Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 358942, 14 pages
http://dx.doi.org/10.1155/2014/358942

Research Article

A Novel Selective Ensemble Algorithm for Imbalanced Data Classification Based on Exploratory Undersampling

Qing-Yan Yin, Jiang-She Zhang, Chun-Xia Zhang, and Nan-Nan Ji

School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an 710049, China

Correspondence should be addressed to Jiang-She Zhang; jszhang@mail.xjtu.edu.cn

Received 22 November 2013; Revised 23 January 2014; Accepted 14 February 2014; Published 30 March 2014

Academic Editor: Panos Liatsis

Learning with imbalanced data is one of the emergent challenging tasks in machine learning. Recently, ensemble learning has arisen as an effective solution to class imbalance problems. The combination of bagging and boosting with data preprocessing resampling, namely, the simplest and accurate exploratory undersampling, has become the most popular method for imbalanced data classification. In this paper, we propose a novel selective ensemble construction method based on exploratory undersampling, `RotEasy`, with the advantage of improving storage requirement and computational efficiency by ensemble pruning technology. Our methodology aims to enhance the diversity between individual classifiers through feature extraction and diversity regularized ensemble pruning. We made a comprehensive comparison between our method and some state-of-the-art imbalanced learning methods. Experimental results on 20 real-world imbalanced data sets show that `RotEasy` possesses a significant increase in performance, contrasted by a nonparametric statistical test and various evaluation criteria.

1. Introduction

Recently, classification with imbalanced data sets has emerged as one of the most challenging tasks in data mining community. Class imbalance occurs when examples of one class are severely outnumbered by those of other classes. When data are imbalanced, traditional data mining algorithms tend to favor the overrepresented (majority or negative) class, resulting in unacceptably low recognition rates with respect to the underrepresented (minority or positive) class. However, the underrepresented minority class usually represents the positive concept with great interest than the majority class. The classification accuracy of the minority class is more preferred than the majority class. For instance, the recognition goal of medical diagnosis is to provide a higher identification accuracy for rare diseases. Similar to most of the existing imbalanced learning methods in the literature, we also focus on two-class imbalanced classification problems in our current study.

Class imbalance problems have appeared in many real-world applications, such as fraud detection [1], anomaly detection [2], medical diagnosis [3], DNA sequences analysis [4], etc. On account of the prevalence of potential applications, a large amount of techniques have been developed to deal with class imbalance problems. Interested readers can refer to some review papers [5–7]. These proposals can be divided into three categories, depending on the way they work.

(i) External approaches at data level: this type of methods consists of resampling data in order to decrease the effect of imbalanced class distribution. These approaches can be broadly categorized into two groups: undersampling the majority class and oversampling the minority class [8, 9]. They have the advantage of being independent from the classifier used, so they are considered as resampling preprocessing techniques.

(ii) Internal approaches at algorithmic level: these approaches try to adapt the decision threshold to impose a bias on the minority class or by adjusting misclassification costs for each class in the learning process [10–12]. These approaches are more dependent on the problem and the classifier used.

(iii) Combined approaches that are based on data preprocessing and ensemble learning, most commonly used boosting and bagging: they usually include data preprocessing techniques before ensemble learning.

The third group has arisen as popular methods for solving imbalanced data classification, mainly due to their ability to significantly improve the performance of a single classifier. In general, there are three kinds of ensemble patterns that are integrated with data preprocessing techniques: boosting-based ensembles, bagging-based ensembles, and hybrid ensembles. In the first boosting-based category, these methods alter and bias the weight distribution towards the minority class to train the next classifier, including SMOTEBoost [13], RUSBoost [14], and RAMOBoost [15]. In the second bagging-based category, the main difference lies in the way how to take into account each class of instances when they are randomly drawn in each bootstrap sampling. There are several different proposals, such as UnderBagging [16] and SMOTEBagging [17].

The main characteristic of the third category is that they carry out hierarchical ensemble learning, combining both bagging and boosting with resampling preprocessing technique. The simplest method in this group is exploratory undersampling, which was proposed by Liu et al. [18], also known as `EasyEnsemble`. It uses bagging as the main ensemble learning framework, and each bag member is actually an AdaBoost ensemble classifier. Hence, it combines the merits of boosting and bagging and strengthens the diversity of ensemble classifiers. The empirical study confirms that `EasyEnsemble` is highly effective in dealing with imbalanced data classification tasks.

It is widely recognized that diversity among individual classifiers is pivotal to the success of ensemble learning system. Rodriguez et al. [19] proposed a novel forward extension of bagging, rotation forest, which promotes diversity within the ensemble through feature extraction based on PCA. Moreover, many ensemble pruning techniques have been developed to select more diverse subensemble classifiers. For example, Li et al. [20] proposed a novel diversity regularized ensemble pruning method, namely DREP method, and greatly improved the generalization capability of ensemble classifiers.

Motivated by the above analysis, we will propose a novel ensemble construction technique, `RotEasy`, in order to enhance the diversity between component classifiers. The main idea of `RotEasy` is to inherit the advantages of `EasyEnsemble` and rotation forest by integrating them. We conducted a comprehensive suite of experiments on 20 real-world imbalanced data sets. They provide a complete perspective on the performance of the proposed algorithm. Experimental results indicate that our approach outperforms the compared state-of-the-art imbalanced learning methods significantly.

The remainder of this paper is organized as follows. Section 2 presents some related learning algorithms with the aim to facilitate discussions. In Section 3, we describe in detail the proposed methodology and its rationale. Section 4 introduces the experimental framework, including experimental data sets, the compared methods, and the used performance evaluation criteria. In Section 5, we show and discuss the experimental results. Finally, conclusions and some future work are outlined in Section 6.

2. Related Work and Motivation

In order to facilitate our later discussions, we will give a brief introduction to exploratory undersampling, rotation forest, and DREP ensemble pruning method.

2.1. Exploratory Undersampling. Undersampling is an efficient method for handling class imbalance problems, which uses only a subset of the majority class. Since many majority examples are ignored, the training set becomes more balanced and the training process becomes faster. However, some potentially useful information contained in these ignored majority examples is neglected. Liu et al. [18] proposed exploratory undersampling to further exploit these ignored examples while keeping the fast training speed, also known as `EasyEnsemble`.

Given a minority set $\mathcal{P}$ and a majority set $\mathcal{N}$, `EasyEnsemble` independently samples several subsets $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_T$ from $\mathcal{N}$, where $|\mathcal{N}_i| < |\mathcal{N}|$. For each majority subset $\mathcal{N}_i$ combined with the minority set $\mathcal{P}$, AdaBoost [22] is used to train the base classifier H_i. All generated base classifiers are fused by weighted voting for the final decision. The pseudocode for `EasyEnsemble` is shown in Algorithm 1.

`EasyEnsemble` generates T balanced subproblems, in which the ith subproblem is to learn an Adaboost ensemble H_i. So it looks like an "ensemble of ensembles." It is well known that boosting mainly reduces bias, while bagging mainly reduces variance. It is evident that `EasyEnsemble` has benefited from good qualities of boosting and a bagging-like strategy with balanced class distribution.

Experimental results in [18] show that `EasyEnsemble` has higher *AUC*, *F*-measure, and *G*-mean values than many existing imbalanced learning methods. Moreover, `EasyEnsemble` has approximately the same training time as that of undersampling, which is significantly faster than other algorithms.

2.2. Rotation Forest. Bagging consists in training different classifiers with multiple bootstrapped replicas of the original training data. The only factor encouraging diversity between individual classifiers is the proportion of different samples in the training data, and so bagging appears to generate ensembles of low diversity. Hence, Rodriguez et al. [19] proposed a novel forward extension of bagging, rotation forest, which promotes diversity within the ensemble through feature extraction based on Principal Component Analysis (PCA).

