced classification problem to balanced classification problem. This work uses UnderBagging technique to convert an imbalance classification problem into several balanced classification problems. It has been shown [15] ELM attain similar or better generalization performance compared to SVM with much faster speed for the balanced classification problems. Motivated by this, the proposed work uses kernelized ELM for developing classification models for these balanced classification problems. The outcomes of the created ensemble of classifiers is combined using majority voting [16] and soft voting [17]. The proposed method does not require any weighting scheme to assign weights to the training samples.

The rest of this paper is organized as follows. The Section 2 discusses the related work in detail. The Section 3 describes the proposed method. The Section 4 presents the experimental setup and the performance evaluation in detail. The Section 5 concludes the proposed work and also provides guidelines for the future work.

2. Related work

2.1. Class imbalance learning

Recently the imbalanced learning problem [18–20], has received huge attention from machine learning community. It is observed that most of the classification problems in the real world have a difference in the class proportion. Some of the examples of the class imbalance problems are imbalanced credit scoring [21], Customer classification [22], Cancer malignancy grading [23] etc. The problem associated with the class imbalanced learning is that the standard learners are often biased towards the majority class. For problems like cancer malignancy grading, the minority class detection has more concern compared to the majority class detection. Therefore, higher positive recall value for the minority class is desirable in case of the imbalanced classification problem.

2.1.1. Solutions for class imbalance learning

The solution available for the imbalanced classification problems [24] can be broadly categorized as data level approaches, algorithmic level approaches and hybrid-based approaches to the class imbalance problem which are described in the following section.

Data level approaches. Data level approaches use resampling approaches like oversampling and undersampling [19,25] to modify the data distribution. They reduce the degree of the class imbalance of an imbalanced classification problem in order to achieve better classification accuracy for the minority class.

1. Undersampling: undersampling approach removes a fraction of the majority class samples and balances the data distribution at the cost of the information loss.
2. Oversampling: oversampling approach balances the class distribution by random replication of the minority class samples. This may lead to over-fitting. One of the more effective strategies is the synthetic minority over-sampling techniques (SMOTE) [26] and weighted kernel based SMOTE [27].

Algorithmic level approaches. Algorithmic level approaches make change in the classifier design to handle the class imbalance problem. Algorithmic level approaches may be broadly categorized as:

1. Cost-sensitive learning: standard classifiers minimize the least squares error due to which the samples belonging to the minority class are misclassified. Cost-sensitive learning assigns different cost to the minority and the majority class samples. For example, WELM [4] and WSVM [28]. These classifiers assign more weight to the minority class samples compared to the majority class samples. To find the optimal solution they minimize the weighted least squares error. The problem with this category of classifiers is determining the optimal value of the weights which can be assigned to the training samples [24].
2. Ensembling approaches: it has been reported in [24,29] that ensembling approaches have received much attention because of their flexible characteristics. Several ensemble based solutions have been proposed to handle the class imbalance problem. BWELM [5] and EWELM [7] are some of them. Both EWELM and BWELM assign dynamic weights to the training samples. The samples which are difficult i.e., the samples which are misclassified are assigned more weight compared to the samples which are correctly classified. Another category of ensembling approaches is the boosting-based ensembling approach. For example, RUSBoost [30] and SMOTEBoost [31]. RUSBoost are the algorithms that combines boosting and data random undersampling. RUSBoost is a variant of SMOTEBoost [31].

Hybrid-based approaches. The hybrid solution uses both data level and algorithmic level approaches. For example, EasyEnsemble and BalanceCascade [25] make change at the data level to alleviate the class imbalance problem using undersampling.

1. EasyEnsemble: It uses random undersampling to create several balanced subsets of the data. It also uses the algorithmic approach more specifically the ensemble method to deal with the class imbalance problem.
2. BalancedCascade: It creates the first training subset by random undersampling of the samples belonging to the majority class. The classification model obtained using the above-mentioned

training subset is used to predict the class label of the training subset. The correctly classified majority class samples are removed from the training subset and new majority class samples are added in their place. In this way, new training subset is created. Here, the component classification models are developed in a sequential manner. The final outcome of the classifier ensemble is determined using weighted voting.

2.2. Extreme learning machine

ELM [1,15] is a single-hidden layer feedforward neural network with random weights between the input and the hidden layer. ELM provides good generalization performance and gets rid of the iterative time-consuming training process. It uses Moore–Penrose pseudo inverse for computing the weights between the hidden and the output layer which makes it fast. Given N training samples $\{(\mathbf{x_i},\mathbf{t_i})\}_{i=1}^{N}$. Here, input vector $\mathbf{x_i}=[x_{i1},x_{i2},\ldots,x_{in}]^T \in R^n$ and its desired output $\mathbf{t_i}=[t_{i1},t_{i2},\ldots t_{im}]^T \in R^m$, superscript T denotes the matrix/vector transpose. Here, n is the number of the input neurons which is equal to the number of the input features, L is the number of the hidden neurons and m is the number of the output neurons, which is equal to the number of classes. The input weight matrix is represented by $\mathbf{U}=[\mathbf{u_1},\mathbf{u_2},\ldots\mathbf{u_j},\ldots,\mathbf{u_L}]^T \in \boldsymbol{R^{L\times n}}$ and the bias of the hidden neurons, $\mathbf{b}=[b_1,b_2,\ldots b_j,\ldots b_L]^T \in \boldsymbol{R^L}$, here, $\mathbf{u_j}=[u_{j1},u_{j2},\ldots u_{jn}]$ are the weights connecting the jth hidden neuron with the input neurons, b_j is the bias of the jth hidden neuron. These weights remain unchanged during the training phase. The output of the hidden layer for ith sample is given as follows:

$$\mathbf{h}(\mathbf{x_i})=G(\mathbf{U}\mathbf{x_i}+\mathbf{b}) \tag{1}$$

Here, $G(.)$ represents the activation function. The output of the hidden layer for all the training samples is represented by $\mathbf{H}$ which shown in the following equation:

$$\mathbf{H}=\begin{bmatrix} h_1(x_1) & h_2(x_1) & . & h_L(x_1)\\ h_1(x_2) & h_2(x_2) & . & h_L(x_2)\\ . & . & . & .\\ h_1(x_N) & h_2(x_N) & . & h_L(x_N)\end{bmatrix}_{N\times L} \tag{2}$$

In original ELM, the following optimization problem is formulated for finding the weights between the hidden and the output layer.

$$\text{Minimize: } \frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2+\lambda\frac{1}{2}\sum_{i=1}^{N}\|\xi_i\|^2$$
$$\text{Subject to: } \mathbf{h}(\mathbf{x_i})\boldsymbol{\beta}=\mathbf{t_i^T}-\xi_\mathbf{i}^\mathbf{T},\quad i=1,\ldots,N \tag{3}$$

The first term of the objective function is the regularization term, also known as the structural risk $\|\boldsymbol{\beta}\|^2$ and the second term is the least squares error, also known as the empirical risk $\|\boldsymbol{\xi}\|^2$. The structural risk maximizes the margin for good generalization [32]. The regularization parameter, λ is used to control the trade-off between the two risks. Here, $\boldsymbol{\beta}$ is the weight vector connecting the hidden layer and the output layer, $\boldsymbol{\xi}$ is the error vector. The solution of the problem given in (3) is determined by Huang et al. [15] which is given below:

$$\boldsymbol{\beta}=\begin{cases}\mathbf{H^T}\left(\frac{I}{\lambda}+\mathbf{HH^T}\right)^{-1}\mathbf{T} & if\ N<L\\ \left(\frac{I}{\lambda}+\mathbf{H^TH}\right)^{-1}\mathbf{H^TT} & if\ N>L\end{cases} \tag{4}$$

Here, I represent the identity matrix of appropriate dimension. The predicted final outcome for any test sample $\mathbf{x}$ is given by following equation:

$$\mathbf{f}(\mathbf{x})=\begin{cases} sign\ \mathbf{h}(\mathbf{x})\mathbf{H^T}\left(\frac{I}{\lambda}+\mathbf{HH^T}\right)^{-1}\mathbf{T} & if\ N<L\\ sign\ \mathbf{h}(\mathbf{x})\left(\frac{I}{\lambda}+\mathbf{H^TH}\right)^{-1}\mathbf{H^TT} & if\ N>L\end{cases} \tag{5}$$

Here, $\mathbf{f}(\mathbf{x})=[f_k(\mathbf{x}),\ldots f_m(\mathbf{x})]$ is the output function vector. The predicted class label can be obtained by using the following equation:

$$label(\mathbf{x})=\underset{k}{\operatorname{argmax}}\ f_k(\mathbf{x}),\quad k=1,\ldots,m. \tag{6}$$

2.3. Kernelized extreme learning machine

It has been shown [33,34] ELM with the Sigmoid node usually may needs a large number of hidden nodes to attain good generalization performance. Moreover, it also has been shown [35,36] more hidden nodes with randomly chosen weights and biases result in the large fluctuation of the performance. Kernelized ELM has good generalization performance than ELM with Sigmoid node [12,13]. KELM with Gaussian kernel function maps the data from the input space to the feature space as follows:

$$K(\mathbf{x_i},\mathbf{x_j})=exp\left(\frac{-\|\mathbf{x_i}-\mathbf{x_j}\|^2}{\sigma^2}\right) \tag{7}$$

Here, σ is the kernel width parameter, $\mathbf{x_i}$ represents the ith sample and $\mathbf{x_j}$ represents the jth centroid $i,\ j\in 1,\ 2,\ \ldots N$. The number of Gaussian kernel function used in [15] was equal to the number of training samples. Whereas, in [37] the author has randomly varied the number of kernels and has chosen kernel centroid, $\mathbf{x_i}$ and the width, σ randomly. The kernel matrix of the hidden layer is given by the following equation:

$$\mathbf{H}=\begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & . & K(x_1,x_N)\\ K(x_2,x_1) & K(x_2,x_2) & . & K(x_2,x_N)\\ . & . & . & .\\ . & . & . & .\\ K(x_N,x_1) & K(x_N,x_2) & . & K(x_N,x_N)\end{bmatrix}_{N\times N} \tag{8}$$

Here, $K(\mathbf{x_i},\ \mathbf{x_j})$ represents the output of the jth hidden neuron for the ith input, $\mathbf{x_i}$, here $i,j\in 1,2,\ldots,N$. All the notations used in this section are same as that of Section 2.2. In kernelized ELM the problem of finding the output layer weights is formulated as follows:

$$\text{Minimize: } \frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2+\lambda\frac{1}{2}\sum_{i=1}^{N}\|\xi_\mathbf{i}\|^2$$
$$\text{Subject to: } \mathbf{h}(\mathbf{x_i})\boldsymbol{\beta}=\mathbf{t_i^T}-\xi_\mathbf{i}^\mathbf{T},\quad i=1,\ldots,N \tag{9}$$

The Lagrangian function for the above optimization problem is formulated as follows:

$$\mathcal{L}_{DELM}=\frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2+\lambda\frac{1}{2}\sum_{i=1}^{N}\|\xi_\mathbf{i}\|^2-\sum_{i=1}^{N}\boldsymbol{\alpha_i}\left(\mathbf{h}(\mathbf{x_i})\boldsymbol{\beta}-\mathbf{t_i^T}+\xi_\mathbf{i}^\mathbf{T}\right) \tag{10}$$

Here, $\mathcal{L}_{DELM}$ represents the dual optimization problem. $\boldsymbol{\alpha_i}$ is the Lagrangian coefficient for the sample $\mathbf{x_i}$. The KKT conditions can be obtained by taking partial derivatives with respect to the variables ($\boldsymbol{\alpha_i}$, $\boldsymbol{\beta}$, ξ_i) as shown in the following equations:

$$\frac{\partial\mathcal{L}_{DELM}}{\partial\boldsymbol{\beta}}=0\Rightarrow\boldsymbol{\beta}=\sum_{i=1}^{N}\boldsymbol{\alpha}_i\mathbf{h}(\mathbf{x_i})^T=\mathbf{H^T}\boldsymbol{\alpha} \tag{11}$$

$$\frac{\partial\mathcal{L}_{DELM}}{\partial\xi_i}=0\Rightarrow\boldsymbol{\alpha_i}=\lambda\xi_i,\quad i=1,2,\ldots,N \tag{12}$$

$$\frac{\partial\mathcal{L}_{DELM}}{\partial\alpha_i}=0\Rightarrow\left(\mathbf{h}(\mathbf{x_i})\boldsymbol{\beta}-\mathbf{t_i^T}+\xi_\mathbf{i}^\mathbf{T}\right)=0,\quad i=1,2,\ldots,N \tag{13}$$

For the small training samples, using Eqs. (11)–(13), [15] has determined the solution which is given below.

$$\boldsymbol{\beta} = \mathbf{H^T}\left(\frac{I}{\lambda} + \mathbf{HH^T}\right)^{-1}\mathbf{T} \tag{14}$$

Here, I represents the identity matrix of, $N \times N$ dimension. Applying Mercer's condition to define the kernel matrix of KELM as illustrated in the following equation:

$$\Omega_{ELM} = \mathbf{HH^T} : \Omega_{ELM_{i,j}} = h(x_i).h(x_j) = K(\mathbf{x_i}, \mathbf{x_j}) \tag{15}$$

The output of KELM can be computed by utilizing Eqs. (14) and (15) as follows.

$$\mathbf{f}(\mathbf{x}) = sign \begin{bmatrix} K(\mathbf{x}, \mathbf{x_1}) \\ \cdot \\ \cdot \\ K(\mathbf{x}, \mathbf{x_N}) \end{bmatrix}^T \left(\frac{I}{\lambda} + \Omega_{\mathbf{ELM}}\right)^{-1} \mathbf{T} \tag{16}$$

2.4. Weighted extreme learning machine

Conventional ELM does not account for the good generalization performance while dealing with the class imbalanced problems. WELM [4] was proposed for handling the class imbalanced problem effectively. In WELM two generalized weighting schemes were proposed and assessed. These generalized weighting schemes assign weights to the samples as per their class distribution. The two weighting schemes proposed by WELM are:

Weighting scheme $W1$:

$$W_{ii} = \frac{1}{q_k} \tag{17}$$

Here, q_k is the total number of samples belonging to kth class.

