Journal of Big Data

Boosting methods for multi-class imbalanced data classification: an experimental review

Jafar Tanha[*] , Yousef Abdi, Negin Samadi, Nazila Razzaghi and Mohammad Asadpour

*Correspondence:
tanha@tabrizu.ac.ir
Faculty of Electrical
and Computer Engineering,
University of Tabriz, P.O.
Box 51666-16471, Tabriz, Iran

Abstract

Since canonical machine learning algorithms assume that the dataset has equal number of samples in each class, binary classification became a very challenging task to discriminate the minority class samples efficiently in imbalanced datasets. For this reason, researchers have been paid attention and have proposed many methods to deal with this problem, which can be broadly categorized into data level and algorithm level. Besides, multi-class imbalanced learning is much harder than binary one and is still an open problem. Boosting algorithms are a class of ensemble learning methods in machine learning that improves the performance of separate base learners by combining them into a composite whole. This paper's aim is to review the most significant published boosting techniques on multi-class imbalanced datasets. A thorough empirical comparison is conducted to analyze the performance of binary and multi-class boosting algorithms on various multi-class imbalanced datasets. In addition, based on the obtained results for performance evaluation metrics and a recently proposed criteria for comparing metrics, the selected metrics are compared to determine a suitable performance metric for multi-class imbalanced datasets. The experimental studies show that the CatBoost and LogitBoost algorithms are superior to other boosting algorithms on multi-class imbalanced conventional and big datasets, respectively. Furthermore, the MMCC is a better evaluation metric than the MAUC and G-mean in multi-class imbalanced data domains.

Keywords: Boosting algorithms, Imbalanced data, Multi-class classification, Ensemble learning

Introduction

Imbalanced data set classification is a relatively new research line within the broader context of machine learning studies, which tries to learn from the skewed data distribution. A data set is imbalanced when the samples of one class consist of more instances than the rest of the classes in two-class and multi-class data sets [1]. Most of the standard machine learning algorithms show poor performance in this kind of datasets, because they tend to favor the majority class samples, resulting in poor predictive accuracy over the minority class [2]. Therefore, it becomes tough to learn the rare but

Tanha *et al. J Big Data*   (2020) 7:70

Page 2 of 47

important instances. In fact, they assume equal misclassification cost for all samples for minimizing the overall error rate.

Learning from skew datasets becomes very important when many real-world classification problems are usually imbalanced, e.g. fault prediction [3], fraud detection [4], medical diagnosis [5], text classification [6], oil-spill detection in satellite images [7] and cultural modeling [8]. An equal misclassification cost associated to each of the classes in these datasets is not true. For example, in software fault prediction, if the defective module is regarded as the positive class and non-defective module as negative, then missing a defect (false negative) is much expensive than the false-positive error in testing phase of software development process [9].

In recent years, several proposals have been made to deal with the class imbalance problem that can be broadly categorized into two groups: data level and algorithm level. Data level approaches rebalance the data distribution by resampling methods. These methods solve the problem by either increasing the minority class or decreasing the majority class observations. Many different sampling techniques such as random over-sampling, random under-sampling, synthetic minority over-sampling technique (SMOTE), direct oversampling, and other related methods have been proposed, each with their pros and cons [10]. The algorithm level approaches modify the machine learning algorithm in such way that it accommodates imbalance data. The modification can be considering different misclassification costs for each class, also known as cost-sensitive methods, and minimizing the cost error instead of maximizing the accuracy rate [11]. The other modification is choosing a suitable inductive bias. For example, adjusting the probability estimate at the leaf when using a decision tree as a learner, or considering multiple minimum support for different classes in association rules. Many other approaches have been proposed in the learning algorithms aiming to reinforce them towards the minority class samples, such as ensemble approaches and deep-based algorithms. However, the ensemble method is one of the most well-known approaches in this group which is our main focus in this study.