In each iteration of rotation forest algorithm, it consists in randomly splitting the feature set into K subsets, running feature extraction based on PCA separately on each subset, and then reassembling a new extracted feature set while keeping all the components. A decision tree classifier is

(i) **Input**: A minority training set $\mathscr{P}$ and a majority training set $\mathscr{N}$, $|\mathscr{P}| \ll |\mathscr{N}|$. T: the number of subsets undersampling from $\mathscr{N}$, s_i: the number of iterations in Adaboost learning.
(ii) **Training Phase:**
(iii) **For** $i = 1$ **to** T **do**
(1) Randomly sample a subset $\mathscr{N}_i$ from $\mathscr{N}$, $|\mathscr{N}_i| = |\mathscr{P}|$.
(2) Learn an ensemble classifier H_i using $\mathscr{P}$ and $\mathscr{N}_i$. H_i is an Adaboost ensemble with s_i number of weak classifiers $h_{i,j}$, corresponding weights $\alpha_{i,j}$ and threshold θ_i:

$$H_i(x) = \operatorname{sign}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right).$$

(iv) **Endfor**
(v) **Output**: The final ensemble:

$$H(x) = \operatorname{sign}\left(\sum_{i=1}^{T}\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right).$$

Here, $\operatorname{sign}(x) = 1$ means that x is predicted as the positive class. Conversely, it means that x belongs to the negative class.

Algorithm 1: `EasyEnsemble` algorithm.

(i) **Input**: $X = \{x_i\}_{i=1}^{N}$: the objects in the training data set (an $N \times n$ matrix).
$Y = \{y_i\}_{i=1}^{N}$: the class labels of the training set (an $N \times 1$ matrix).
T: number of classifiers in the ensemble.
K: number of feature subsets.
$\Phi = \{-1, +1\}$: the set of class labels.
(ii) **Training Phase:**
(iii) **For** $i = 1$ **to** T **do**
(1) Calculate the rotation matrix R_i^a:
(a) Randomly split the feature set F into K subsets$F_{i,j}$, $j = 1, 2, \ldots, K$.
(b) For $j = 1$ to K do
Let $X_{i,j}$ be the data set X for the features in $F_{i,j}$.
Select a bootstrap sample $X_{i,j}^{\dagger}$ of 75% number of objects in $X_{i,j}$.
Apply PCA on $X_{i,j}^{\dagger}$ and store the component coefficients in a matrix $C_{i,j}$.
(c) Endfor
(d) Arrange the $C_{i,j} (j = 1, 2, \ldots, K)$ into a block diagonal matrix R_i.
(e) Construct R_i^a by rearranging columns of R_i to match the order of features in F.
(2) Build the classifier h_i using (XR_i^a, Y) as the training set.
(iv) **Endfor**
(v) **Output**: For a given x, calculate its class label assigned by the ensemble classifier h^*:

$$h^*(x) = \arg\max_{y \in \Phi} \sum_{i=1}^{T} I\left(h_i\left(xR_i^a\right) = y\right),$$

where $I(\cdot)$ is an indicator function.

Algorithm 2: Rotation forest algorithm.

trained with the transformed data set. Different splits of the feature set will lead to different rotations. Thus diverse classifiers are obtained. On the other hand, the information about the scatter of the data is completely preserved in the new space of extracted features. In this way, accurate and more diverse classifiers are built.

In the study of Rodriguez et al. [19], through the analysis tool of kappa-error diagram, they showed that rotation forest has similar diversity-accuracy pattern as bagging, but is slightly more diverse than bagging. Hence, rotation forest promotes diversity within the ensemble through feature extraction. The pseudocode of rotation forest is listed in Algorithm 2.

2.3. DREP Ensemble Pruning. With the goal of improving storage requirement and computational efficiency, ensemble pruning deals with the problem of reducing ensemble sizes. Furthermore, theoretical and empirical studies have shown that ensemble pruning can also improve the generalization performance of the complete ensemble.

Guided by theoretical analysis on the effect of diversity on the generalization performance, Li et al. [20] proposed

(i) **Input**: $H = \{h_i(x)\}_{i=1}^{n}$: ensemble to be pruned,:
$S = \{(x_i, y_i)\}_{i=1}^{m}$ validation data set, $\rho \in (0, 1)$: the tradeoff parameter.
(ii) **Output**: pruned ensemble H^*.
(1) initialize $H^* \leftarrow \varnothing$.
(2) $h(x) \leftarrow$ the classifier in H with the lowest error on S.
(3) $H^* \leftarrow \{h(x)\}$ and $H \leftarrow H \setminus \{h(x)\}$.
(4) **repeat**
(5) **for** each $h'(x) \in H$ do
(6) compute $d_{h'} \leftarrow \texttt{diff}(h', H^*)$
(7) **endfor**
(8) sort classifiers $h'(x) \in H$ in the ascending order of $d_{h'}$.
(9) $\Gamma \leftarrow$ the first $\lceil \rho \cdot |H| \rceil$ classifiers in the sorted list.
(10) $h(x) \leftarrow$ the classifier in Γ which most reduces the error of H^* on S.
(11) $H^* \leftarrow \{h(x)\}$ and $H \leftarrow H \setminus \{h(x)\}$
(12) **until** the error of H^* on S can not be reduced.

Algorithm 3: DREP ensemble pruning method.

Table 1: Description of the experimental data sets. Imbalance ratio is the value of $N_{\text{maj}}/N_{\text{min}}$.

Data set	Samples	Attributes	Minority class	$N_{\text{min}}/N_{\text{maj}}$	Imbalance ratio
Spambase	4601	57	Class 1	1813/2788	1.54
Vote	435	16	Class 1	168/267	1.59
Wdbc	569	30	Malignant	212/357	1.68
Ionosphere	351	33	Bad	126/225	1.79
Pima	768	8	Class 1	268/500	1.87
German	1000	24	Class 2	300/700	2.33
Phoneme	5404	5	Class 1	1586/3818	2.41
Haberman	306	3	Class 2	81/225	2.78
Vehicle	846	18	Opel	212/634	2.99
Cmc	1473	9	Class 2	333/1140	3.42
House	506	13	[20, 21]	106/400	3.77
Scrapie	3113	14	Class 1	531/2582	4.86
Yeast	1484	5	Class 4	163/1321	8.10
Mfeat_zer	2000	47	Digit 9	200/1800	9.00
Mfeat_kar	2000	64	Digit 9	200/1800	9.00
Satimage	6435	36	Class 4	626/5809	9.28
Abalone7	4177	8	Class 7	391/3786	9.68
Sick	3163	25	Class 1	293/2870	9.80
Cbands	12000	30	Class 1	500/11500	23.00
Ozone	2536	72	Class 1	73/2463	33.74

Table 2: Confusion matrix.

	Positive prediction	Negative prediction
Actually positive	True positive (TP)	False negative (FN)
Actually negative	False positive (FP)	True negative (TN)

Diversity Regularized Ensemble Pruning (DREP) method, which is a greedy forward ensemble pruning method with explicit diversity regularization. The pseudocode of DREP method is presented in Algorithm 3.

In Algorithm 3, the diversity is measured based on pairwise difference and is defined as follows:

$$\begin{aligned} \text{diff}\left(h_i, h_j\right) &= \frac{1}{m}\sum_{k=1}^{m} h_i\left(x_k\right) h_j\left(x_k\right), \\ \text{diff}\left(h', H\right) &= \frac{1}{m * n}\sum_{k=1}^{m}\sum_{i=1}^{n} h'\left(x_k\right) h_i\left(x_k\right). \end{aligned} \tag{1}$$

Starting with the classifier with the lowest error on validation set S, DREP method iteratively selects the best classifier based on both empirical error and diversity. Concretely, at each step it first sorts the candidate classifiers in the ascending order of their differences with current subensemble, and then from

Table 3: Performance results of all methods based on AUC evaluation metric. The values with boldface mean the best result.