Weighting scheme $W2$:

$$W_{ii} = \begin{cases} \frac{0.618}{q_k} & if \ \ (q_k > q_{avg}) \\ \frac{1}{q_k} & if \ \ (q_k <= q_{avg}) \end{cases} \tag{18}$$

Here, q_{avg} represents the average number of samples for all classes. Weight W_{ii} is assigned to the ith samples. Samples belonging to the minority-class will be assigned equal weights to $1/q_i$, in both the weighting schemes. The second weighting scheme assigns lesser weight to the majority-class samples compared to the first weighting scheme.

In WELM [4] the following optimization problem is formulated:

$$\text{Minimize:} \ \ \frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2 + \frac{1}{2}\lambda\mathbf{W}\sum_{i=1}^{N}\|\xi_\mathbf{i}\|^2$$

$$\text{Subject to:} \ \ \mathbf{h}(\mathbf{x_i})\boldsymbol{\beta} = \mathbf{t_i^T} - \xi_\mathbf{i}^\mathbf{T}, \quad i = 1, \ldots, N \tag{19}$$

Here, $\mathbf{W} = diag(W_{ii})$ is an $N \times N$ diagonal matrix whose diagonal elements are the weights assigned to the training samples. WELM maps the input data to the feature space by using the following two approaches.

1. WELM using the Sigmoid node. It uses random weights and the Sigmoid activation function to find the hidden layer output matrix **H**. The solution of the problem given by Eq. (19) is determined in [4] which is given below.

$$\boldsymbol{\beta} = \begin{cases} \mathbf{H^T}\left(\frac{I}{\lambda} + \mathbf{WHH^T}\right)^{-1}\mathbf{WT} & if \ N < L \\ \left(\frac{I}{\lambda} + \mathbf{H^TWH}\right)^{-1}\mathbf{H^TWT} & if \ N > L \end{cases} \tag{20}$$

2. WELM using the Gaussian kernel function maps the input data to the feature space. The solution for kernelized WELM (KWELM) given by Eq. (19) is determined in [4] which is given below.

$$\boldsymbol{\beta} = \mathbf{H^T}\left(\frac{I}{\lambda} + \mathbf{WHH^T}\right)^{-1}\mathbf{WT} \tag{21}$$

Using Eqs. (15) and (21) the output of the KWELM can be rewritten as follow:

$$\mathbf{f}(\mathbf{x}) = sign \begin{bmatrix} K(\mathbf{x}, \mathbf{x_1}) \\ \cdot \\ \cdot \\ K(\mathbf{x}, \mathbf{x_N}) \end{bmatrix}^T \left(\frac{I}{\lambda} + \mathbf{W}\Omega_{\mathbf{ELM}}\right)^{-1} \mathbf{WT} \tag{22}$$

Compared to the Sigmoid node based WELM, KWELM has a better classification performance [4].

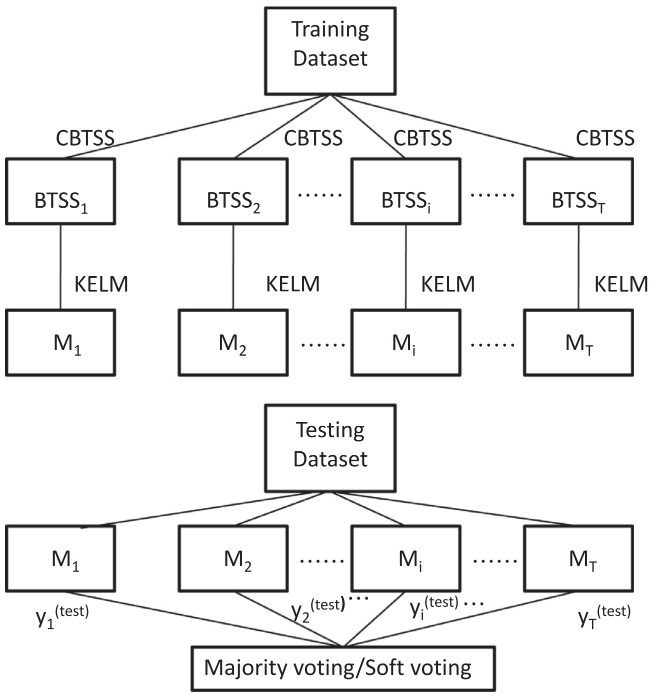


Fig. 1. UnderBagging kernelized ELM.

3. Proposed UnderBagging based kernelized ELM (UBKELM)

ELM attains similar or better generalization performance compared to SVM with much faster speed for the balanced classification problems as mentioned in [15]. Variants of ELM-like WELM, BWELM, EWELM assign weights to the training samples to address the class imbalance problem effectively. It has been stated in [10] that the performance of WELM may be suboptimal due to the weight computation schemes which is based on the number of samples. BWELM and EWELM are the ensemble based variants of WELM with Sigmoid node, which outperform WELM at the cost of increased redundancy. Both BWELM and EWELM use dynamic weights as the value of the weights assigned to the samples heavily affects the performance of WELM. So, the proposed work develops an algorithm which does not require the assignment of weights to the samples. For this, the proposed work divides an imbalanced classification problem into several balanced classification problems. For developing the classification models for the aforementioned balanced classification problems this work uses ELM with Gaussian kernel due to the reasons delineated below.

- As mentioned above, ELM with Gaussian kernel outperforms ELM with sigmoid node.

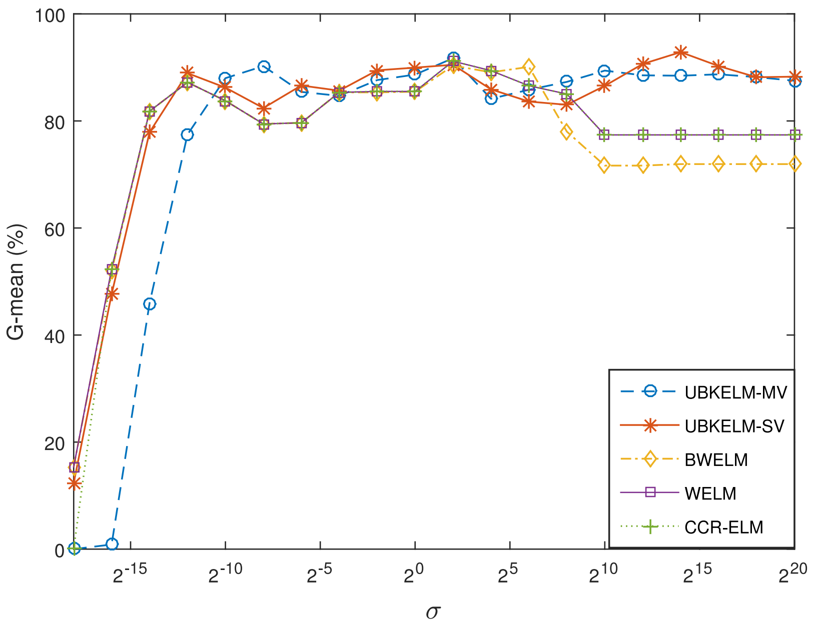


Fig. 2. G-mean for glass4 dataset with σ variation and λ having the optimal value.

- In kernelized WELM the diagonal matrix, $W_{N\times N}$ associated with every training samples. Hence, the Kernelized WELM has more computational complexity compared to KELM. Kernelized WELM performs better than KELM for imbalanced classification problems. As the component classifier is to be designed for balanced classification problem, KELM is a better choice.

The proposed work (Algorithm 1) creates several balanced training datasets by random undersampling of the majority class samples. The number of training subsets, T is determined by using the following equation.

$$T = \left\lceil \frac{\max(t_k)}{\min(t_k)} \right\rceil, \quad t_k \in 1, 2, \ldots, m \tag{23}$$

Here, t_k represents the number of samples belonging to kth class.

The outcome of these classifiers is combined using both majority voting and soft voting. The work differs from Bagging as the size of balanced training subset is not equal to the size of the original training dataset. Only majority class samples are under-sampled. No oversampling is done in the proposed work whereas, Bagging uses both undersampling and oversampling to create datasets whose size is equal to the original training dataset.

This work differs from EasyEnsemble as EasyEnsemble is the ensemble of ensemble whereas, the proposed work is just an ensemble. EasyEnsemble uses AdaBoost ensemble as the component classifier whereas the proposed work uses KELM as the component classifier.

The Fig. 1 gives the pictorial representation of the proposed work. The term CBTSS represents "Creating balanced training subset" and BTSS represents "Balanced training subset".

Algorithm 1: UnderBagging based KELM (UBKELM).

```
    Input: Given $N$ observations along with the class labels
    $(\mathbf{x_i}, \mathbf{t_i})$, $\mathbf{x_i} \in R^n$, $\mathbf{t_i} \in R^m$.
    Output: UBKELM model for classification.
 1: procedure UBKELM
 2:     Find the number of balanced training subset to be created,
        $T$ by employing the Eq. (23).
 3:     $i = 1$
 4:     repeat
 5:         Randomly create balanced training subset (CBTSS) by
            employing the following steps
            (1) Find $N_{min}$ which represents the number of samples
            belonging to the class with the least number of samples.
            (2) The balanced training subset, $CBTSS_i$ can be formed by
            randomly selecting $N_{min}$ samples from each class, whose
            size is
            $\widetilde{N} = m \times N_{min}$    (24)
 6:         Using the above-mentioned training subsets develop the
            classification model, $M_i$ by using KELM algorithm.
 7:         Compute the predicted output, $\mathbf{y}_i^{test}$ of the samples
            corresponding to the test dataset using the model, $M_i$.
 8:         $i = i + 1$
 9:     until $i \le T$
        Compute the final outcome using
        (1) Majority voting: $\mathbf{x} = mode(\mathbf{y}_1^{test}, \mathbf{y}_2^{test}, ..\mathbf{y}_i^{test}..\mathbf{y}_T^{test})$.
        (2) Soft voting::
            $\mathbf{x} = argmax \sum_{i=1}^{T} \mathbf{y}_i^{test}$.
10: end procedure
```

3.1. Computational cost

The following subsection analyzes the computational cost of KELM and KWELM).

3.1.1. Computational cost of KELM

Eq. (14) can be used to determine the output weight , $\boldsymbol{\beta}$ of KELM. The computational cost of computing kernel matrix, Ω is equal to $O(nN^2)$. The computational cost of computing the inverse of the kernel matrix of size $N \times N$ is equal to $O(N^3)$. The computational complexity of determining the output weight, $\boldsymbol{\beta}$ is given below:

$$O(2N^3 + nN^2 + mN^2) \tag{25}$$

UBKELM uses KELM as the component classifier. The computational cost of KELM mainly depends on the size of kernel matrix which is $N \times N$. In UBKELM the size of balanced training subset is $\widetilde{N}$. So,