Ensemble classifiers also known as multiple classifier system, improve the performance of learning method by combining a collection of base classifiers for the classification system. The output of each base classifier is collected and used to prepare the classification decision for new samples. Specifically, in boosting, a sequential aggregate of base classifier is constructed on weighted versions of the training data, focusing on misclassified samples at each stage of generating classifiers based on the sample weights that are changed according to the performance of the classifier [12].

The imbalanced dataset problems become more complicated in multi-class imbalanced classification tasks, in which there may be multiple minority and majority classes that cause skew data distribution. In this case, for example, a class may be a minority one when compared to some other classes, but a majority of the rest of them [13]. Therefore, many new challenges are imposed which do not exist in two-class cases. Fortunately, some strategies such as decomposition methods and introducing new loss functions have been proposed to deal with multi-class classification problem [14, 15].

The recent emergence of new technologies, such as internet of things (IoT), online social network (OSN), have eased the way to originate huge amount of data called as Big data which brings some difficulties for the extracting knowledge by learning algorithms

due to their specific features, i.e., volume, velocity, variety, and veracity [16, 17]. Moreover, the imbalanced data problem can be often found in this kind of datasets and by considering multi-class property, learning from multi-class imbalanced big datasets becomes a very challenging task that has not been explored well by the researches, although this kind of data frequently appears in real-world classification problems [18].

Performance evaluation is another issue that should be addressed in this domain. It is obvious that for evaluating a classifier on imbalanced data, we require such metrics that could reflect its ability on predicting minority class(es) appropriately. Researchers have considered this issue by proposing some metrics for imbalanced data domain, such as area under the curve (AUC), Matthews correlation coefficient (MCC), G-mean, Kappa, and others that some of them have been successfully extended to multi-class problems [19, 20]. However, determining which one is more suitable for describing the performance of the classifier, needs to be examined. In this regard, some researchers have proposed frameworks for comparing evaluation metrics based on the empirical results [21, 22].

To the best of our knowledge, this is the first work to compare boosting algorithms on various multi-class imbalanced datasets. Therefore, in this study, we examine the performance of 14 most significant boosting algorithms on 19 multi-class imbalanced conventional and big datasets and compare them with each other. A comprehensive statistical test suite is used in order to investigate which boosting algorithm performs better in terms of evaluation metrics. Subsequently, three most prevalent evaluation metrics used in the imbalanced data literature are compared to determine which metric is more suitable for the multi-class imbalanced data domain.

Based on the aforementioned descriptions, the contributions and objectives of this review study are:

1. To evaluate the performance of the most significant boosting classifiers on multi-class imbalanced datasets with different imbalance ratio.
2. To determine which evaluation metric is more informative for showing the performance of a learning algorithm in the context of multi-class imbalanced datasets.
3. To evaluate the performance of the most significant boosting algorithms on multi-class imbalanced big datasets.
4. To analyze the computational time of the boosting algorithms.

The rest of this paper is structured as follows: "Handling imbalanced data problem" section reviews imbalanced data handling methods, followed by a summary on the state-of-the-art boosting algorithms in "Boosting methods" section. The experimental results and discussions are presented in "Experimental study" section. Finally, the "Conclusion" section remarks the findings of this study.

Handling imbalanced data problem

In many real-world situations, the data distribution among classes of a dataset is not uniform, such that at least one class of data has fewer samples (minority class) than the other classes (majority classes). This poses a difficulty for standard machine learning algorithms as they will be biased towards majority classes. In this situation, they will

present high accuracy on majority classes and poor performance on minority class [23], while the minority class may possess more valuable knowledge. A multi-class problem can have different styles, including many minority and majority classes, one minority class and many minority classes or one majority class and many minority classes [24]. In this condition, a class can be considered as a majority one compared to other classes, but it is considered as a minority or well-balanced one for the rest of the classes [25]. Furthermore, the difficulty increases in the case of multi-class imbalanced datasets. For handling this unwanted circumstance, researchers have proposed some methods that can be classified into two major groups: data-level methods and algorithm-level methods. The details of these methods are explained in the following sub-sections.