Dataset	CART	RUSB	SMOB	UNBag	SMBag	AdaC	RAMO	RotF	Easy	RotE-un	RotEasy
Spambase	0.9308	0.9762	0.9847	0.9800	0.9798	0.9846	0.9808	0.9858	0.9881	0.9889	**0.9891**
Vote	0.8732	**0.9932**	0.9855	0.9879	0.8875	0.8955	0.9924	0.9871	0.9926	0.9873	0.9868
Wdbc	0.9315	0.9899	0.9913	0.9872	0.9876	0.9874	0.9946	0.9892	0.9948	**0.9952**	0.9928
Ionosphere	0.8827	0.8599	0.9786	0.8798	0.8742	0.9749	**0.9872**	0.8933	0.9728	0.9731	0.8876
Pima	0.7226	0.8187	0.8102	0.8305	0.8127	0.1876	0.7971	0.7711	0.8258	0.8191	**0.8317**
German	0.6918	0.7861	0.7814	0.7823	0.7974	0.2139	0.7711	0.8007	0.7924	0.8032	**0.8891**
Phoneme	0.8852	0.8180	0.9682	0.9525	0.9582	0.9608	0.9691	0.9629	0.9623	0.9617	**0.9736**
Haberman	0.6329	0.6135	0.6595	0.6827	0.6458	0.3285	0.6219	0.6643	0.6645	0.6969	**0.8435**
Vehicle	0.7289	0.8701	0.8636	0.8669	0.8477	0.1923	0.8465	0.8738	0.8684	0.8154	**0.9272**
Cmc	0.6655	0.7118	0.6670	0.7322	0.7060	0.2756	0.6626	0.7165	0.7117	0.8341	**0.8460**
House	0.6812	0.8136	0.8258	0.8259	0.8121	0.5293	0.8274	0.8181	0.8486	0.8438	**0.8917**
Scrapie	0.6334	0.6393	0.6065	0.6543	0.6256	0.3479	0.6099	0.6395	0.6465	0.6444	**0.7208**
Yeast	0.8525	0.9733	0.9648	0.9749	0.9539	0.9739	0.9695	0.9721	0.9756	0.9759	**0.9767**
Mfeat_zer	0.8538	0.9923	**0.9965**	0.9793	0.9715	0.9843	0.9939	0.9944	0.9933	0.9906	0.9889
Mfeat_kar	0.8611	0.9942	**0.9962**	0.9761	0.9796	0.9854	0.9923	0.9906	0.9932	0.9942	0.9932
Satimage	0.7955	0.9462	0.9703	0.9522	0.9535	0.9607	0.9704	0.9593	0.9596	0.9576	**0.9716**
Abalone7	0.6887	0.8423	0.8144	0.8569	0.8245	0.8368	0.8191	0.8506	0.8485	0.8554	**0.8647**
Sick	0.9521	0.9879	0.9885	0.9882	0.9843	0.9858	0.9876	0.9818	0.9883	0.9884	**0.9896**
Cbands	0.8795	0.9952	**0.9974**	0.9913	0.9935	0.9938	0.9958	0.9954	0.9936	0.9955	0.9958
Ozone	0.6825	0.8807	0.8895	0.8911	0.8566	0.8689	0.8772	0.8778	0.8957	0.9018	**0.9220**
Average	0.7912	0.8751	0.8870	0.8885	0.8726	0.7234	0.8833	0.8862	0.8958	0.8955	**0.9241**

the front part of the sorted list it selects the classifier which can most reduce the empirical error on the validate data set. These two criteria are balanced by the parameter ρ, that is, the fraction of classifiers that are considered when minimizing empirical error. Obviously, a large value ρ means that more emphasis is put on the empirical error, while a small ρ pays more attention on the diversity. Thus it can be expected that the obtained ensemble will have both large diversity and small empirical error.

Experimental results show that, with the help of diversity regularization, DREP is able to achieve significantly better generalization performance with smaller ensemble size than other compared ensemble pruning methods.

3. RotEasy: A New Selective Ensemble Algorithm Based on EasyEnsemble and Rotation Forest

Based on the above analysis, we propose a novel selective ensemble construction technique `RotEasy`, integrating feature extraction, and ensemble pruning with `EasyEnsemble` to further improve the ensemble diversity.

The main steps of `RotEasy` can be summarized as follows: firstly, a subset $\mathcal{N}_i$ of size $|\mathcal{P}|$ from the majority class is undersampled. Secondly, we construct an inner-layer ensemble H_i through integrating rotation forest and AdaBoost. Lastly, DREP method is used to prune the learned ensemble with the aim to enhance the ensemble diversity. The pseudocode of `RotEasy` method is listed in Algorithm 4.

It should be pointed out that some parameters in `RotEasy` need to be specified in advance. With respect to the values of T and s_i, we set them in the same manner as that of `EasyEnsemble`. As for the validation set S, we randomly split the training set into two parts with approximately the same size, one part is used to train ensemble members, and the other one is used to prune ensemble classifiers. The best value for the parameter ρ can be found by a line-search strategy over $\{0.2, 0.25, \ldots, 1\}$. In fact, the performance of `RotEasy` is very robust to the variation of ρ values, and this will be confirmed in the later experimental analysis.

4. Experimental Framework

In this section, we present the experimental framework to examine the performance of our proposed `RotEasy` method and compare it with some state-of-the-art imbalanced learning methods.

4.1. Experimental Data Sets. To evaluate the effectiveness of the proposed method, extensive experiments were carried out on 20 public imbalanced data sets from the UCI repository. In order to ensure a thorough performance assessment, the chosen data sets vary in sample size, class distribution, and imbalance ratio.

Table 1 summarizes the properties of data sets: the number of examples, the number of attributes, sample size of minority and majority class, and the imbalance ratio, that is, sample size of the majority class divided by that of the minority class. These data sets are sorted by imbalance ratio in the ascending order. For several multiclass data sets, they were modified into two-class cases by keeping one class as

Table 4: Performance results of all methods based on G-mean evaluation metric. The values with boldface mean the best result.

Dataset	CART	RUSB	SMOB	UNBag	SMBag	AdaC	RAMO	RotF	Easy	RotE-un	RotEasy
Spambase	0.9128	0.8914	0.9391	0.9374	0.9330	0.9478	0.9474	0.9503	0.9565	**0.9586**	0.9457
Vote	0.9443	0.9536	0.9497	0.9524	0.9471	0.9472	0.9422	0.9457	0.9573	0.9528	**0.9666**
Wdbc	0.9207	0.9471	0.9648	0.9402	0.9444	0.9486	0.9663	0.9567	0.9621	0.9577	**0.9688**
Ionosphere	0.8621	0.9039	0.9382	0.8989	0.8942	0.9126	0.9369	0.9327	0.9215	0.9238	**0.9538**
Pima	0.6599	0.7446	0.7366	0.7375	0.6898	0.2401	0.7216	0.7206	0.7423	0.7253	**0.8307**
German	0.6374	0.7258	0.6683	0.7085	0.6325	0.2408	0.6527	0.6436	0.7238	0.7203	**0.8007**
Phoneme	0.8363	0.7836	0.9079	0.8826	0.8854	0.8864	0.9054	0.8875	0.8966	0.8917	**0.9222**
Haberman	0.5299	0.5932	0.5975	0.6311	0.4673	0.2395	0.4876	0.4746	0.6167	0.6332	**0.7452**
Vehicle	0.6585	0.7981	0.7602	0.7828	0.6626	0.2243	0.6845	0.6542	0.7944	0.7352	**0.8379**
Cmc	0.5314	0.6308	0.5793	0.6625	0.5406	0.2389	0.5544	0.5121	0.6519	0.6569	**0.7610**
House	0.5539	0.7196	0.7394	0.7393	0.6172	0.4818	0.6725	0.5386	0.7789	0.7369	**0.7874**
Scrapie	0.3980	0.5663	0.3968	0.6017	0.5651	0.1860	0.4667	0.3951	0.5969	0.5926	**0.6471**
Yeast	0.8144	0.9089	0.8956	0.9300	0.8706	0.8740	0.8705	0.8218	0.9289	0.9285	**0.9314**
Mfeat_zer	0.8167	0.9268	**0.9645**	0.9373	0.8789	0.8839	0.9342	0.8894	0.9621	0.9638	0.9568
Mfeat_kar	0.8282	0.9182	**0.9638**	0.9075	0.8754	0.8730	0.9177	0.8638	0.9554	0.9579	0.9586
Satimage	0.7218	0.8375	0.8633	0.8797	0.7849	0.7645	0.8067	0.7423	0.8915	0.8895	**0.9032**
Abalone7	0.4591	0.5625	0.5733	0.7845	0.4991	0.3692	0.4504	0.2616	0.7911	0.7921	**0.7939**
Sick	0.9064	0.9417	0.9458	0.9599	0.9322	0.9342	0.9330	0.9285	0.9588	0.9566	**0.9642**
Cbands	0.8531	0.9371	0.9631	0.9557	0.9138	0.9150	0.9390	0.9049	0.9663	0.9744	**0.9763**
Ozone	0.3627	0.2427	0.6495	0.8101	0.4911	0.2280	0.2649	0.1143	0.8121	0.8161	**0.8407**
Average	0.7103	0.7765	0.7998	0.8320	0.7513	0.6168	0.7527	0.7069	0.8432	0.8382	**0.8746**

the positive class and joining the remainder into the negative class.