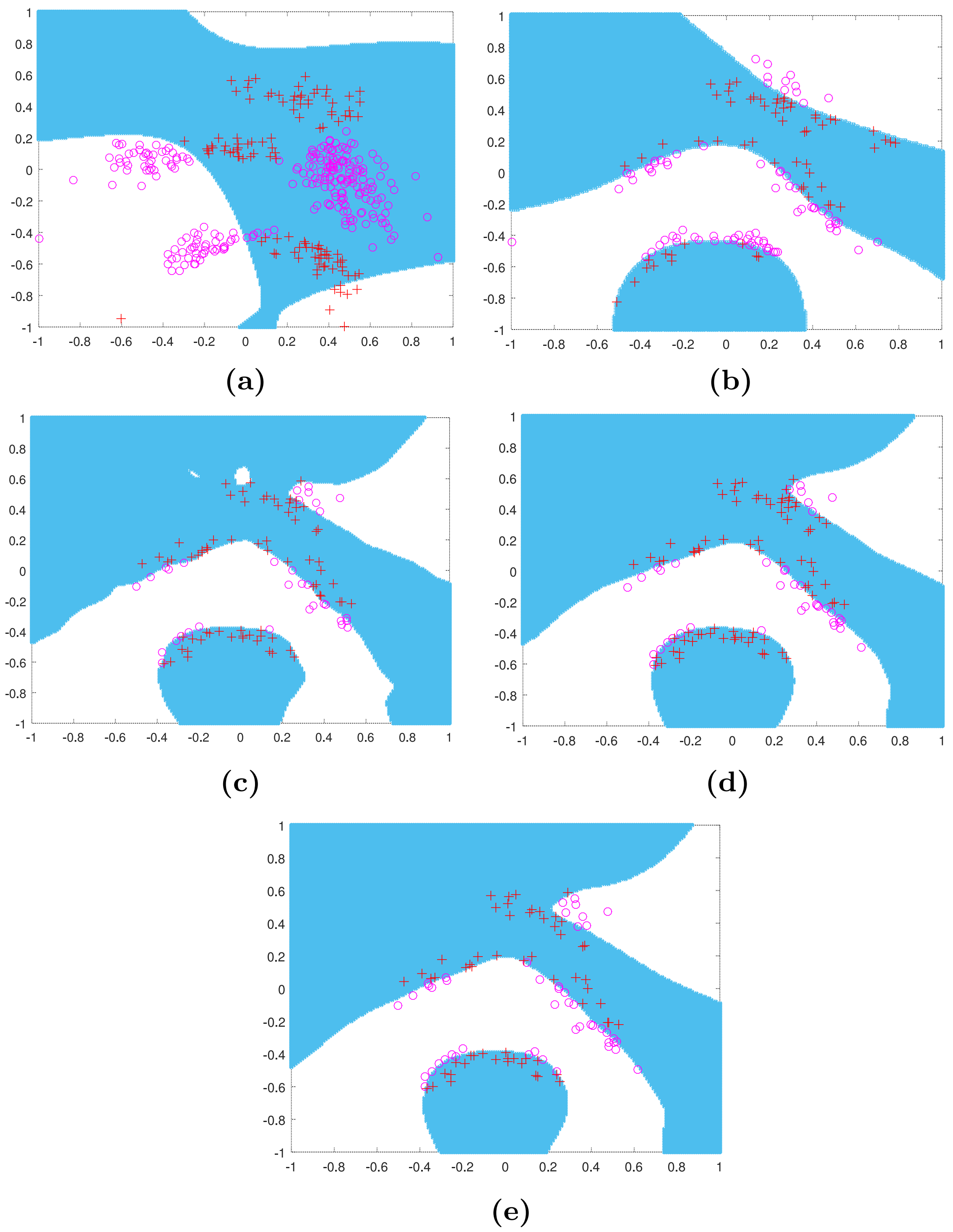


Fig. 3. Decision boundary distribution of the banana dataset: circle represents the misclassified majority class samples and plus represents the misclassified minority class samples and shaded blue color represents the majority region whereas, white color represents the minority region: (a) BWELM; (b) WELM; (C) CCR-ELM; (d) UBKELM-MV; (e) UBKELM-SV.

the cost of creating one component classifier is given below.

$$O(2\tilde{N}^3 + n\tilde{N}^2 + m\tilde{N}^2) \tag{26}$$

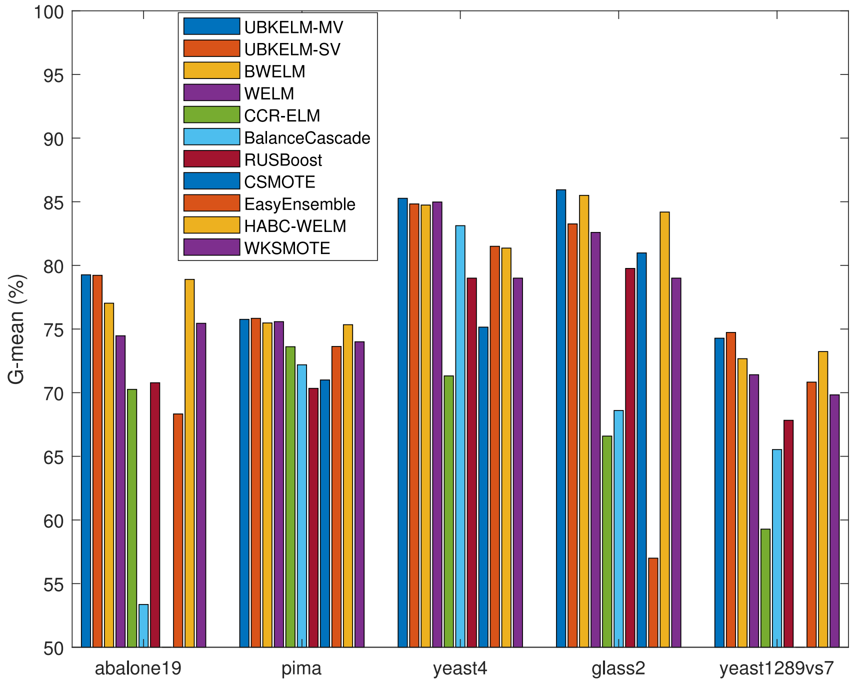
Here, $\tilde{N} < N$.

3.1.2. Computational cost of kernelized WELM

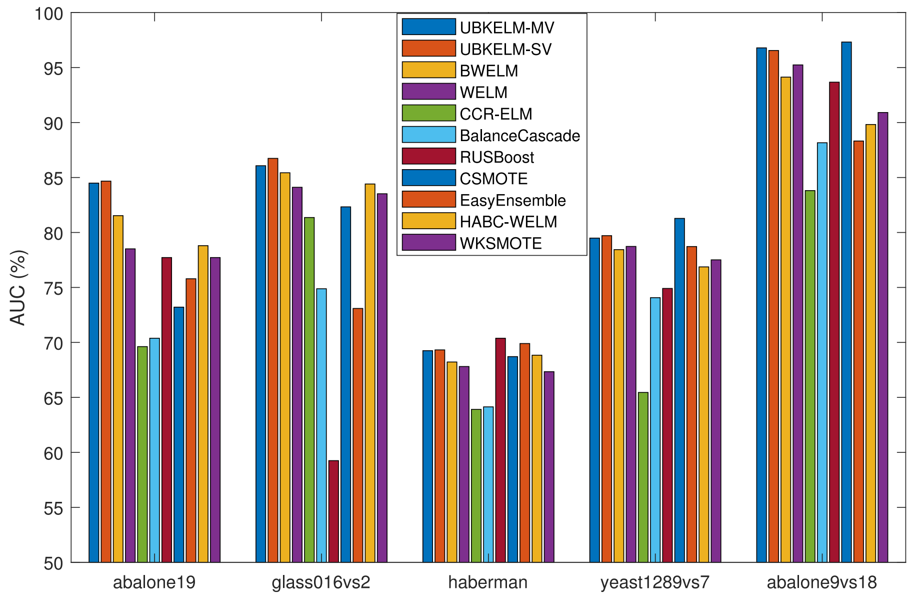
Eq. (21) can be used to determine the output weight, $\boldsymbol{\beta}$ of kernelized WELM. The computational cost of computing kernel matrix, Ω is equal to $O(nN^2)$. The computational cost of determining the matrix, $\mathbf{W}\Omega$ is $O(N^3 + nN^2)$. The computational cost of computing the inverse of the kernel matrix of size $N \times N$ is equal to $O(N^3)$. The computational cost of matrix multiplication, **WT** is equal to $O(mN^2)$. The computational cost of determining the output weight matrix, $\boldsymbol{\beta}$ is given below:

$$O(3N^3 + nN^2 + 2mN^2) \tag{27}$$

BWELM is a variant of AdaBoost algorithm with KWELM as the component classifier. Its training time depends on the number of



(a)



(b)

Fig. 4. Display of the performance of different classifiers. (a) G-mean (b) AUC.

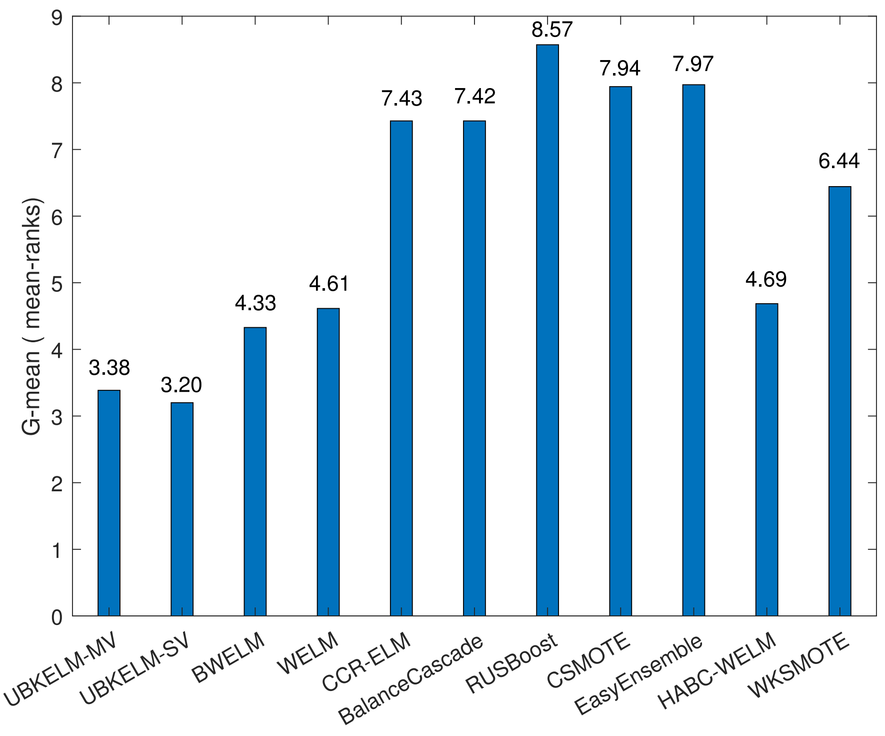
iterations. It takes about 10–50 times training and testing time compared to KWELM.

3.1.3. Stability and convergence of UBKELM

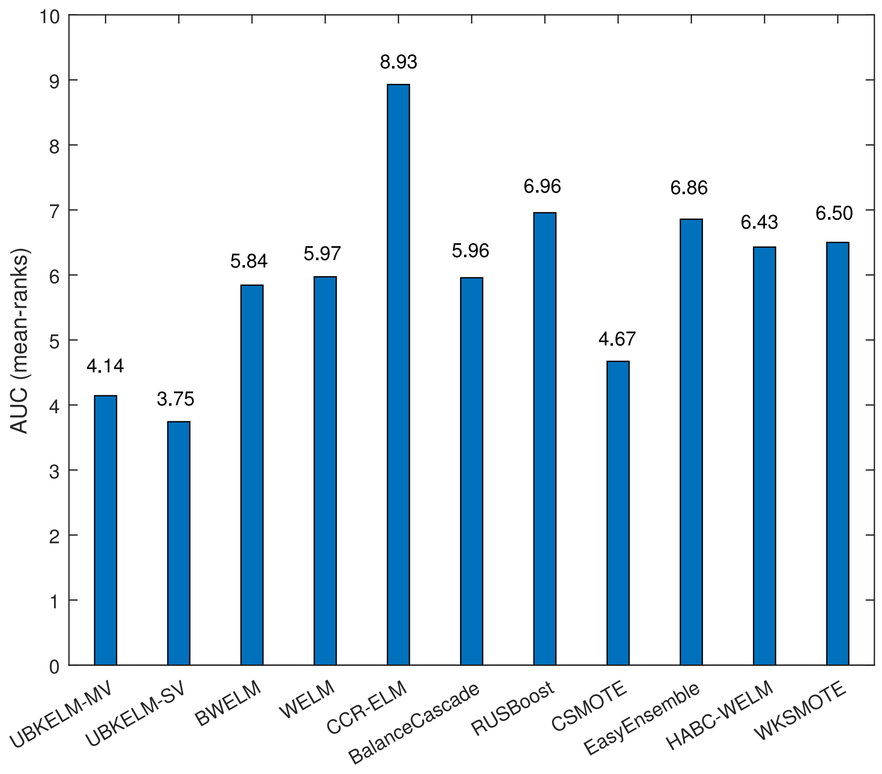
The balance training datasets created by using the majority random undersampling has $\widetilde{N}$ samples which are given as input to KELM, the base classifiers of the proposed UBKELM. It is guaranteed to converge as illustrated in the Theorem 1 stated in [1,38] which are expressed below.

Theorem 1. *Given $\widetilde{N}$ distinct training samples $(\mathbf{x_i}, \mathbf{t_i})|\mathbf{x_i} \in R^n$, $\mathbf{t_i} \in R^m$, If a kernelized single-hidden layer feed-forward neural networks (SLFNs) has $\widetilde{N}$ strict positive definite (SPD) kernel functions, then the kernel matrix $\mathbf{H}_{\widetilde{N}\times\widetilde{N}}$ is a square matrix which is invertible and satisfies $\| \mathbf{H}_{\widetilde{N}\times\widetilde{N}}\boldsymbol{\beta} - \mathbf{T} \| = 0$.*

Proof. The detailed proof of Theorem 1 is given by [1,38]. The main results are briefly summarized as follows: As per Theorem 13.2 stated in the [39], if $K(.,x_1),\dots,K(.,x_{\widetilde{N}})$ is strictly positive definite, the function $K(.,x_1),\dots,K(.,x_{\widetilde{N}})$ are linearly independent, which implies that for every finite set $\mathbf{x_i}|\mathbf{x_i} \in R^n, i = 1, 2\dots\widetilde{N}$, the kernel matrix $\mathbf{H}_{\widetilde{N}\times\widetilde{N}}$ is a square matrix and the column vectors $\left[K(.,x_1),\dots,K(.,x_{\widetilde{N}})\right]^T$ in the matrix $\mathbf{H}_{\widetilde{N}\times\widetilde{N}}$ are linearly independent. So, rank $\left(\mathbf{H}_{\widetilde{N}\times\widetilde{N}}\right) = \widetilde{N}$, i.e., the kernel matrix $\mathbf{H}_{\widetilde{N}\times\widetilde{N}}$ is invertible.