4.2. Benchmark Methods. Regarding ensemble-based imbalanced learning algorithms, we compare our `RotEasy` approach with some competitive relevant algorithms, including RUSBoost [14], SMOTEBoost [13], UnderBagging [16], SMOTEBagging [17], AdaCost [10], RAMOBoost [15], rotation forest [19], and `EasyEnsemble` [18].

In our experiments, we use classification and regression tree (CART) as the base classifier in all compared methods, because it is sensitive to the changes of training samples, and can still be very accurate. We set the total amount of base classifiers in the ensemble to be $T = 100$. These benchmark methods and their parameters are described as follows.

(1) *CART*. It is implemented by the "classregtree" function with default parameter values in MATLAB software.

(2) *RUSBoost* (ab. *RUSB*). A majority subset $\mathcal{N}'$ is sampled (without replacement) from $\mathcal{N}$, $|\mathcal{N}'| = |\mathcal{P}|$. Then, AdaBoost is used to train an ensemble classifier using $\mathcal{P}$ and $\mathcal{N}'$.

(3) *SMOTEBoost* (ab. *SMOB*). It firstly uses SMOTE to get new minority class examples. Both classes contribute to the training data with N_{maj} instances. Then AdaBoost is used to train the ensemble classifiers using the new minority class samples and majority samples. In SMOTE algorithm, the number of nearest neighbors is set to be $k = 5$.

(4) *UnderBagging* (ab. *UNBag*). It removes instances from the majority class by random undersampling (without replacement) in each bagging member. Both classes contribute to each iteration with $N_{\min}$ instances.

(5) *SMOTEBagging* (ab. *SMBag*). Both classes contribute to each bag with N_{maj} instances. In each bag, a SMOTE resampling rate (a%) is set (ranging from 10% in the first iteration to 100% in the last). This ratio defines the number of positive instances ($a\% \cdot N_{\text{maj}}$) randomly resampled (with replacement) from the original positive class. The rest of positive instances are generated by the SMOTE algorithm. The number of nearest neighbors used in SMOTE is set to be $k = 5$.

(6) *AdaCost* (ab. *AdaC*). The cost factor of positive and negative instances is set to be $C_P = 1, C_N = 0.7$, respectively, according to the study in Yin et al. [23].

(7) *RAMOBoost* (ab. *RAMO*). According to the suggestion of [15], the number of nearest neighbors in adjusting the sampling probability of the minority is set to be $k_1 = 5$, the number of nearest neighbors used to generate the synthetic data instances is set to be $k_2 = 5$, and the scaling coefficient α is set to be 0.3.

(8) *Rotation forest* (ab. *RotF*). The feature set is randomly split into K subsets and PCA is applied to each bootstrapped subset. The number of features in each subset is set to be $n = 5$.

(9) *EasyEnsemble* (ab. *Easy*). It is firstly randomly undersampling (without replacement) the majority class in each outer-layer iteration. Then, AdaBoost is used to

Table 5: Performance results of all methods based on F-measure evaluation metric. The values with boldface mean the best result.

Dataset	CART	RUSB	SMOB	UNBag	SMBag	AdaC	RAMO	RotF	Easy	RotE-un	RotEasy
Spambase	0.8947	0.8630	0.9201	0.9226	0.9233	0.9371	0.9351	0.9412	0.9454	0.9481	**0.9565**
Vote	0.9305	0.9356	0.9420	0.9391	0.9405	0.9320	0.9263	0.9319	0.9394	0.9367	**0.9561**
Wdbc	0.9001	0.9218	0.9481	0.9228	0.9375	0.9361	**0.9567**	0.9507	0.9538	0.9441	0.9531
Ionosphere	0.8261	0.8708	0.9250	0.8661	0.8688	0.8941	0.9138	0.9194	0.8942	0.9034	**0.9373**
Pima	0.5683	0.6750	0.6562	0.6607	0.6020	0.2789	0.6427	0.6399	0.6662	0.6421	**0.7649**
German	0.5134	0.6149	0.5621	0.5977	0.5319	0.2953	0.5344	0.5446	0.6114	0.6039	**0.7027**
Phoneme	0.7744	0.6770	0.8529	0.8075	0.8373	0.8423	0.8586	0.8468	0.8274	0.8196	**0.8661**
Haberman	0.3741	0.4549	0.4218	0.4736	0.3235	0.3412	0.3584	0.3402	0.4652	0.4621	**0.5949**
Vehicle	0.518	0.6567	0.6278	0.6359	0.5307	0.2638	0.5522	0.5432	0.6506	0.6571	**0.7087**
Cmc	0.3649	0.4472	0.3876	0.4693	0.3813	0.2981	0.3828	0.3615	0.4577	0.4630	**0.5797**
House	0.3899	0.5431	0.5532	0.5392	0.4525	0.3066	0.5044	0.3929	**0.5792**	0.5375	0.5745
Scrapie	0.2420	0.3302	0.2087	0.3423	0.3262	0.2578	0.2818	0.2380	0.3362	0.3317	**0.3856**
Yeast	0.6982	**0.7878**	0.7736	0.7302	0.7614	0.7663	0.7603	0.7322	0.7244	0.7220	0.7368
Mfeat_zer	0.7135	0.9059	**0.9201**	0.7388	0.8167	0.8502	0.9127	0.8735	0.8203	0.8478	0.8161
Mfeat_kar	0.7201	0.8955	**0.9191**	0.7042	0.8049	0.8364	0.9070	0.8465	0.8686	0.8910	0.8080
Satimage	0.5491	0.6081	**0.7235**	0.5704	0.6668	0.6749	0.7215	0.6753	0.5794	0.5825	0.5925
Abalone7	0.2407	0.3203	0.3289	0.3769	0.2759	0.1986	0.2543	0.1138	0.3824	0.3865	**0.3985**
Sick	0.8389	0.8703	**0.8857**	0.8054	0.8714	0.8750	0.8848	0.8670	0.8155	0.8159	0.8137
Cbands	0.7514	0.9275	**0.9392**	0.7419	0.8775	0.9024	0.9335	0.8979	0.8199	0.8533	0.7909
Ozone	0.1737	0.1510	**0.2872**	0.1998	0.2572	0.1472	0.1860	0.0740	0.2017	0.2043	0.2068
Average	0.5991	0.6728	0.6890	0.6522	0.6494	0.5917	0.6704	0.6365	0.6769	0.6776	**0.7072**

train inner-layer ensemble classifier. The number of sample subsets is set to be $T = 10$, and the number of AdaBoost iterations is set to be $s_i = 10$.

(10) *Unpruned RotEasy* (ab. *RotE-un*). The number of undersampled subsets is $T = 10$; $s_i = 10$ inner-layer ensemble is constructed through integrating rotation forest and AdaBoost.

(11) *Our proposed method* (ab. *RotEasy*). The number of undersampled subsets is $T = 10$, and the number of inner ensemble iterations is $s_i = 10$. Then DREP method is applied on the validation subset S to prune the above ensemble.

RotE-un and RotEasy, we randomly split the training data set into two parts: 1/2 as training set, 1/2 as validation set. The parameter ρ is selected in $\{0.2, 0.25, \ldots, 1\}$ with an interval of 0.05.

4.3. Evaluation Measures. The evaluation criterion plays a crucial role in both the guidance of classifier modeling and the assessment of classification performance. Traditionally, total accuracy is the most commonly used empirical metric. However, accuracy is no longer a proper measure in the class imbalance problem, since the positive class makes little contribution to the overall accuracy.

For the two-class problem we consider here, the confusion matrix records the results of correctly and incorrectly classified examples of each class. It is shown in Table 2.

Specially, we obtain the following performance evaluation metrics from the confusion matrix:

True positive rate: the percentage of positive instances correctly classified, $\mathrm{TP}_{\mathrm{rate}} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$, also known as *Recall*;

True negative rate: the percentage of negative instances correctly classified, $\mathrm{TN}_{\mathrm{rate}} = \mathrm{TN}/(\mathrm{FP} + \mathrm{TN})$;

False positive rate: the percentage of negative instances misclassified, $\mathrm{FP}_{\mathrm{rate}} = \mathrm{FP}/(\mathrm{FP} + \mathrm{TN})$;

False negative Rate: the percentage of positive instances misclassified, $\mathrm{FN}_{\mathrm{rate}} = \mathrm{FN}/(\mathrm{FN} + \mathrm{TP})$;

F-measure: the harmonic mean of *Precision* and *Recall*, *Precision* $= \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$, F-measure$= (2 \times \mathrm{Precision} \times \mathrm{Recall})/(\mathrm{Precision} + \mathrm{Recall})$;

G-mean: the geometric mean of $\mathrm{TP}_{\mathrm{rate}}$ and $\mathrm{TN}_{\mathrm{rate}}$, G-mean $= \sqrt{\mathrm{TP}_{\mathrm{rate}} \cdot \mathrm{TN}_{\mathrm{rate}}}$;

AUC: the area under the receiver operating characteristic (ROC). *AUC* provides a single measure of the classification performance for evaluating which model is better on average.