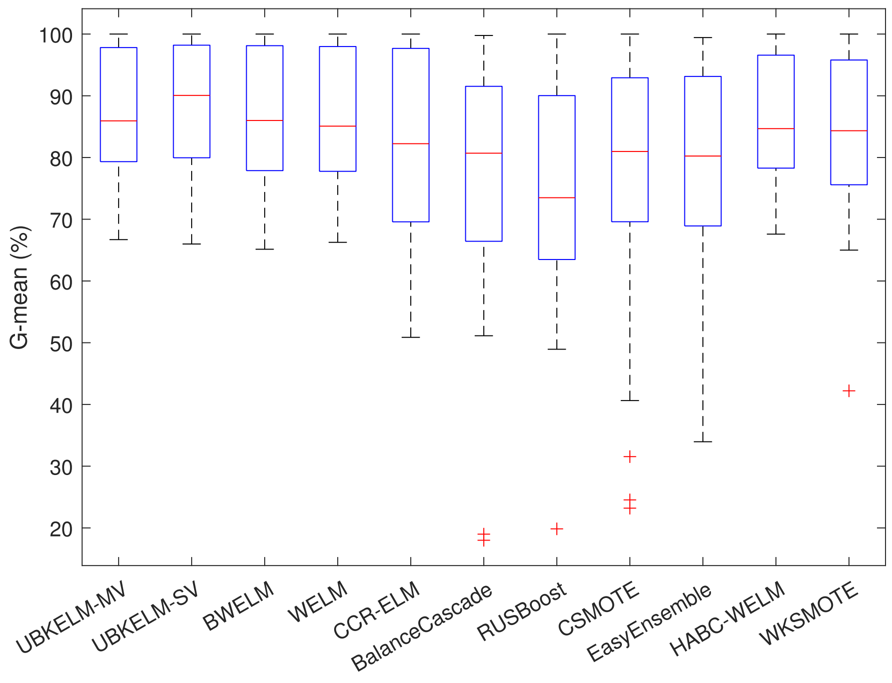


(a)

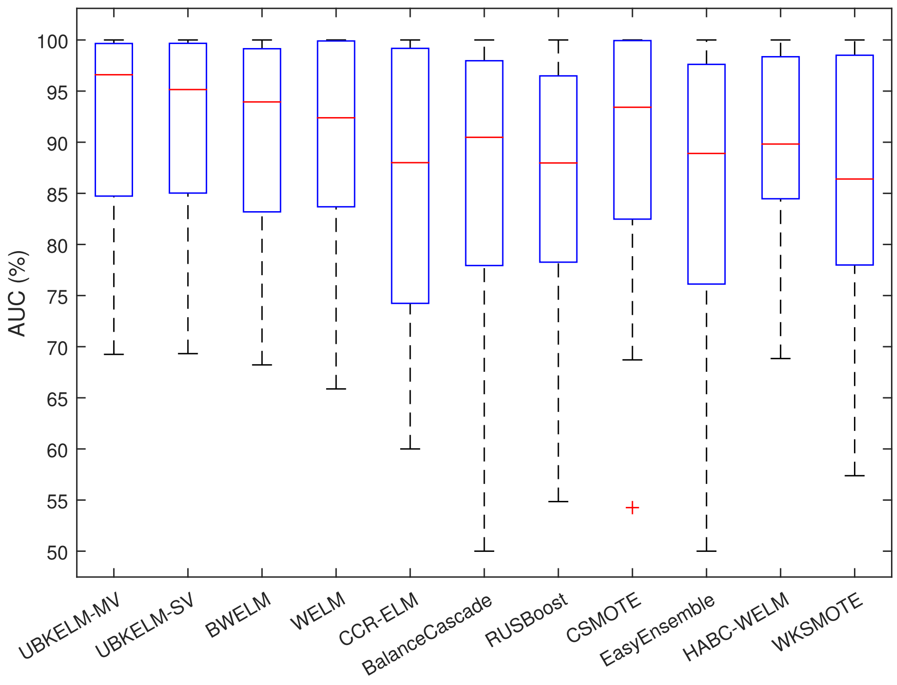


(b)

Fig. 5. Display of the Friedman-aligned mean-ranks for the different classifiers. (a) G-mean (b) AUC.



(a)



(b)

Fig. 6. Display of Box-plot for the different classifiers. (a) G-mean (b) AUC.

As stated in Theorem 1 the SLFNs has $\widetilde{N}$ SPD kernel functions which can approximate $\widetilde{N}$ training samples with zero error. □

3.1.4. *Scalability of UBKELM*

UBKELM solves T balanced problems having $\widetilde{N}$ training samples by using KELM as the component classifier. The value of $\widetilde{N}$ can be computed using Eq. (24). It can be noted that $\widetilde{N}$ is significantly lower than the number of training samples, N for the highly imbalanced classification problems. The proposed method is suitable for the highly imbalanced dataset like abalone19 as illustrated in Fig. 7. The abalone19 dataset has 4174 training samples out of which only 33 samples belong to the minority class. The proposed UBKELM designs 130 component classifiers using balanced training datasets having 66 samples. It can also be observed from the Table 9 the time taken by abalone19 dataset is lower for UBKELM compared to KELM. In other words, KELM takes more training time for a single problem having 4174 training samples than solving 130 problem having 66 training samples. As mentioned above, UBKELM uses KELM algorithm to develops T component classifiers using the balanced datasets having $\widetilde{N}$ training samples. The scalability

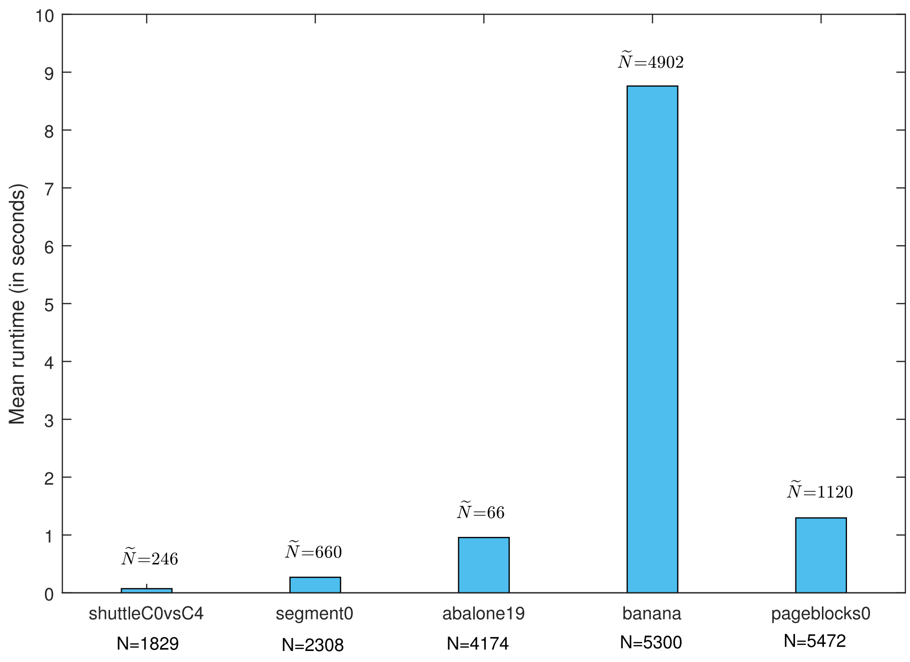


Fig. 7. Scalability of UBKELM for datasets with increasing number of training samples, N.

analysis of the component classifier KELM is given in [15]. It can inferred from the Fig. 7 of UBKELM, that as the value of balanced training samples, $\widetilde{N}$ increases the training time required to develop the component classifier increases. Thus, the proposed method cannot be used to solve the big classification problems having 1 million minority class samples and 1 billion majority class samples.

4. Experimental setup and result analysis

4.1. Dataset specification

The proposed work is assessed using 35 different datasets (30 binary and 5 multiclass classification problems) having a different degree of class imbalance downloaded[1] from KEEL dataset repository [40]. Out of these, some datasets originally belongs to the UCI machine learning repository [41].

To quantitatively measure the imbalance degree of a dataset, the imbalance ratio(IR) is defined as follows.

$$Binary : (IR) = \frac{\#majority\ samples}{\#minority\ samples} \tag{28}$$

$$Multiclass : (IR) = \frac{\max(\#t_k)}{\min(\#t_k)}, k = 1, 2, 3, \ldots m \tag{29}$$

Here, # stands for “number of”. The details of the datasets used are given in Tables 1 and 2. Imbalance dataset given in the table are ordered by high to low (higher IR value indicates a higher degree of imbalanced) imbalance ratio. All the datasets are available in the 5-fold cross-validation format at the KEEL dataset repository. The attributes of the datasets are normalized to the range [1,1], using the following equation.

$$x^{'} = \left(\frac{x - min_n}{max_n - min_n}\right) \times 2 - 1 \tag{30}$$

Here, x is the original attribute value and $x^{'}$ is the normalized attribute value, max_n is the maximum value of feature n and min_n is the minimum value of feature n.

[1] http://sci2s.ugr.es/keel/study.php?cod=24.

4.2. Parameter settings

As the balanced training subsets are created by randomly undersampling of the majority class samples. So, the results might vary due to the change of the training subsets. To study the performance variation, this work averages the performance over 10 independent trials. This work computes the mean accuracy and the standard deviation of the results obtained. This work uses KELM with Gaussian kernel function to map the input data to the feature space. This work presents the best results obtained by varying the regularization parameter, λ on the range $\{2^{-18}, 2^{-16}, \ldots, 2^{48}, 2^{50}\}$. The kernel width parameter, σ is searched on the range $\{2^{-18}, 2^{-16}, \ldots, 2^{18}, 2^{20}\}$ to find the optimal result. The σ variation with the optimal value of λ for different classifiers is shown in the Fig. 2.

4.3. Evaluation metrics

Following are the metrics used to measure the performance of the classification algorithms.

$$\text{Overall accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{31}$$

Here, TP and TN are the correctly classified samples belonging to the positive and the negative class respectively. Whereas, FP and FN are the incorrectly classified samples belonging to the negative and the positive class respectively. For assessing the performance of the classifiers in consideration the following performance metrics are adopted.

$$TPR = \frac{TP}{TP + FN} \quad and \quad TNR = \frac{TN}{TN + FP} \tag{32}$$

Here, TPR represents the true positive rate and TNR represents the true negative rate.

$$Precision = \frac{TP}{TP + FP} \quad and \quad Recall = \frac{TP}{TP + FN} \tag{33}$$

$$\text{G-mean} = \sqrt{TPR \times TNR} \tag{34}$$

Table 1
Specification of binary imbalanced datasets.

Dataset	Number of samples	Number of features	Minority(%)	Majority(%)	Imbalance ratio
abalone19	4174	8	0.77	99.23	128.87
ecoli0137vs26	281	7	2.49	97.51	39.15
yeast1289vs7	947	8	3.17	96.83	30.56
yeast4	1484	8	3.43	96.57	28.81
yeast2vs8	482	8	4.15	95.85	23.10
yeast1458vs7	693	8	4.33	95.67	22.10
shuttleC2vsC4	129	9	4.65	95.35	20.50
abalone9vs18	731	8	5.65	94.25	16.68
pageblocks13vs4	472	10	5.93	94.07	15.85
glass4	214	9	6.07	93.93	15.47
yeast1vs7	459	7	6.72	93.28	13.87
shuttleC0vsC4	1829	9	6.72	93.28	13.87
ecoli4	336	7	6.85	93.15	13.84
ecoli01vs5	240	6	8.33	91.66	11.00
glass2	214	9	8.78	91.22	10.39
glass016vs2	192	9	8.85	91.15	10.29
vowel0	988	13	9.01	90.99	10.10
yeast05679vs4	528	8	9.66	90.34	9.35
glass015vs2	172	9	9.88	90.11	9.12
pageblocks0	5472	10	10.23	89.77	8.77
segment0	2308	19	14.26	85.74	6.01
newthyroid1	215	5	16.28	83.72	5.14
newthyroid2	215	5	16.89	83.11	4.92
glass0123vs456	214	9	23.83	76.17	3.19
haberman	306	3	27.42	73.58	2.68
iris	150	4	33.33	66.67	2.00
pima	768	8	34.84	66.16	1.90
wisconsin	683	9	35.00	65.00	1.86
glass1	214	9	35.51	64.49	1.82
banana	5300	2	46.23	53.77	1.16

Table 2
Specification of multiclass imbalanced datasets.

Dataset	Number of samples	Number of features	Number of classes	Minority (%)	Imbalance ratio
Thyroid	720	21	3	2.36	39.18
Glass	214	9	6	4.20	8.44
Balance	625	4	3	7.84	5.88
Hayesroth	132	4	3	22.73	1.70
Penbased	1100	16	10	9.64	1.10

G-mean for the multiclass problem is defined as follows:

$$\text{G-mean} = \left(\prod_{k=1}^{m} Recall_k \right)^{\frac{1}{m}} \tag{35}$$

Overall accuracy is not a good metric for the class imbalance learning. For example, consider a dataset which has 90 samples belonging to the negative class and 10 samples belonging to the positive class. Assuming that 10 samples of each class are misclassified, the overall accuracy is equal to $\frac{(10-10)+(90-10)}{5+95} = 80\%$. Here, all the positive class samples are misclassified and the overall accuracy is 80 %. G-mean metric is better than the overall accuracy metric when a dataset is imbalanced. For the above example, G-mean is $\sqrt{\frac{(10-10)}{10} \times \frac{(100-10)}{100}} = 0$. Therefore, G-mean is more preferable than overall accuracy to evaluate the effectiveness of classifier for the class imbalance learning. The receiver operating characteristic (ROC) curve [42] computes the classifier performance by varying the threshold value for the classifier score to get different values of true positive rate and false positive rate. In the ROC curve, the X-axis represents the false positive rate and Y-axis represents the true positive rate. The area under the curve (AUC) [43,44] can be used to measure the performance of the model. The perfect model represents coordinates (0, 1) on the ROC curve where all the minority samples and majority samples are correctly classified. The larger the AUC the better the performance of the model.

4.4. Experimental results

The results of the experiments performed for the binary and the multiclass classification problems are given in Tables 3–6. The experimental results for WELM classifier are taken from [4] except for ecoli0137vs26, ecoli01vs5, penbased, balance, thyroid, yeast, glass, hayesroth datasets. This work compares the proposed work with other state-of-the-art ensemble methods like BalanceCascade [25], RUSBoost [30], Critical SMOTE (CSMOTE) [45] an improved version of MSMOTE [46], EasyEnsemble [25] for class imbalance learning. This work uses the MATLAB code[2] available online [45] for evaluating the results of BalanceCascade, RUSBoost, CSMOTE and EasyEnsemble. The experimental results of WKSMOTE [27] in term of G-mean for some of the dataset are taken from [27] and the results for remaining are obtained by experimentation. The AUC for all these classifiers for different datasets is reported in Table 4. The Fig. 3 shows the decision boundary obtained using the optimal parameters for BWELM, WELM, CCR-ELM, UBKELM-MV and UBKELM-SV for the banana dataset. It can be observed from the Fig. 3 that the number of misclassification is less for the proposed classifier contrasted with BWELM, WELM and CCR-ELM. It can also be observed from the Fig. 4 that UBKELM gives better performance than BWELM, WELM, CCR-ELM, BalanceCascade RUSBoost, CSMOTE, EasyEnsemble, HABC-WELM

[2] https://www.dei.unipd.it/node/2357.

Table 3
Performance in terms of G-mean for binary classification.

Dataset	UBKELM															
	MV			SV		BWELM		WELM		CCR-ELM	BalanceCascade	RUSBoost	CSMOTE	EasyEnsemble	HABC-WELM	WKSMOTE
	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%)	(λ, σ)	Testing result(%)	$(\lambda^{+}, \lambda^{-}, \sigma)$	Testing result(%)	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%)
abalone19	$(2^{30}, 2^{20})$	**79.37** ± 0.76	$(2^{14}, 2^{12})$	79.22 ± 0.73	$(2^{28}, 2^{18})$	77.03	$(2^{24}, 2^{16})$	74.47	$(2^{2}, 2^{22}, 2^{2})$	70.26	53.36 ± 7.21	70.78 ± 3.25	40.63 ± 0.00	68.33 ± 3.01	78.90 ± 2.58	75.45
abalone9vs18	$(2^{38}, 2^{18})$	91.53 ± 0.96	$(2^{8}, 2^{4})$	91.07 ± 3.45	$(2^{20}, 2^{8})$	90.12	$(2^{28}, 2^{14})$	89.76	$(2^{2}, 2^{2}, 2^{0})$	76.56	77.68 ± 3.84	86.40 ± 1.33	**92.96** ± 0.32	80.24 ± 1.96	90.55 ± 1.26	91.94
yeast4	$(2^{4}, 2^{-8})$	**85.27** ± 1.34	$(2^{-6}, 2^{-2})$	84.83 ± 1.20	$(2^{12}, 2^{0})$	84.74	$(2^{2}, 2^{-4})$	84.98	$(2^{2}, 2^{-6}, 2^{2})$	71.32	83.12 ± 2.10	79.00 ± 2.59	75.15 ± 2.86	81.50 ± 1.57	81.36 ± 1.25	79.00
ecoli4	$(2^{20}, 2^{14})$	**98.24** ± 1.27	$(2^{4}, 2^{4})$	97.68 ± 0.31	$(2^{26}, 2^{12})$	97.92	$(2^{6}, 2^{0})$	**98.24**	$(2^{8}, 2^{-6}, 2^{0})$	94.92	92.13 ± 1.73	89.30 ± 2.04	88.71 ± 2.03	91.85 ± 2.02	93.04 ± 1.55	92.00
ecoli0137vs26	$(2^{46}, 2^{6})$	77.74 ± 9.20	$(2^{36}, 2^{10})$	81.08 ± 10.56	$(2^{4}, 2^{4})$	78.51	$(2^{-18}, 2^{2})$	73.65	$(2^{2}, 2^{2}, 2^{2})$	74.01	**87.39** ± 6.1	71.37 ± 0.37	75.60 ± 4.12	70.46 ± 9.55	74.16 ± 4.05	74.00
iris0	$(2^{-18}, 2^{-10})$	**100** ± 0.00	$(2^{-18}, 2^{-10})$	**100** ± 0.00	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10}, 2^{8})$	**100**	18.00 ± 5.69	19.85 ± 10.38	**100** ± 0.00	50.00 ± 0.00	**100** ± 0.00	**100**
newthyroid1	$(2^{-2}, 2^{0})$	99.29 ± 0.49	$(2^{-2}, 2^{0})$	99.47 ± 0.13	$(2^{16}, 2^{6})$	**100**	$(2^{14}, 2^{0})$	99.72	$(2^{0}, 2^{2}, 2^{-4})$	99.16	97.85 ± 1.38	98.05 ± 0.95	92.79 ± 1.70	95.08 ± 7.49	96.21 ± 0.75	88.69
newthyroid2	$(2^{6}, 2^{0})$	99.13 ± 0.00	$(2^{-14}, 2^{-4})$	99.30 ± 0.08	$(2^{-18}, 2^{-2})$	**99.72**	$(2^{12}, {}^{-4})$	**99.72**	$(2^{2}, 2^{2}, 2^{-4})$	99.44	95.93 ± 6.12	96.94 ± 0.91	89.64 ± 0.66	96.30 ± 5.55	99.13 ± 2.43	90.72
haberman	$(2^{12}, 2^{8})$	66.70 ± 0.88	$(2^{10}, 2^{6})$	66.49 ± 1.50	$(2^{10}, 2^{4})$	65.14	$(2^{14}, 2^{4})$	66.26	$(2^{46}, 2^{48}, 2^{14})$	59.71	59.47 ± 2.16	53.36 ± 7.21	63.37 ± 0.00	63.20 ± 2.05	**67.59** ± 1.55	65.21
pima	$(2^{30}, 2^{20})$	75.76 ± 0.31	$(2^{-2}, 2^{0})$	**75.84** ± 0.34	$(2^{2}, 2^{6})$	75.48	$(2^{8}, 2^{2})$	75.58	$(2^{2}, 2^{2}, 2^{0})$	73.61	72.19 ± 0.79	70.34 ± 1.45	71.00 ± 0.22	73.63 ± 0.53	75.34 ± 0.63	74.00
wisconsin	$(2^{34}, 2^{20})$	97.72 ± 0.20	$(2^{20}, 2^{14})$	**97.79** ± 0.18	$(2^{8}, 2^{2})$	97.18	$(2^{8}, 2^{4})$	97.70	$(2^{-2}, 2^{2}, 2^{2})$	97.18	97.31 ± 0.32	95.46 ± 0.77	96.86 ± 0.00	95.93 ± 0.35	97.36 ± 0.36	96.33
pageblock0	$(2^{8}, 2^{0})$	93.96 ± 0.19	$(2^{8}, 2^{0})$	93.96 ± 0.24	$(2^{16}, 2^{-2})$	94.19	$(2^{16}, 2^{0})$	93.61	$(2^{6}, 2^{4}, 2^{-2})$	90.96	**96.04** ± 0.28	90.27 ± 0.46	89.01 ± 0.18	92.76 ± 0.21	92.74 ± 1.48	91.14
pageblock13vs4	$(2^{14}, 2^{2})$	**100** ± 0.00	$(2^{44}, 2^{-8})$	**100** ± 0.00	$(2^{4}, 2^{-2})$	99.89	$(2^{20}, 2^{10})$	98.07	$(2^{8}, 2^{0}, 2^{2})$	97.84	89.64 ± 10.04	97.96 ± 1.21	**100** ± 0.00	95.95 ± 2.20	99.44 ± 5.65	97.38
ecoli01vs5	$(2^{10}, 2^{2})$	93.63 ± 0.43	$(2^{10}, 2^{4})$	**94.02** ± 1.07	$(2^{16}, 2^{0})$	89.36	$(2^{10}, 2^{0})$	91.34	$(2^{6}, 2^{10}, 2^{2})$	88.36	89.40 ± 2.61	88.92 ± 1.55	75.48 ± 1.89	90.07 ± 1.46	92.46 ± 1.57	88.00
glass016vs2	$(2^{12}, 2^{8})$	84.48 ± 0.43	$(2^{12}, 2^{2})$	83.89 ± 1.29	$(2^{12}, 2^{8})$	84.21	$(2^{10}, 2^{0})$	83.59	$(2^{6}, 2^{10}, 2^{2})$	81.36	66.52 ± 4.84	52.46 ± 3.04	69.13 ± 2.33	66.52 ± 4.76	**84.68** ± 4.88	79.00
glass015vs2	$(2^{12}, 2^{4})$	**84.45** ± 2.53	$(2^{10}, 2^{2})$	83.96 ± 1.89	$(2^{8}, 2^{12})$	82.47	$(2^{18}, 2^{8})$	82.91	$(2^{6}, 2^{10}, 2^{2})$	69.36	66.40 ± 5.51	48.94 ± 7.33	80.13 ± 0.79	65.92 ± 4.97	81.78 ± 0.95	65.00
glass0123vs456	$(2^{-2}, 2^{-4})$	95.24 ± 0.90	$(2^{-10}, 2^{-4})$	**95.45** ± 0.25	$(2^{18}, 2^{2})$	94.21	$(2^{-18}, 2^{-4})$	95.41	$(2^{10}, 2^{10}, 2^{-2})$	93.26	92.86 ± 1.69	93.74 ± 0.84	85.84 ± 0.64	93.26 ± 0.98	95.01 ± 0.83	94.19
glass1	$(2^{22}, 2^{-8})$	**82.01** ± 8.42	$(2^{46}, 2^{-12})$	81.64 ± 0.61	$(2^{8}, 2^{-4})$	77.68	$(2^{2}, 2^{-4})$	80.35	$(2^{14}, 2^{22}, 2^{2})$	76.60	80.50 ± 1.17	73.02 ± 1.99	60.98 ± 0.89	74.12 ± 0.77	79.30 ± 1.16	73.00
glass2	$(2^{10}, 2^{0})$	**85.94** ± 2.81	$(2^{24}, 2^{18})$	83.26 ± 1.40	$(2^{22}, 2^{8})$	85.50	$(2^{14}, 2^{2})$	82.59	$(2^{16}, 2^{16}, 2^{-2})$	66.59	68.60 ± 5.43	79.76 ± 7.51	80.98 ± 0.12	57.00 ± 3.04	84.19 ± 4.44	79.00
glass4	$(2^{36}, 2^{2})$	92.91 ± 2.82	$(2^{14}, 2^{2})$	92.86 ± 3.30	$(2^{4}, 2^{8})$	90.34	$(2^{8}, 2^{2})$	91.17	$(2^{-18}, 2^{-18}, 2^{-12})$	87.22	89.02 ± 1.08	87.31 ± 2.82	83.81 ± 0.04	92.43 ± 2.63	**96.70** ± 6.85	89.00
vowel0	$(2^{-16}, 2^{-6})$	**100** ± 0.00	$(2^{-16}, 2^{-6})$	**100** ± 0.00	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10}, 2^{2})$	**100**	98.08 ± 0.47	**100** ± 0.21	**100** ± 0.00	98.71 ± 0.32	99.93 ± 0.25	**100**
segment0	$(2^{10}, 2^{4})$	99.63 ± 1.10	$(2^{6}, 2^{2})$	99.64 ± 5.80	$(2^{8}, 2^{2})$	99.87	$(2^{12}, 2^{0})$	99.77	$(2^{-8}, 2^{-8}, 2^{-6})$	99.57	99.77 ± 0.00	99.99 ± 0.00	99.08 ± 0.08	99.43 ± 0.19	99.90 ± 2.30	**100**
shuttleC0vsC4	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{14}, 2^{0})$	**100**	$(2^{26}, 2^{0})$	**100**	$(2^{-18}, 2^{0}, 2^{2})$	**100**	18.99 ± 13.97	60.00 ± 13.33	99.60 ± 0.00	33.96 ± 21.09	**100** ± 0.00	**100**
shuttleC2vsC4	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{-18}, 2^{-12})$	**100**	$(2^{-18}, 2^{0})$	**100**	$(2^{22}, 2^{0}, 2^{2})$	**100**	51.13 ± 14.28	68.50 ± 15.89	23.20 ± 7.37	88.18 ± 9.45	**100** ± 0.00	**100**
yeast05679vs4	$(2^{-16}, 2^{0})$	82.24 ± 0.33	$(2^{-10}, 2^{-8})$	**83.45** ± 2.29	$(2^{6}, 2^{0})$	80.96	$(2^{4}, 2^{-6})$	82.21	$(2^{-16}, 2^{0}, 2^{2})$	82.24	80.70 ± 1.46	77.55 ± 1.63	67.20 ± 2.10	78.48 ± 2.22	82.22 ± 1.98	81.00
yeast1vs7	$(2^{4}, 2^{0})$	77.73 ± 0.00	$(2^{36}, 2^{18})$	77.90 ± 1.54	$(2^{4}, 2^{-4})$	77.72	$(2^{20}, 2^{4})$	77.72	$(2^{8}, 2^{16}, 2^{-2})$	68.32	74.79 ± 1.86	73.49 ± 1.79	76.10 ± 0.00	73.50 ± 3.47	**78.40** ± 2.15	76.00
yeast1458vs7	$(2^{26}, 2^{-4})$	**71.24** ± 1.69	$(2^{6}, 2^{-2})$	70.15 ± 1.31	$(2^{-2}, 2^{-4})$	69.87	$(2^{0}, 2^{-8})$	69.32	$(2^{26}, 2^{-4}, 2^{2})$	66.24	60.28 ± 2.30	59.59 ± 3.43	58.75 ± 1.52	60.95 ± 1.75	70.34 ± 0.25	67.00
yeast1289vs7	$(2^{4}, 2^{0})$	74.28 ± 1.05	$(2^{2}, 2^{0})$	**74.73** ± 1.72	$(2^{42}, 2^{8})$	72.67	$(2^{4}, 2^{-2})$	71.41	$(2^{2}, 2^{2}, 2^{2})$	59.28	65.53 ± 3.47	67.83 ± 2.96	31.55 ± 3.20	70.83 ± 2.35	73.23 ± 3.87	69.83
yeast2vs8	$(2^{18}, 2^{20})$	80.05 ± 2.44	$(2^{-16}, 2^{-2})$	81.69 ± 1.51	$(2^{14}, 2^{0})$	78.35	$(2^{14}, 2^{-4})$	77.89	$(2^{22}, 2^{26}, 2^{-2})$	78.91	75.55 ± 3.07	72.16 ± 1.83	**95.32** ± 0.02	70.71 ± 3.57	78.24 ± 3.14	80.00
banana	$(2^{-18}, 2^{-6})$	89.85 ± 0.80	$(2^{-6}, 2^{-6})$	90.06 ± 0.06	$(2^{2}, 2^{-2})$	89.93	$(2^{12}, 2^{0})$	89.26	$(2^{-8}, 2^{-8}, 2^{-8})$	89.79	88.18 ± 0.11	83.13 ± 1.98	76.10 ± 2.32	88.21 ± 0.21	**90.73** ± 3.15	84.34
Avg. G-mean		87.10		**87.89**		87.01		86.69		81.88	76.90	75.76	77.39	79.23	86.38	83.98