5. Experimental Results and Analysis

This section shows the experimental results and their associated statistical analysis for the comparison with standard imbalanced learning algorithms. All the reported results are obtained by ten trials of stratified 10-fold cross-validation. That is, the total data is split into 10 folds, with each fold containing 10% of data patterns for prediction. For each fold, each algorithm is trained with the examples of the remaining folds, and the prediction accuracy rate tested on the current

Table 6: Running time of all methods ($\times 10^3$ seconds).

Dataset	CART	RUSB	SMOB	UNBag	SMBag	AdaC	RAMO	RotF	Easy	RotE-un	RotEasy	Pruned size
Spambase	10.4082	7.2693	10.1375	5.5486	15.4115	6.9242	59.0225	11.2853	4.9132	2.6672	2.6911	30.7
Vote	0.2003	0.2210	0.4009	0.1390	0.2992	0.2809	1.0919	0.4173	0.1868	0.3886	0.1394	31.8
Wdbc	0.4012	0.3595	0.5896	0.2458	0.5172	0.4473	1.5186	0.6133	0.2273	0.5998	0.1739	27.5
Ionosphere	0.4127	0.3364	0.4921	0.2382	0.4762	0.4260	1.0328	0.5349	0.2097	0.5211	0.1769	33.6
Pima	0.5231	0.5059	0.8742	0.3236	0.7141	0.4880	2.0187	0.7559	0.2974	0.6468	0.2173	29.3
German	1.4815	1.1213	1.6231	0.7268	1.5688	1.1799	4.0155	1.9123	0.6149	1.3117	0.3958	30.9
Phoneme	1.8902	2.4271	6.0667	0.9614	4.1095	3.3139	27.158	2.5821	1.2318	2.6678	0.8184	28.4
Haberman	0.2025	0.2012	0.3240	0.1234	0.2793	0.2202	0.5496	0.2607	0.1223	0.2046	0.1186	32.5
Vehicle	0.8187	0.6848	1.1640	0.3923	1.0866	0.7974	2.7819	1.1901	0.3578	0.7882	0.2524	31.5
Cmc	1.1856	1.0555	1.6478	0.5068	1.5559	1.0299	4.7806	1.6271	0.5041	0.3332	0.3554	30.8
House	0.3794	0.3531	0.6048	0.1914	0.5908	0.4065	1.2942	0.7000	0.1815	0.1395	0.1461	32.7
Scrapie	2.8443	2.3650	3.8515	0.9799	3.8089	2.9333	17.0674	3.4533	0.9927	0.7025	0.7461	26.8
Yeast	0.3719	0.5099	0.9379	0.1362	0.7436	0.6230	4.4375	0.6172	0.1494	0.1116	0.1266	28.6
Mfeat_zer	1.9185	1.3039	2.6371	0.4285	3.6426	2.0611	11.7596	2.7819	0.3542	0.2572	0.2761	31.8
Mfeat_kar	2.9168	1.8031	3.8492	0.5879	5.1841	3.2124	13.3192	4.6771	0.4755	0.3293	0.3546	32.5
Satimage	5.8025	5.1291	10.9213	1.3928	12.9263	8.5528	49.5446	6.6679	1.2619	0.7116	0.7444	28.6
Abalone7	1.6428	1.8622	3.4376	0.4320	3.1747	1.5699	24.0045	2.2932	0.4304	0.0872	0.1317	30.3
Sick	1.3551	1.4166	2.8468	0.2974	2.4132	1.1650	19.4192	2.2750	0.3248	0.2741	0.2846	27.8
Cbands	9.0694	6.0648	15.1124	0.9900	16.4782	7.3421	114.5434	11.5554	0.6354	0.3855	0.4689	31.5
Ozone	3.3591	2.1542	5.9775	0.3463	8.8094	2.0600	17.8972	3.6767	0.2672	0.1995	0.2280	30.6
Average	2.3592	1.8578	3.6754	0.7356	4.1905	2.2541	18.8632	2.9946	0.6875	0.6664	0.4425	30.41

(i) **Input**: $\mathcal{N}$: the majority set, $\mathcal{P}$: the minority set, T: the number of subsets undersampling from $\mathcal{N}$, s_i: the number of inner-layer ensemble, $S = \{(\widetilde{x}_i, \widetilde{y}_i)\}_{j=1}^{m}$: validation dataset, $\rho \in (0, 1)$: tradeoff parameter, $\Phi = \{-1, +1\}$: the set of class labels.

(ii) **For** $i = 1$ **to** T **do**

(a) Randomly undersampling a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.

(b) Learning the inner-layer ensemble $H_i = \{h_{i,j}\}_{j=1}^{s_i}$:

(1) Set $\mathcal{L}_i = [\mathcal{P}, \mathcal{N}_i] = \{(x_k^i, y_k^i)\}_{k=1}^{N} = (X_i, Y_i)$, the weak classifier $\mathcal{W}$, initial weight distribution on the training set as $D_i^1(x_k^i) = 1/N$.

(2) **for** $j = 1$ **to** s_i **do**

(3) Calculate the rotation matrix $R_{i,j}^a$ using X_i, based on Algorithm 2.

(4) Get the sampling subset $\mathcal{L}_{i,j} = (X_{i,j}, Y_{i,j})$ using weight distribution D_i^j.

(5) Learn $h_{i,j}$ by providing the transformed subset $(X_{i,j}R_{i,j}^a, Y_{i,j})$ as the input of classifier $\mathcal{W}$.

(6) Calculate the training error $\varepsilon_{i,j}$ over $\mathcal{L}_i$: $\varepsilon_{i,j} = \sum_{k:y_k^i \neq h_{i,j}(x_k^i R_{i,j}^a)} D_i^j(k)$.

(7) Set the weight $\alpha_{i,j}$: $\alpha_{i,j} = (1/2)\log\left((1-\varepsilon_{i,j})/\varepsilon_{i,j}\right)$.

(8) Update D_i^{j+1} over $\mathcal{L}_i$: $D_i^{j+1}(k) = \left[D_i^j(k) \cdot e^{-\alpha_{i,j}h_{i,j}(x_k^i R_{i,j}^a)y_k^i}\right]/Z_{i,j}$, where $Z_{i,j}$ is the normalization constant: $Z_{i,j} = \sum_{k=1:N}\left[D_i^j(k) \cdot e^{-\alpha_{i,j}h_{i,j}(x_k^i R_{i,j}^a)y_k^i}\right]$.

(9) **Endfor**

(iii) **Endfor**

(iv) **Pruning**: Apply the DREP method on the validation subset S to prune the ensemble $H = \{h_{i,j}, i = 1, 2 \ldots T, j = 1, 2 \ldots, \mathrm{si}\}$. Denote the pruned ensemble members as $\{\widehat{h}_k\}_{k=1}^{K}$, their corresponding normalized weights $\{\widehat{\alpha}_k\}_{k=1}^{K}$ and rotation matrices $\{\widehat{R}_k^a\}_{k=1}^{K}$.

(v) **Classification Phase**: For a given x, calculate its class label $\widehat{h}(x)$ as follows:

$$\widehat{h}(x) = \arg\max_{y\in\Phi} \sum_{k=1}^{K} \widehat{\alpha}_k I\left(\widehat{h}_k\left(x\widehat{R}_k^a\right) = y\right).$$

Algorithm 4: RotEasy algorithm.

Table 7: Pairwise comparisons of all algorithms based on the AUC criterion.