Table 4
Performance in terms of AUC for binary classification.

Dataset	UBKELM															
	MV			SV		BWELM		WELM		CCR-ELM	BalanceCascade	RUSBoost	CSMOTE	EasyEnsemble	HABC-WELM	WKSMOTE
	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%)	(λ, σ)	Testing result(%)	$(\lambda^+, \lambda^-, \sigma)$	Testing result(%)	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%)
abalone19	$(2^{28}, 2^{20})$	84.77 ± 0.21	$(2^{30}, 2^{20})$	**84.95** ± 0.56	$(2^{30}, 2^{18})$	81.53	$(2^{22}, 2^{16})$	78.51	$(2^{40}, 2^{22}, 2^{2})$	69.62	70.38 ± 4.29	77.72 ± 1.21	73.21 ± 0.84	75.79 ± 1.72	78.80 ± 2.58	77.72
abalone9vs18	$(2^{32}, 2^{16})$	**96.79** ± 0.55	$(2^{40}, 2^{20})$	96.55 ± 0.50	$(2^{6}, 2^{4})$	94.13	$(2^{14}, 2^{4})$	95.24	$(2^{18}, 2^{18}, 2^{2})$	83.81	88.16 ± 2.16	93.67 ± 0.87	97.32 ± 0.00	88.32 ± 1.11	89.82 ± 1.27	90.91
yeast4	$(2^{2}, 2^{-2})$	90.87 ± 0.52	$(2^{6}, 2^{-2})$	90.75 ± 0.34	$(2^{2}, 2^{4})$	85.83	$(2^{2}, 2^{0})$	86.71	$(2^{2}, 2^{6}, 2^{-2})$	70.32	**91.55** ± 1.15	87.10 ± 1.57	90.34 ± 0.00	90.84 ± 0.78	86.33 ± 3.21	79.62
ecoli4	$(2^{22}, 2^{20})$	**99.38** ± 0.13	$(2^{26}, 2^{16})$	**99.38** ± 0.31	$(2^{6}, 2^{8})$	98.97	$(2^{6}, 2^{0})$	98.24	$(2^{8}, 2^{-6}, 2^{0})$	94.92	98.43 ± 0.54	96.45 ± 2.09	98.95 ± 0.00	96.97 ± 1.22	97.14 ± 0.91	96.45
ecoli0137vs26	$(2^{40}, 2^{20})$	92.34 ± 1.55	$(2^{2}, 2^{2})$	90.46 ± 5.89	$(2^{44}, 2^{8})$	79.81	$(2^{18}, 2^{0})$	84.09	$(2^{44}, 2^{44}, 2^{8})$	83.45	96.36 ± 5.51	89.01 ± 5.98	95.89 ± 3.26	88.90 ± 5.47	84.73 ± 3.21	**97.63**
iris0	$(2^{-18}, 2^{-10})$	**100** ± 0.00	$(2^{-18}, 2^{-10})$	**100** ± 0.00	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10}, 2^{8})$	**100**	50.00 ± 0.00	54.85 ± 5.12	**100** ± 0.00	50.00 ± 0.00	**100** ± 0.00	**100**
newthyroid1	$(2^{4}, 2^{2})$	**100** ± 0.00	$(2^{4}, 2^{2})$	**100** ± 0.00	$(2^{2}, 2^{2})$	**100**	$(2^{4}, 2^{0})$	**100**	$(2^{2}, 2^{6}, 2^{-2})$	99.60	**100** ± 0.13	99.70 ± 0.51	**100** ± 0.00	98.17 ± 3.82	97.10 ± 0.21	99.71
newthyroid2	$(2^{6}, 2^{4})$	**100** ± 0.00	$(2^{6}, 2^{2})$	**100** ± 0.00	$(2^{2}, 2^{4})$	**100**	$(2^{12}, 2^{-4})$	**100**	$(2^{8}, 2^{8}, 2^{-4})$	99.60	98.80 ± 3.11	99.60 ± 0.21	**100** ± 0.00	98.89 ± 2.71	99.37 ± 2.10	99.92
haberman	$(2^{22}, 2^{12})$	69.25 ± 0.97	$(2^{6}, 2^{4})$	69.32 ± 1.46	$(2^{38}, 2^{14})$	68.22	$(2^{50}, 2^{18})$	67.81	$(2^{48}, 2^{50}, 2^{2})$	63.91	64.14 ± 1.75	**70.38** ± 4.29	68.71 ± 0.14	69.90 ± 1.72	68.84 ± 1.03	67.34
pima	$(2^{4}, 2^{4})$	79.87 ± 0.42	$(2^{-2}, 2^{0})$	80.55 ± 0.48	$(2^{2}, 2^{4})$	79.10	$(2^{6}, 2^{0})$	78.30	$(2^{4}, 2^{4}, 2^{0})$	74.14	80.30 ± 0.56	79.91 ± 0.93	**81.55** ± 0.00	81.02 ± 0.40	78.86 ± 0.55	79.18
wisconsin	$(2^{8}, 2^{6})$	98.72 ± 0.15	$(2^{26}, 2^{16})$	99.08 ± 0.14	$(2^{16}, 2^{12})$	98.15	$(2^{20}, 2^{10})$	98.75	$(2^{16}, 2^{12}, 2^{2})$	98.15	99.23 ± 0.00	98.37 ± 0.31	**99.85** ± 0.00	98.97 ± 0.17	98.18 ± 0.25	98.80
pageblock0	$(2^{8}, 2^{0})$	96.60 ± 0.18	$(2^{8}, 2^{0})$	97.46 ± 0.32	$(2^{8}, 2^{0})$	96.98	$(2^{16}, 2^{0})$	95.45	$(2^{6}, 2^{4}, 2^{-2})$	90.45	**99.12** ± 0.10	96.50 ± 0.17	97.93 ± 0.00	97.79 ± 0.10	95.44 ± 4.11	96.35
pageblock13vs4	$(2^{14}, 2^{2})$	**100** ± 0.00	$(2^{4}, 2^{0})$	99.68 ± 0.28	$(2^{-8}, 2^{-8})$	98.00	$(2^{8}, 2^{0})$	**100**	$(2^{8}, 2^{8}, 2^{0})$	98.00	95.23 ± 5.47	99.86 ± 0.20	99.97 ± 0.00	99.47 ± 1.01	99.96 ± 2.27	99.96
ecoli01vs5	$(2^{16}, 2^{8})$	94.90 ± 2.34	$(2^{10}, 2^{4})$	94.13 ± 2.25	$(2^{46}, 2^{6})$	93.94	$(2^{10}, 2^{0})$	92.39	$(2^{6}, 2^{10}, 2^{2})$	89.50	**97.97** ± 0.84	93.94 ± 2.37	95.00 ± 0.33	95.89 ± 1.46	95.48 ± 2.68	96.22
glass016vs2	$(2^{10}, 2^{4})$	86.07 ± 0.46	$(2^{8}, 2^{2})$	**86.74** ± 1.58	$(2^{6}, 2^{-2})$	85.43	$(2^{10}, 2^{0})$	84.11	$(2^{6}, 2^{10}, 2^{2})$	81.36	74.88 ± 4.35	59.25 ± 4.34	82.33 ± 5.46	73.09 ± 3.70	84.41 ± 4.32	83.52
glass015vs2	$(2^{12}, 2^{4})$	84.45 ± 2.53	$(2^{10}, 2^{2})$	83.96 ± 1.89	$(2^{8}, 2^{2})$	**86.43**	$(2^{18}, 2^{8})$	84.91	$(2^{6}, 2^{10}, 2^{2})$	79.36	78.23 ± 3.20	58.70 ± 3.90	83.92 ± 0.56	73.98 ± 2.15	85.64 ± 2.94	81.80
glass0123vs456	$(2^{0}, 2^{-2})$	97.36 ± 0.80	$(2^{-16}, 2^{-4})$	97.35 ± 0.00	$(2^{6}, 2^{4})$	96.57	$(2^{8}, 2^{2})$	97.03	$(2^{10}, 2^{10}, 2^{-2})$	93.84	97.97 ± 0.29	97.48 ± 0.72	**98.86** ± 0.00	97.08 ± 0.82	96.93 ± 1.33	**98.86**
glass1	$(2^{28}, 2^{-8})$	81.00 ± 1.25	$(2^{6}, 2^{0})$	83.60 ± 0.65	$(2^{14}, 2^{2})$	80.67	$(2^{2}, 2^{-4})$	82.86	$(2^{14}, 2^{22}, 2^{2})$	76.60	**89.41** ± 0.79	80.70 ± 1.65	78.87 ± 0.29	79.96 ± 1.27	76.27 ± 0.75	80.15
glass2	$(2^{10}, 2^{2})$	87.08 ± 1.71	$(2^{24}, 2^{14})$	**88.95** ± 0.20	$(2^{8}, 2^{0})$	86.90	$(2^{12}, 2^{0})$	84.71	$(2^{18}, 2^{18}, 2^{0})$	70.06	79.84 ± 4.13	88.09 ± 3.00	86.87 ± 1.96	67.66 ± 3.85	85.58 ± 4.50	88.80
glass4	$(2^{22}, 2^{2})$	97.38 ± 0.19	$(2^{50}, 2^{-2})$	97.83 ± 0.00	$(2^{8}, 2^{4})$	96.37	$(2^{8}, 2^{2})$	93.47	$(2^{8}, 2^{8}, 2^{-2})$	88.00	93.42 ± 0.56	96.18 ± 2.78	97.45 ± 1.42	94.42 ± 0.89	**98.42** ± 9.35	94.86
vowel0	$(2^{-16}, 2^{-6})$	**100** ± 0.00	$(2^{-16}, 2^{-6})$	**100** ± 0.00	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10})$	**100**	$(2^{-18}, 2^{-10}, 2^{2})$	**100**	**100** ± 0.00	**100** ± 0.00	**100** ± 0.00	**100** ± 0.00	**100** ± 0.00	57.38
segment0	$(2^{2}, 2^{0})$	99.84 ± 0.00	$(2^{6}, 2^{2})$	99.64 ± 5.80	$(2^{-8}, 2^{-4})$	99.93	$(2^{10}, 2^{0})$	99.85	$(2^{6}, 2^{10}, 2^{-2})$	99.80	**100** ± 0.00	**100** ± 0.00	**100** ± 0.00	99.98 ± 0.00	99.89 ± 1.25	99.91
shuttleC0vsC4	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{14}, 2^{0})$	**100**	$(2^{26}, 2^{0})$	**100**	$(2^{-18}, 2^{0}, 2^{2})$	**100**	58.99 ± 6.98	80.00 ± 6.67	**100** ± 0.00	67.00 ± 10.59	**100** ± 0.00	**100**
shuttleC2vsC4	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{-18}, 2^{0})$	**100** ± 0.00	$(2^{26}, 2^{4})$	99.20	$(2^{-18}, 2^{0})$	**100**	$(2^{22}, 2^{0}, 2^{2})$	**100**	77.84 ± 5.96	81.91 ± 7.10	**100** ± 0.00	94.15 ± 5.23	**100** ± 0.00	**100**
yeast05679vs4	$(2^{-4}, 2^{0})$	85.86 ± 0.84	$(2^{30}, 2^{-2})$	88.75 ± 1.07	$(2^{2}, 2^{10})$	84.08	$(2^{2}, 2^{-4})$	85.72	$(2^{8}, 2^{8}, 2^{0})$	74.54	**90.48** ± 0.63	87.97 ± 1.02	87.28 ± 0.08	88.30 ± 0.80	84.81 ± 1.73	78.80
yeast1vs7	$(2^{4}, 2^{4})$	84.06 ± 1.29	$(2^{26}, 2^{14})$	86.09 ± 0.58	$(2^{2}, 2^{8})$	82.89	$(2^{6}, 2^{-4})$	81.00	$(2^{8}, 2^{16}, 2^{-2})$	72.15	79.47 ± 2.21	**86.53** ± 2.05	85.42 ± 5.11	81.44 ± 1.98	81.36 ± 2.34	82.89
yeast1458vs7	$(2^{22}, 2^{-4})$	72.98 ± 2.11	$(2^{8}, 2^{-4})$	73.08 ± 1.37	$(2^{-2}, 2^{-2})$	70.72	$(2^{14}, 2^{8})$	70.63	$(2^{26}, 2^{28}, 2^{2})$	61.76	70.82 ± 2.51	65.94 ± 2.14	69.74 ± 3.16	68.28 ± 1.71	71.93 ± 2.49	**74.91**
yeast1289vs7	$(2^{4}, 2^{0})$	80.52 ± 0.08	$(2^{-8}, 2^{-4})$	80.59 ± 0.07	$(2^{2}, 2^{2})$	78.44	$(2^{4}, 2^{0})$	78.73	$(2^{22}, 2^{26}, 2^{2})$	65.45	74.07 ± 4.21	74.91 ± 1.81	**81.28** ± 0.56	78.72 ± 1.27	76.87 ± 2.84	77.51
yeast2vs8	$(2^{6}, 2^{2})$	84.08 ± 0.12	$(2^{20}, 2^{-8})$	84.35 ± 0.00	$(2^{2}, 2^{2})$	84.42	$(2^{18}, 2^{0})$	83.55	$(2^{22}, 2^{26}, 2^{-2})$	79.39	**85.81** ± 2.63	79.92 ± 2.45	85.19 ± 0.00	82.27 ± 1.79	84.73 ± 2.13	85.56
banana	$(2^{28}, 2^{2})$	92.68 ± 0.00	$(2^{14}, 2^{-2})$	93.53 ± 0.00	$(2^{6}, 2^{0})$	91.43	$(2^{0}, 2^{-4})$	90.47	$(2^{-8}, 2^{-4}, 2^{-8})$	91.71	95.64 ± 0.00	91.16 ± 0.84	82.92 ± 1.40	**95.65** ± 0.00	92.31 ± 5.26	86.40
Avg. AUC		**92.59**		92.55		90.52		90.01		85.57	86.43	85.67	89.64	86.54	89.90	85.81

Table 5
Performance in terms of G-mean for multiclass classification.

Dataset	UBKELM				BWELM		WELM		CCR-ELM		BalanceCascade	RUSBoost	CSMOTE	EasyEnsemble	HABC-WELM	WKSMOTE
	MV		SV													
	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%)	(λ, σ)	Testing result(%)	$(\lambda^+, \lambda^-, \sigma)$	Testing result(%)	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%)
glass	$(2^{10}, 2^{2})$	70.37 ± 2.95	$(2^{8}, 2^{6})$	70.83 ± 8.30	$(2^{46}, 2^{20})$	68.68	$(2^{10}, 2^{4})$	70.34	$(2^{26}, 2^{22}, 2^{8})$	52.37	**89.73** ± 5.07	62.67 ± 2.43	86.19 ± 3.60	82.70 ± 9.15	69.94 ± 6.10	79.83
thyroid	$(2^{6}, 2^{4})$	78.37 ± 2.25	$(2^{40}, 2^{8})$	77.77 ± 3.66	$(2^{46}, 2^{18})$	76.60	$(2^{24}, 2^{12})$	72.02	$(2^{8}, 2^{6}, 2^{4})$	68.37	**84.10** ± 9.95	61.26 ± 6.59	74.56 ± 0.00	68.86 ± 3.78	70.46 ± 2.29	77.87
penbased	$(2^{14}, 2^{8})$	98.25 ± 0.20	$(2^{22}, 2^{10})$	98.36 ± 0.11	$(2^{12}, 2^{2})$	98.18	$(2^{-18}, 2^{-2})$	97.20	$(2^{2}, 2^{2}, 2^{2})$	98.16	98.62 ± 0.41	65.90 ± 8.32	99.40 ± 0.00	98.49 ± 0.16	93.47 ± 0.64	**99.82**
balance	$(2^{6}, 2^{8})$	82.68 ± 0.00	$(2^{18}, 2^{8})$	**82.92** ± 0.30	$(2^{10}, 2^{16})$	82.90	$(2^{32}, 2^{-8})$	82.82	$(2^{46}, 2^{44}, 2^{8})$	50.87	57.12 ± 1.88	58.83 ± 3.22	24.54 ± 3.26	69.09 ± 1.71	72.47 ± 3.28	42.21
hayesroth	$(2^{6}, 2^{4})$	83.65 ± 0.46	$(2^{6}, 2^{4})$	83.92 ± 1.51	$(2^{2}, 2^{14})$	86.00	$(2^{10}, 2^{4})$	85.08	$(2^{4}, 2^{6}, 2^{2})$	83.98	75.69 ± 16.21	97.50 ± 2.60	84.86 ± 2.19	96.44 ± 1.77	84.21 ± 3.12	**98.67**

Table 6
Performance in terms of AUC for multiclass classification.

Dataset	UBKELM				BWELM		WELM		CCR-ELM		BalanceCascade	RUSBoost	CSMOTE	EasyEnsemble	HABC-WELM	WKSMOTE
	MV		SV													
	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%) ± std	(λ, σ)	Testing result(%)	(λ, σ)	Testing result(%)	$(\lambda^+, \lambda^-, \sigma)$	Testing result(%)	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%) ± std	Testing result(%)
glass	$(2^{24}, 2^{10})$	**98.59** ± 1.05	$(2^{24}, 2^{10})$	98.09 ± 1.21	$(2^{48}, 2^{10})$	96.61	$(2^{50}, 2^{12})$	96.84	$(2^{28}, 2^{32}, 2^{6})$	97.34	96.82 ± 2.11	90.47 ± 5.09	93.42 ± 3.79	94.08 ± 2.57	82.32 ± 5.50	87.83
thyroid	$(2^{28}, 2^{20})$	**98.07** ± 0.21	$(2^{32}, 2^{10})$	98.04 ± 0.23	$(2^{22}, 2^{8})$	90.32	$(2^{24}, 2^{12})$	88.10	$(2^{24}, 2^{28}, 2^{4})$	81.77	93.82 ± 5.37	70.28 ± 3.29	88.55 ± 0.00	77.13 ± 3.13	84.65 ± 2.56	69.94
penbased	$(2^{48}, 2^{18})$	99.90 ± 0.00	$(2^{40}, 2^{20})$	99.90 ± 0.00	$(2^{-2}, 2^{-2})$	99.52	$(2^{50}, 2^{18})$	99.92	$(2^{-18}, 2^{-18}, 2^{-2})$	99.52	99.94 ± 0.00	83.12 ± 5.31	**99.98** ± 0.00	99.23 ± 0.32	98.56 ± 1.60	86.52
balance	$(2^{40}, 2^{8})$	85.42 ± 0.11	$(2^{16}, 2^{4})$	84.23 ± 1.07	$(2^{48}, 2^{10})$	80.00	$(2^{32}, 2^{14})$	65.87	$(2^{50}, 2^{50}, 2^{6})$	60.00	60.26 ± 1.66	74.79 ± 1.46	54.26 ± 1.21	75.55 ± 1.62	**86.42** ± 2.02	63.87
hayesroth	$(2^{8}, 2^{4})$	**100** ± 0.00	$(2^{12}, 2^{8})$	**100** ± 0.00	$(2^{4}, 2^{2})$	**100**	$(2^{8}, 2^{2})$	**100**	$(2^{4}, 2^{6}, 2^{2})$	**100**	87.81 ± 8.21	99.86 ± 0.20	72.13 ± 0.86	99.46 ± 0.86	96.57 ± 3.46	60.15

Table 7
The Wilcoxon signed-rank test results based on the G-mean. Here, R^+ is the sum of ranks for the datasets in which the first algorithm outperforms the second and R^- is the sum of ranks for the opposite.

Comparison	R^+	R^-	p-value	Hypothesis (0.05)	Comparison	R^+	R^-	p-value	Hypothesis (0.05)
UBKELM-MV vs. BWELM	358.0	138.0	0.0311	**Rejected**	UBKELM-SV vs. BWELM	381.5	114.5	0.0089	**Rejected**
UBKELM-MV vs. WELM	376.5	88.5	0.0031	**Rejected**	UBKELM-SV vs. WELM	384.5	80.5	0.0018	**Rejected**
UBKELM-MV vs. CCR-ELM	418.0	17.0	$1.4538e^{-05}$	**Rejected**	UBKELM-SV vs. CCR-ELM	494.0	2.0	$1.4302e^{-06}$	**Rejected**
UBKELM-MV vs. BalanceCascade	545.0	85.0	$1.6509e^{-04}$	**Rejected**	UBKELM-SV vs. BalanceCascade	544.0	86.0	$1.7625e^{-04}$	**Rejected**
UBKELM-MV vs. RUSBoost	566.0	29.0	$4.4237e^{-06}$	**Rejected**	UBKELM-SV vs. RUSBoost	586.0	9.0	$8.1248e^{-07}$	**Rejected**
UBKELM-MV vs. CSMOTE	483.0	78.0	$2.9662e^{-04}$	**Rejected**	UBKELM-SV vs. CSMOTE	496.0	65.0	$1.1787e^{-04}$	**Rejected**
UBKELM-MV vs. EasyEnsemble	568.0	62.0	$3.4141e^{-05}$	**Rejected**	UBKELM-SV vs. EasyEnsemble	589.0	41.0	$7.1934e^{-06}$	**Rejected**
UBKELM-MV vs. HABC-WELM	396.0	100.0	0.0037	**Rejected**	UBKELM-SV vs. HABC-WELM	410.5	117.5	0.0062	**Rejected**
UBKELM-MV vs. WKSMOTE	420.5	75.5	$7.2358e^{-04}$	**Rejected**	UBKELM-SV vs. WKSMOTE	440.0	56.0	$1.6820e^{-04}$	**Rejected**

Table 8
The Wilcoxon signed-rank test results based on the AUC. Here, R^+ is the sum of ranks for the datasets in which the first algorithm outperforms the second and R^- is the sum of ranks for the opposite.

Comparison	R^+	R^-	p-value	Hypothesis (0.05)	Comparison	R^+	R^-	p-value	Hypothesis (0.05)
UBKELM-MV vs. BWELM	381.5	53.5	$3.9067e^{-04}$	**Rejected**	UBKELM-SV vs. BWELM	395.0	40.0	$1.2398e^{-04}$	**Rejected**
UBKELM-MV vs. WELM	355.0	51.0	$5.3766e^{-04}$	**Rejected**	UBKELM-SV vs. WELM	380.5	25.5	$5.2984e^{-05}$	**Rejected**
UBKELM-MV vs. CCR-ELM	465.0	0.0	$1.7333e^{-06}$	**Rejected**	UBKELM-SV vs. CCR-ELM	464.0	1.0	$1.9185e^{-06}$	**Rejected**
UBKELM-MV vs. BalanceCascade	420.0	141.0	0.0127	**Rejected**	UBKELM-SV vs. BalanceCascade	417.0	144.0	0.0147	**Rejected**
UBKELM-MV vs. RUSBoost	566.0	76.0	$1.5251e^{-04}$	**Rejected**	UBKELM-SV vs. RUSBoost	559.0	36.0	$7.7950e^{-06}$	**Rejected**
UBKELM-MV vs. CSMOTE	250.5	184.5	0.4754	Not rejected	UBKELM-SV vs. CSMOTE	279.0	156.0	0.1836	Not Rejected
UBKELM-MV vs. EasyEnsemble	497.0	98.0	$6.4783e^{-04}$	**Rejected**	UBKELM-SV vs. EasyEnsemble	529.0	66.0	$7.5630e^{-05}$	**Rejected**
UBKELM-MV vs. HABC-WELM	416.0	80.0	$9.9400e^{-04}$	**Rejected**	UBKELM-SV vs. HABC-WELM	428.0	68.0	$4.1970e^{-04}$	**Rejected**
UBKELM-MV vs. WKSMOTE	425.5	102.5	0.0025	**Rejected**	UBKELM-SV vs. WKSMOTE	442.5	85.5	$8.4434e^{-04}$	**Rejected**

Table 9
The mean runtime (MRT) (in seconds) for the imbalanced problems.