Algorithms		CART	RUSB	SMOB	UNBag	SMBag	AdaC	RAMO	RotF	Easy	RotE-un	RotEasy
Mean		0.7912	0.8751	0.8870	0.8885	0.8726	0.7234	0.8833	0.8862	0.8958	0.8955	**0.9241**
CART	$\dot{r}$		1.1053	1.1191	1.1248	1.1032	0.7894	1.1133	1.1204	1.1327	1.1329	1.1749
	s		17-0-3	19-0-1	19-0-1	18-0-2	13-0-7	17-0-3	20-0-0	20-0-0	20-0-0	20-0-0
	q		0.0015	0.0000	0.0000	0.0002	0.1892	0.0015	0.0000	0.0000	0.0000	0.0000
RUSB	$\dot{r}$			1.0124	1.0176	0.9981	0.7142	1.0072	1.0136	1.0248	1.0250	1.0629
	s			12-0-8	13-0-7	6-0-14	5-0-15	9-0-11	13-0-7	15-0-5	16-1-3	17-0-3
	q			0.3833	0.1892	0.0784	0.0266	0.6636	0.1892	0.0266	0.0026	0.0015
SMOB	$\dot{r}$				1.0051	0.9858	0.7054	0.9949	1.0012	1.0122	1.0124	1.0499
	s				11-0-9	5-0-15	2-0-18	9-0-11	9-0-11	13-0-7	12-0-8	16-0-4
	q				0.6636	0.0266	0.0002	0.6636	0.6636	0.1892	0.3833	0.0072
UNBag	$\dot{r}$					0.9808	0.7019	0.9898	0.9961	1.0071	1.0072	1.0445
	s					6-0-14	8-0-12	10-0-10	10-0-10	15-0-5	14-0-6	19-0-1
	q					0.0784	0.3833	1.0000	1.0000	0.0266	0.0784	0.0000
SMBag	$\dot{r}$						0.8633	1.0000	1.0108	1.0146	1.0166	1.0497
	s						12-0-8	13-0-7	18-0-2	19-0-1	19-0-1	20-0-0
	q						0.3833	0.1892	0.0002	0.0000	0.0000	0.0000
AdaC	$\dot{r}$							1.4103	1.4192	1.4349	1.4351	1.4883
	s							17-0-3	16-0-4	17-0-3	18-0-2	19-0-1
	q							0.0015	0.0072	0.0015	0.0002	0.0000
RAMO	$\dot{r}$								1.0063	1.0174	1.0176	1.0553
	s								10-0-10	15-0-5	13-0-7	15-0-5
	q								1.0000	0.0266	0.1892	0.0266
RotF	$\dot{r}$									1.0110	1.0112	1.0486
	s									13-0-7	15-0-5	16-1-3
	q									0.1892	0.0266	0.0026
Easy	$\dot{r}$										1.0002	1.0372
	s										12-0-8	15-1-4
	q										0.3833	0.0118
RotE-un	$\dot{r}$											1.0370
	s											14-0-6
	q											0.0784

fold is considered to be the performance result. For each data set, we compute the mean value of 100 prediction accuracy as the final prediction result.

Firstly, we investigated the sensitivity of proposed `RotEasy` algorithm with respect to the variation of hyperparameter ρ.

5.1. Sensitivity of the Hyperparameter ρ. In the DREP ensemble pruning method, there is a trade-off parameter ρ between ensemble diversity and empirical error. We should first examine the influence of the parameter ρ on the algorithm performance. To do so, we considered various values of ρ in $\{0.2, 0.25, \ldots, 1\}$ with increment 0.05 in this study.

Figure 1 shows the curves of performance results as a function of parameter ρ on several training data sets, based on *AUC*, *G-mean*, and *F-measure* evaluation metrics, respectively.

As seen in Figure 1, the performance of `RotEasy` varies by a small margin along with the change of parameter ρ. Thus, the proposed `RotEasy` algorithm is insensitive to the variation of parameter ρ. Hence, it is proper that we fix the value of parameter ρ to be 0.5 in the subsequent experiments.

5.2. Performance Comparison. In this section, we will compare our proposal `RotEasy` against the previously presented state-of-the-art methods. Before going through further analysis, we first show the *AUC*, *G-mean*,and *F-measure* values of all the methods on each data set in Tables 3, 4, and 5 respectively. We also draw the box plots of test results for all methods on the "Scrapie" data set in Figure 2. In this

Table 8: Pairwise comparisons of all algorithms based on the G-mean criterion.

Algorithms		CART	RUSB	SMOB	UNBag	SMBag	AdaC	RAMO	RotF	Easy	RotE-un	RotEasy
Mean		0.7103	0.7765	0.7998	0.8320	0.7513	0.6168	0.7527	0.7069	0.8432	0.8382	**0.8746**
CART	$\dot{r}$		1.0917	1.1389	1.2038	1.0672	0.7524	1.0469	0.9351	1.2187	1.2108	1.2709
	s		17-0-3	19-0-1	20-0-0	18-0-2	11-0-9	16-0-4	13-0-7	20-0-0	20-0-0	20-0-0
	q		0.0015	0.0000	0.0000	0.0002	0.6636	0.0072	0.1892	0.0000	0.0000	0.0000
RUSB	$\dot{r}$			1.0432	1.1027	0.9775	0.6891	0.9590	0.8565	1.1163	1.1091	1.1641
	s			13-0-7	13-0-7	3-0-17	4-0-16	7-0-13	4-0-16	17-0-3	16-0-4	20-0-0
	q			0.1892	0.1892	0.0015	0.0072	0.1892	0.0072	0.0015	0.0072	0.0000
SMOB	$\dot{r}$				1.0570	0.9370	0.6606	0.9192	0.8210	1.0700	1.0631	1.1159
	s				12-0-8	1-0-19	1-0-19	3-0-12	1-0-19	15-0-5	12-0-8	18-0-2
	q				0.3833	0.0000	0.0000	0.0015	0.0000	0.0266	0.3833	0.0002
UNBag	$\dot{r}$					0.8865	0.6250	0.8697	0.7768	1.0124	1.0058	1.0557
	s					2-0-11	4-0-16	5-0-15	4-0-16	15-0-5	13-0-7	20-0-0
	q					0.0002	0.0072	0.0266	0.0072	0.0266	0.1892	0.0000
SMBag	$\dot{r}$						0.7050	0.9810	0.8762	1.1420	1.1346	1.1909
	s						9-0-11	15-0-5	8-0-12	20-0-0	20-0-0	20-0-0
	q						0.6636	0.0266	0.3833	0.0000	0.0000	0.0000
AdaC	$\dot{r}$							1.3915	1.2429	1.6198	1.6093	1.6892
	s							16-0-4	12-0-8	20-0-0	20-0-0	19-0-1
	q							0.0072	0.3833	0.0000	0.0000	0.0000
RAMO	$\dot{r}$								0.8932	1.1640	1.1565	1.2139
	s								2-0-18	17-0-3	17-0-3	19-0-1
	q								0.0002	0.0015	0.0015	0.0000
RotF	$\dot{r}$									1.3033	1.2949	1.3591
	s									19-0-1	19-0-1	19-0-1
	q									0.0000	0.0000	0.0000
Easy	$\dot{r}$										0.9935	1.0428
	s										9-0-11	18-0-2
	q										0.6636	0.0002
RotE-un	$\dot{r}$											1.0496
	s											18-0-2
	q											0.0002

figure, the numbers shown on the horizontal axis indicate the corresponding algorithms introduced in Section 4.2. We can clearly see the relative performance of all the methods from these box plots.

It is obvious from Tables 3–5 that `RotEasy` always obtains the highest average values of *AUC*, *G*-mean, and *F*-measure. It outperforms all other methods by a large margin. Furthermore, `EasyEnsemble` and the new unpruned `RotEasy` (RotE-un) achieve better performance than other benchmark methods. However, `RotEasy` still outperforms them with a certain degree and becomes the best algorithm.

Moreover, we also investigate the computational efficiency of newly proposed `RotEasy` algorithm, through computing the running time of all algorithms and pruned ensemble size of `RotEasy` algorithm on all data sets. These results are listed in Table 6. From the last column of Table 6, we can see that the size of pruned ensemble drops from 100 to around 30. Then, it will greatly improve computational efficiency of `RotEasy` algorithm in prediction stage, particularly when we encounter the large-scale classification problems.

Hence, the average running time of `RotEasy` is the shortest in all methods, comparable to that of `EasyEnsemble` and UnderBagging. The RAMOBoost algorithm has the longest running time. Other compared algorithms can be ranked in the order from fast to slow as RUSB, AdaC, CART, RotF, SMOB, SMBag.