Dataset (IR)	UBKELM	BWELM	KELM	ELM	WELM	CCR-ELM	BalanceCascade	RUSBoost	CSMOTE	EasyEnsemble	HABC-WELM	WKSMOTE
abalone19(128.87)	0.9569	166.9090	1.6035	**0.5612**	9.0122	1.8243	5.7848	3.8221	5.0417	3.6524	637.4956	6.0858
yeast4 (28.81)	**0.0446**	2.5986	0.1224	0.2445	0.5291	0.1401	4.4556	0.5787	0.8894	0.5961	99.6794	1.1686
yeast2vs8 (23.10)	**0.0193**	0.6773	0.0201	0.1139	0.0312	0.0201	1.9828	0.7549	0.1865	1.6257	42.4012	0.0569
abalone9vs18 (16.68)	**0.0235**	1.4002	0.0290	0.1419	0.0784	0.0294	1.8014	0.2301	0.3369	2.2280	53.5670	0.1833
yeast1vs7 (13.87)	**0.0286**	0.2757	0.0315	0.1134	0.0272	0.0425	1.7897	0.7821	0.1768	1.8260	41.2768	0.1485
shuttleC0vsC4 (13.87)	**0.0727**	3.0708	0.2363	0.2898	0.9090	0.2363	1.6760	0.1969	1.2968	0.5022	193.6881	1.2916
vowel0 (10.10)	**0.0567**	8.9212	0.0568	0.1802	0.1795	0.0568	4.2168	0.9988	0.6805	0.6792	66.6374	0.8107
yeast05679vs4 (9.35)	**0.0133**	0.3879	0.0179	0.1186	0.0369	0.0149	1.4938	0.8673	0.2246	0.5592	43.5549	0.1618
pageblocks0 (8.77)	1.2955	410.6249	3.6028	**0.7870**	22.7756	3.8258	110.1193	35.5376	8.5573	61.7882	997.1405	9.8182
segment0 (6.01)	**0.2691**	51.3260	0.4950	0.3543	1.8420	0.4986	22.5037	4.5051	2.2903	9.3957	277.8248	2.5651
haberman (2.68)	0.0055	0.0508	**0.0029**	0.1109	0.0081	0.0049	1.5652	0.1142	0.0927	0.4762	31.2439	0.1376
pima (1.90)	0.0501	0.7433	**0.0296**	0.1590	0.0863	0.0314	3.5683	0.2811	0.3592	1.8347	56.1573	0.3317
banana (1.16)	8.7605	305.4290	3.4290	**0.9037**	20.2836	3.7015	620.7466	42.3283	6.4599	309.4667	942.8051	8.2405

and WKSMOTE. For further comparison of the performance, this work uses Wilcoxon signed-rank test [47]. The statistical test results obtained using G-mean and AUC are shown in Tables 7 and 8 respectively. The threshold value for these tests is set to 0.05. If the p-value is less than 0.05, then there is a significant difference between the two algorithms. The smaller the p-value, the difference is more statistically significant. The mean ranking of each algorithm is also obtained using the Friedman mean-ranks test [47] for the G-mean and AUC scores of the classifiers in consideration. The proposed UBKELM algorithm was chosen as the control method, as it achieves the lowest mean-ranks in the Friedman test which is shown in the Fig. 5. In additions, this work also presents Box plots for the distribution of the G-mean and AUC of these classifiers which are illustrated in the Fig. 6. The computational cost of UBKELM is significantly lower than BWELM, BalanceCascade, EasyEnsemble, hybrid artificial bee colony WELM which can be observed from the Table 9.

5. Conclusion

The imbalanced problems are widely present in the real-world application. However, most traditional classification algorithms are biased towards the majority class as they are originally designed to deal with the balanced classification problems. This work proposes and evaluates an ensemble based solution to handle the class imbalance problem more effectively. This work proposes and evaluates an UnderBagging ensemble based variants of a kernelized extreme learning machine. The existing variants of ELM designed to handle class imbalance problem assign weights to the training samples. This work develops an algorithm which does not assign any weights to the training samples. The outcomes of these component classifiers are combined using majority voting and soft voting. The proposed algorithm is evaluated using datasets downloaded from KEEL dataset repository. The results show that the proposed work outperforms the rest of the classifiers for most of the evaluated datasets. The computational cost of UBKELM is significantly lower than BWELM, BalanceCascade, EasyEnsemble, hybrid artificial bee colony WELM. The superiority of UBKELM is also revealed by the Wilcoxon signed-rank test and the Friedman mean-ranks test. The computational cost required for datasets with a large number of minority samples is more as the UBKELM utilizes the Moore-Penrose pseudoinverse for determining the output weight matrix. The UBKELM has a computational cost of the order of $O(\widetilde{N}^3)$. As $\widetilde{N}$ gets larger, the computational cost of UBKELM become prohibitive. In the future, we will continue to investigate our work on how to handle the large imbalance problem more effectively. From this aspects, the proposed method includes the extension of UBKELM as online sequential UBKELM to address the large imbalance classification problem more effectively. The future work also includes evaluating other ensemble methods to address the class imbalance problem more effectively.

References

[1] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (13) (2006) 489–501.
[2] V.M. Janakiraman, X. Nguyen, J. Sterniak, D. Assanis, Identification of the dynamic operating envelope of HCCI engines using class imbalance learning, IEEE Trans. Neural Netw. Learn. Syst. 26 (1) (2015) 98–112.
[3] V.M. Janakiraman, X. Nguyen, D. Assanis, Stochastic gradient based extreme learning machines for stable online learning of advanced combustion engines, Neurocomputing 177 (2016) 304–316.
[4] W. Zong, G.-B. Huang, Y. Chen, Weighted extreme learning machine for imbalance learning, Neurocomputing 101 (2013) 229–242.
[5] K. Li, X. Kong, Z. Lu, L. Wenyin, J. Yin, Boosting weighted {ELM} for imbalanced learning, Neurocomputing 128 (2014) 15–21.
[6] B.S. Raghuwanshi, S. Shukla, Class-specific cost-sensitive boosting weighted elm for class imbalance learning, Memet. Comput. (2018).
[7] Y. Zhang, B. Liu, J. Cai, S. Zhang, Ensemble weighted extreme learning machine for imbalanced data classification based on differential evolution, Neural Comput. Appl. 28 (2016) 259–267.
[8] S. Shukla, R.N. Yadav, Regularized weighted circular complex-valued extreme learning machine for imbalanced learning, IEEE Access 3 (2015) 3048–3057.
[9] B.S. Raghuwanshi, S. Shukla, Underbagging based reduced kernelized weighted extreme learning machine for class imbalance learning, Eng. Appl. Artif. Intell. 74 (2018) 252–270.
[10] W. Xiao, J. Zhang, Y. Li, S. Zhang, W. Yang, Class-specific cost regulation extreme learning machine for imbalanced classification, Neurocomputing 261 (2017) 70–82.
[11] B.S. Raghuwanshi, S. Shukla, Class-specific extreme learning machine for handling binary class imbalance problem, Neural Netw. 105 (2018) 206–217.
[12] A. Iosifidis, M. Gabbouj, On the kernel extreme learning machine speedup, Pattern Recognit. Lett. 68 (2015) 205–210.
[13] A. Iosifidis, A. Tefas, I. Pitas, On the kernel extreme learning machine classifier, Pattern Recognit. Lett. 54 (2015) 11–17.
[14] R.E. Schapire, A brief introduction to boosting, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2, in: IJCAI'99, 1999, pp. 1401–1406.
[15] G.B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst. Man Cybern. Part B (Cybern.) 42 (2) (2012) 513–529.
[16] J. Cao, Z. Lin, G.-B. Huang, N. Liu, Voting based extreme learning machine, Inf. Sci. 185 (1) (2012) 66–77.
[17] Z. Zhou, Ensemble Methods: Foundations and Algorithms, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, Taylor & Francis, 2012.
[18] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, G. Bing, Learning from class-imbalanced data: review of methods and applications, Expert Syst. Appl. 73 (2017) 220–239.
[19] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[20] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics, Inf. Sci. 250 (2013) 113–141.
[21] I. Brown, C. Mues, An experimental comparison of classification algorithms for imbalanced credit scoring data sets, Expert Syst. Appl. 39 (3) (2012) 3446–3453.
[22] J. Xiao, L. Xie, C. He, X. Jiang, Dynamic classifier ensemble model for customer classification with imbalanced class distribution, Expert Syst. Appl. 39 (3) (2012) 3668–3675.
[23] B. Krawczyk, M. Galar, L. Jele, F. Herrera, Evolutionary under-sampling boosting for imbalanced classification of breast cancer malignancy, Appl. Soft. Comput. 38 (C) (2016) 714–726.
[24] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches, IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) 42 (4) (2012) 463–484.
[25] X.Y. Liu, J. Wu, Z.H. Zhou, Exploratory under-sampling for class-imbalance learning, IEEE Trans. Syst. Man Cybern. Part B (Cybern.) 39 (2) (2009) 539–550.
[26] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artif. Int. Res. 16 (1) (2002) 321–357.
[27] J. Mathew, C.K. Pang, M. Luo, W.H. Leong, Classification of imbalanced data by oversampling in kernel space of support vector machines, IEEE Trans. Neural Netw. Learn. Syst. (2018) 1–12.
[28] X. YANG, Q. SONG, Y. WANG, A weighted support vector machine for data classification, Int. J. Pattern Recognit. Artif. Intell. 21 (05) (2007) 961–976.
[29] T.G. Dietterich, Ensemble methods in machine learning, in: Multiple Classifier Systems, Springer, 2000, pp. 1–15.
[30] C. Seiffert, T.M. Khoshgoftaar, J.V. Hulse, A. Napolitano, Rusboost: a hybrid approach to alleviating class imbalance, IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. 40 (1) (2010) 185–197.
[31] N.V. Chawla, A. Lazarevic, L.O. Hall, K.W. Bowyer, Smoteboost: improving prediction of the minority class in boosting, in: N. Lavrač, D. Gamberger, L. Todorovski, H. Blockeel (Eds.), Knowledge Discovery in Databases, PKDD, 2003, pp. 107–119.
[32] W. Deng, Q. Zheng, L. Chen, Regularized extreme learning machine, IEEE Symp. Comput. Intell. Data Min. (2009) 389–395.
[33] Q.-Y. Zhu, A. Qin, P. Suganthan, G.-B. Huang, Evolutionary extreme learning machine, Pattern Recognit. 38 (10) (2005) 1759–1763.
[34] Y.-P. Zhao, Parsimonious kernel extreme learning machine in primal via Cholesky factorization, Neural Netw. 80 (2016) 95–109.
[35] C.-X. You, J.-Q. Huang, F. Lu, Recursive reduced kernel based extreme learning machine for aero-engine fault pattern recognition, Neurocomputing 214 (2016) 1038–1045.
[36] F. Luo, W. Guo, Y. Yu, G. Chen, A multi-label classification algorithm based on kernel extreme learning machine, Neurocomputing 260 (2017) 313–320.
[37] Y. Zeng, X. Xu, D. Shen, Y. Fang, Z. Xiao, Traffic sign recognition using kernel extreme learning machines with deep perceptual features, IEEE Trans. Intell. Transp. Syst. 18 (6) (2017) 1647–1653.
[38] W.-Y. Deng, Y.-S. Ong, P.S. Tan, Q.-H. Zheng, Online sequential reduced kernel extreme learning machine, Neurocomputing 174 (2016) 72–84.
[39] G.-F. Fasshauer, Meshfree Approximation Methods with MATLAB, World Scientific Publishing Co., Inc, 2007.
[40] J. Alcalá, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework, J. Multiple-Valued Logic Soft Comput. 17 (2–3) (2010) 255–287.
[41] D. Dua, E.K. Taniskidou, UCI Machine Learning Repository, University of California, School of Information and Computer Science, Irvine, CA, 2017 http://archive.ics.uci.edu/ml.
[42] A.P. Bradley, The use of the area under the roc curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.
[43] T. Fawcett, Technical Report, HPL-2003-4, HP Labs, 2003.
[44] J. Huang, C.X. Ling, Using AUC and accuracy in evaluating learning algorithms, IEEE Trans. Knowl. Data Eng. 17 (3) (2005) 299–310.
[45] L. Nanni, C. Fantozzi, N. Lazzarini, Coupling different methods for overcoming the class imbalance problem, Neurocomputing 158 (C) (2015) 48–61.
[46] S. Hu, Y. Liang, L. Ma, Y. He, Msmote: improving classification performance when training data is imbalanced, in: Proceedings of the 2009 Second International Workshop on Computer Science and Engineering, 2, 2009, pp. 13–17.
[47] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.

Bhagat Singh Raghuwanshi, received the Bachelor of Engineering (B.E.) degree in computer science and engineering from the Lakshmi Narain College of Technology (LNCT), Bhopal, India, in 2009, and the Master of Technology (M.Tech) from the Department of Computer Science and Engineering, Maulana Azad National Institute of Technology, Bhopal, in 2012. Since 2016, he has been Ph.D. scholar in Department of Computer Science and Engineering, Maulana Azad National Institute of Technology Bhopal (MANIT). His research interests include Extreme Learning Machine, Neural Networks, and Machine Learning.

Dr. Sanyam Shukla, received the Bachelor of Engineering (B.E.) degree in computer science and engineering from the Shri Govindram Seksaria Institute of Technology and Science, Indore, India, in 2002, and the Master of Technology (M.Tech) and Ph.D. degrees from the Department of Computer Science and Engineering, Maulana Azad National Institute of Technology, Bhopal, in 2009 and 2015, respectively. Since 2005, he has been an Assistant Professor with the Department of Computer Science and Engineering, Maulana Azad National Institute of Technology. He has authored over ten papers in journals and conference proceedings. His research interests include data mining, neural networks, machine learning, optimization techniques, and evolutionary algorithms.