5.3. Statistical Tests. In order to show whether the newly proposed method offers a significant improvement over other methods for some given problems, we have to give

Table 9: Pairwise comparisons of all algorithms based on the F-measure criterion.

Algorithms		CART	RUSB	SMOB	UNBag	SMBag	AdaC	RAMO	RotF	Easy	RotE-un	RotEasy
Mean		0.5991	0.6728	0.6890	0.6522	0.6494	0.5917	0.6704	0.6365	0.6769	0.6776	**0.7072**
CART	$\dot{r}$		1.1397	1.1664	1.1304	1.1010	0.9199	1.1149	0.9823	1.1670	1.1664	1.2320
	s		17-0-3	19-0-1	17-0-3	19-0-1	12-0-8	18-0-2	15-0-5	19-0-1	19-0-1	19-0-1
	q		0.0015	0.0000	0.0015	0.0000	0.3833	0.0002	0.0266	0.0000	0.0000	0.0000
RUSB	$\dot{r}$			1.0234	0.9918	0.9661	0.8072	0.9783	0.8619	1.0239	1.0235	1.0810
	s			13-0-7	9-0-11	7-0-13	6-0-14	10-0-10	5-0-15	11-0-9	11-0-9	14-0-6
	q			0.1892	0.6636	0.1892	0.0784	1.0000	0.0266	0.6636	0.6636	0.0784
SMOB	$\dot{r}$				0.9691	0.9440	0.7887	0.9559	0.8422	1.0005	1.0001	1.0563
	s				8-0-12	3-0-17	2-0-18	4-0-16	3-0-17	10-0-10	7-0-13	13-0-7
	q				0.3833	0.0015	0.0002	0.0072	0.0015	1.0000	0.1892	0.1892
UNBag	$\dot{r}$					0.9740	0.8138	0.9863	0.8690	1.0324	1.0319	1.0899
	s					12-0-8	10-0-10	10-0-10	10-0-10	16-0-4	13-0-7	20-0-0
	q					0.3833	1.0000	1.0000	1.0000	0.0072	0.1892	0.0000
SMBag	$\dot{r}$						0.8355	1.0126	0.8922	1.0599	1.0594	1.1190
	s						10-0-10	15-0-5	12-0-8	13-0-7	13-0-7	14-0-6
	q						1.0000	0.0266	0.3833	0.1892	0.1892	0.0784
AdaC	$\dot{r}$							1.2120	1.0678	1.2685	1.2679	1.3393
	s							17-0-3	12-0-8	14-0-6	14-0-6	14-0-6
	q							0.0015	0.3833	0.0784	0.0784	0.0784
RAMO	$\dot{r}$								0.8810	1.0467	1.0462	1.1050
	s								4-0-16	11-0-9	10-0-10	13-0-7
	q								0.0072	0.6636	1.0000	0.1892
RotF	$\dot{r}$									1.1880	1.1875	1.2542
	s									13-0-7	12-0-8	15-0-5
	q									0.1892	0.3833	0.0266
Easy	$\dot{r}$										0.9996	1.0558
	s										11-0-9	14-0-6
	q										0.6636	0.0784
RotE-un	$\dot{r}$											1.0562
	s											16-0-4
	q											0.0072

the comparison a statistical support. A popular way to compare the overall performances is to count the number of problems on which an algorithm is the winner. Some authors use these counts in inferential statistics with a form of two-tailed binomial test, also known as the sign test.

Here, we employed the sign test utilized by Webb [24] to compare the relative performance of all considered algorithms. In the following description, *row* indicates the mean performance of the algorithm with which a row is labeled, while *col* indicates that of the algorithm with which a column is labeled. The first row represents the mean performance across all data sets. Rows labeled as $\dot{r}$ represent the geometric mean of the performance ratio col/row. Rows labeled as s represent the *win-tie-loss* statistic, where the three values refer to the numbers of data sets for which col > row, col = row, and col < row, respectively. Rows labeled as q represent the test P values of a two-tailed sign test based on the *win-tie-loss* record. If the value of q is smaller than the given significance level, the difference between the two considered algorithms is significant and otherwise it is not significant.

Tables 7, 8, and 9 show all the pairwise comparisons of considered algorithms based on *AUC*, *G*-mean, and *F*-measure metrics, respectively. The results show that `RotEasy` obtains the best performance among the compared algorithms. `RotEasy` not only achieves the highest mean performance, but also always gains the largest *win* records in the light of the last columns in Tables 7–9.

In terms of three used evaluation measures, the top three best algorithms are ranked in the same order of `RotEasy`, unpruned `RotEasy`, and `EasyEnsemble`. Other compared algorithms are approximately ranked in the order from better to worse as SMOB, RUSB, UNBag, RAMO, RotF, AdaC,

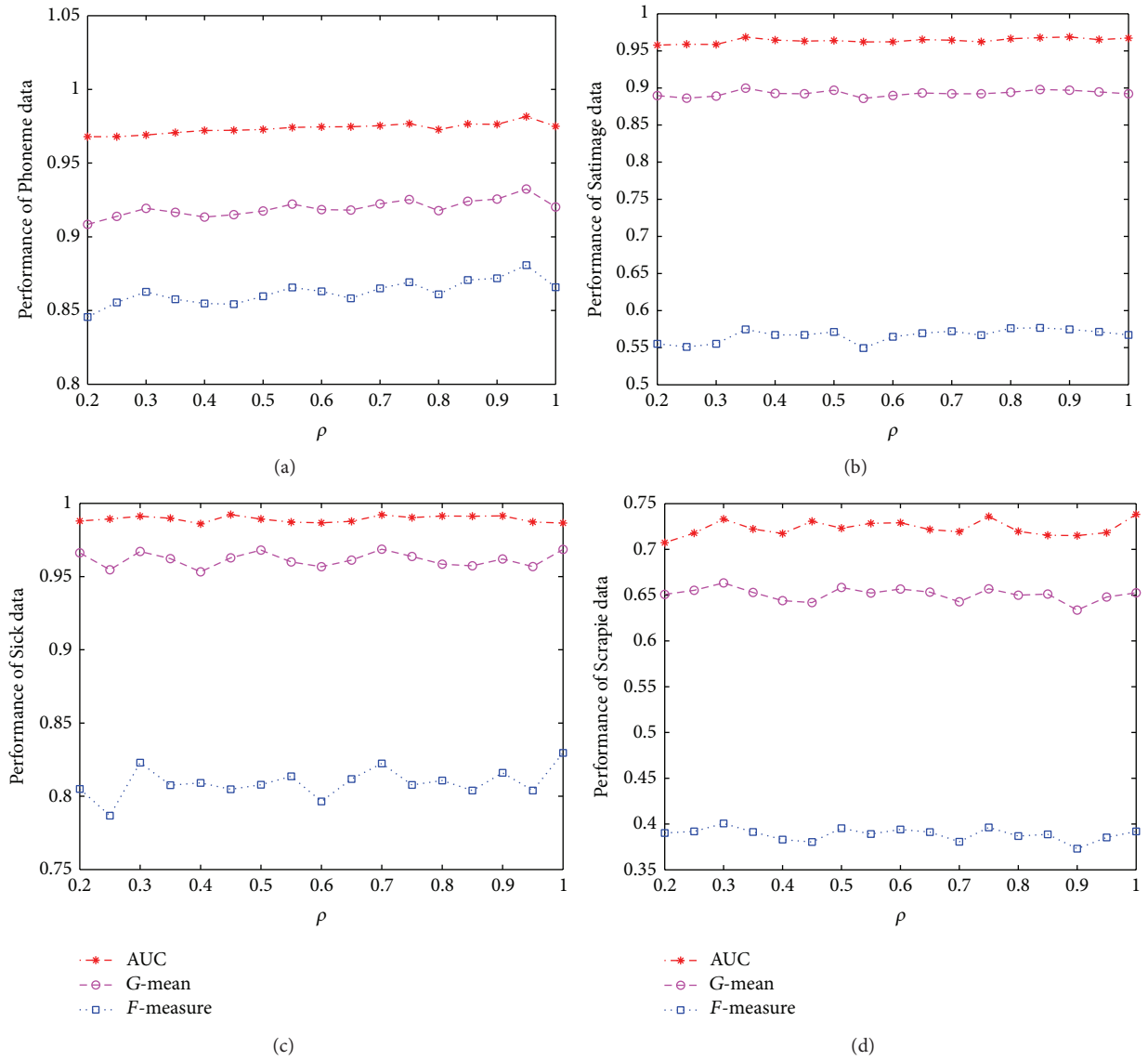

FIGURE 1: Performance of RotEasy algorithm versus the various values of parameter ρ on several data sets.

SMBag, and CART. This result is consistent with the findings of previous study [6, 7, 18].

6. Conclusions and Future Work

In this paper, we presented a new method RotEasy for constructing ensembles based on combining the principles of EasyEnsemble, rotation forest, and diversity regularized ensemble pruning methodology. EasyEnsemble uses bagging as the main ensemble learning framework, and each bagging member is composed of an AdaBoost ensemble classifier. It combines the merits of boosting and bagging ensemble strategy and becomes the most advanced approach handling class imbalance problems. The main innovation of RotEasy is to use the more diverse AdaBoost-based rotation forest as inner-layer ensemble instead of AdaBoost in the EasyEnsemble, and then further enhance the diversity through using DREP ensemble pruning method.

To verify the superiority of our proposed RotEasy approach, we established empirical comparisons with some state-of-the-art imbalanced learning algorithms, including RUSBoost, SMOTEBoost, UnderBagging, SMOTEBagging, AdaCost, RAMOBoost, rotation forest, and EasyEnsemble. The experimental results on 20 real-world imbalanced data sets show that RotEasy outperforms other compared imbalanced learning methods in term of *AUC*, *G*-mean, and *F*-measure, due to the ability of strengthening diversity. The improvement over other standard methods was also confirmed through the nonparametric sign test.

Based on the present work, there are also some interesting research work that deserved to be further investigated: (1) to integrate latest evolutionary undersampling with our proposed ensemble framework [21, 25], instead of common

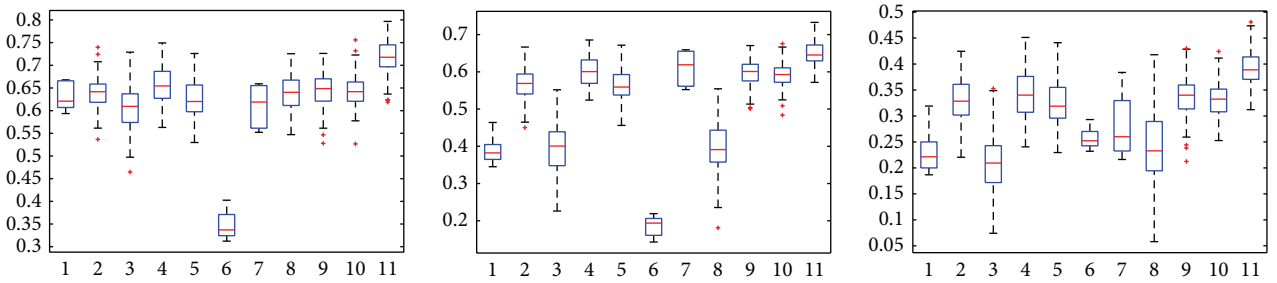

Figure 2: The box plots of *AUC*, *G*-mean, and *F*-measure results for all the algorithms on the "Scrapie" data set.

random undersampling; (2) to generalize this technique into multiclass imbalanced learning problems, while only binary class imbalanced classification were considered in current experiment [26–28]; (3) to extend our study into semisupervised learning of imbalanced classification problems [29, 30].

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the National Basic Research Program of China (973 Program) under Grant no. 2013CB329404, the Major Research Project of the National Natural Science Foundation of China under Grant no. 91230101, the National Natural Science Foundations of China under Grant no. 61075006 and no. 11201367, the Key Project of the National Natural Science Foundation of China under Grant no. 11131006.

References

[1] Z.-B. Zhu and Z.-H. Song, "Fault diagnosis based on imbalance modified kernel fisher discriminant analysis," *Chemical Engineering Research and Design*, vol. 88, no. 8, pp. 936–951, 2010.

[2] W. Khreich, E. Granger, A. Miri, and R. Sabourin, "Iterative Boolean combination of classifiers in the ROC space: an application to anomaly detection with HMMs," *Pattern Recognition*, vol. 43, no. 8, pp. 2732–2752, 2010.

[3] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi, "Training neural network classifiers for medical decision making: the effects of imbalanced datasets on classification performance," *Neural Networks*, vol. 21, no. 2-3, pp. 427–436, 2008.

[4] N. Garcia-Pedrajas, J. Perez-Rodriguez, M. Garcia-Pedrajas, D. Ortiz-Boyer, and C. Fyfe, "Class imbalance methods for translation initiation site recognition in DNA sequences," *Knowledge-Based Systems*, vol. 25, no. 1, pp. 22–34, 2012.

[5] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[6] T. M. Khoshgoftaar, J. van Hulse, and A. Napolitano, "Comparing boosting and bagging techniques with noisy and imbalanced data," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 41, no. 3, pp. 552–568, 2011.

[7] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, no. 4, pp. 463–484, 2012.

[8] N. V. Chawla, K. W. Bowyer, L. O. Hall et al., "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[9] A. Estabrooks, T. Jo, and N. Japkowicz, "A multiple resampling method for learning from imbalanced data sets," *Computational Intelligence*, vol. 20, no. 1, pp. 18–36, 2004.

[10] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[11] G. Wu and E. Chang, "KBA: kernel boundary alignment considering imbalanced data distribution," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 786–795, 2005.

[12] N. V. Chawla, D. Cieslak, L. O. Hall, and A. Joshi, "Automatically countering imbalance and its empirical relationship to cost," *Data Mining and Knowledge Discovery*, vol. 17, no. 2, pp. 225–252, 2008.

[13] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTEBoost: improving prediction of the minority class in boosting," in *Knowledge Discovery in Databases*, pp. 107–119, 2003.

[14] C. Seiffert, T. Khoshgoftaar, J. van Hulse, and A. Napolitano, "RUSBoost: a hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010.

[15] S. Chen, H. He, and E. A. Garcia, "RAMOBoost: ranked minority oversampling in boosting," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1624–1642, 2010.

[16] R. Barandela, R. M. Valdovinos, and J. S. Sanchez, "New applications of ensembles of classifiers," *Pattern Analysis and Applications*, vol. 6, no. 3, pp. 245–256, 2003.

[17] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM '09)*, pp. 324–331, 2009.

[18] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 2, pp. 539–550, 2009.

[19] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: a new classifier ensemble method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619–1630, 2006.

[20] N. Li, Y. Yu, and Z. H. Zhou, "Diversity regularized ensemble pruning," in *Proceedings of the 23rd European Conference on Machine Learning*, pp. 330–345, 2012.

[21] M. Galar, A. Fernandez, and E. Barrenechea, "EUSBoost: enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling," *Pattern Recognition*, vol. 46, no. 12, pp. 3460–3471, 2013.

[22] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771–780, 1999.

[23] Q. Y. Yin, J. S. Zhang, C. X. Zhang et al., "An empirical study on the performance of cost-sensitive boosting algorithms with different levels of class imbalance," *Mathematical Problems in Engineering*, vol. 2013, Article ID 761814, 12 pages, 2013.

[24] G. I. Webb, "MultiBoosting: a technique for combining boosting and wagging," *Machine Learning*, vol. 40, no. 2, pp. 159–196, 2000.

[25] S. Garcia and F. Herrera, "Evolutionary under-sampling for classification with imbalanced datasets: proposals and taxonomy," *Evolutionary Computation*, vol. 17, no. 3, pp. 275–306, 2009.

[26] L. Cerf, D. Gay, F. N. Selmaoui, B. Crémilleux, and J.-F. Boulicaut, "Parameter-free classification in multiclass imbalanced data sets," *Data and Knowledge Engineering*, vol. 87, pp. 109–129, 2013.

[27] S. Wang and X. Yao, "Multiclass imbalance problems: analysis and potential solutions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 4, pp. 1119–1130, 2012.

[28] M. Lin, K. Tang, and X. Yao, "Dynamic sampling approach to training neural networks for multiclass imbalance classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 4, pp. 647–660, 2013.

[29] K. Chen and S. Wang, "Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 129–143, 2011.

[30] M. Frasca, A. Bertoni, M. Re, and G. Valentini, "A neural network algorithm for semi-supervised node label learning from unbalanced data," *Neural Networks*, vol. 43, pp. 84–94, 2013.