Data-level methods

Data-level methods balance data among classes by re-sampling approaches. These approaches are categorized into three major techniques: under-sampling, over-sampling and hybrid-sampling. These techniques include many different forms of re-sampling such as random under-sampling (samples of the majority classes are randomly removed from the dataset), directed under-sampling (the choice of samples to eliminate is informed), random over-sampling (generate new samples for minority classes randomly), directed over-sampling (no new samples are created, but the choice of samples to replace is informed), and combinations of the above techniques [10]. However, the main drawbacks of these techniques are removing useful samples or introducing meaningless new samples to the datasets.

Under-sampling methods remove samples from majority classes until the minority and majority classes become balanced. Therefore, the training process can be easily implemented, and it improves the problem associated with run time and storage [26]. The most common under-sampling method is random under-sampling (RUS) that removes samples from majority classes randomly.

Over-sampling approaches create copies of existing samples or add more samples to the minority class [27]. They increase the training set size, thus consume extra time. The simplest method to increase the size of the minority class is random over-sampling. This method increases the number of minority class samples by randomly replicating, but it is prone to overfitting [28]. For solving these problems, Chawla proposed the Synthetic Minority Over-sampling technique (SMOTE), which creates synthetic samples from the minority class. The SMOTE samples are linear combinations of two similar samples from the minority class [29].

In the hybrid-sampling methods, more than one sampling technique is used together to remove the drawback of each sampling method [30].

For multi-class datasets, the sampling strategies should be adjusted the sampling procedures to consider individual properties of classes and their mutual relations. However, the class decomposition schemes are another most existing solution in order to handle multi-class data and alleviate the difficulties of this type of datasets [31]. The advantage of such approaches is simplifying the problems into multiple two-class problem. But, there are some disadvantages with these kind of approaches, including losing the balanced performance on all classes and rejecting the global outlook on the multi-class problems. Despite the mentioned disadvantages, this direction can be considered as a

promising one. Two commonly used schemes for class decomposition are one-versus-all (OVA) and one-versus-one (OVO) schemes. In the OVA strategy, the data points of a class are considered with the data point of the other classes. In this case, for the *K*-class dataset, *K* classification problems are introduced. While in the OVO strategy, a binary classification problem is created between any two classes; therefore *K(K−1)/2* classification problems are introduced.

Algorithm-level methods

In the algorithm-level category, the existing learner is modified to remove its bias towards the majority classes. The most common approach is the cost-sensitive learning in which the learner is forced to correctly classify the minority class samples by setting a high cost to the misclassified samples of minority class [32]. For correct classification samples, no penalty is assigned, but the misclassifying cost for minority samples is higher than the majority samples. The goal is to minimize the total cost of the training dataset [33]. However, determining the cost values is not an easy task, as they depend on multi factors that have trade-off relationships.

Hybrid methods

Hybrid methods combine previously mentioned approaches. Ensemble learning is one of the most frequently used classifiers that combine data level and algorithmic level methods for handling the imbalanced data problem [34]. The main goal of the ensemble is obtaining better predictive performance than the case of using one classifier. However, its main drawback is generating more classifiers, which increases computational complexity [35]. The best-known methods of the ensemble classifiers are bagging and boosting [36, 37]. In bagging, the original training set is divided into *N* subsets of the same size, and then each subset is used to create a classifier. The whole classification model is built by aggregating particular classifiers. On the other hand, boosting algorithms generates a new weak learning model, and after many rounds, the boosting algorithms combine these weak learners into a single prediction model that will be much more accurate than anyone of the weak learners.

Ensemble learning has been investigated in order to handle multiclass imbalanced problems, as well. Specifically, hybridizing ensemble methods such as boosting, bagging and random subspace with the previously mentioned solutions (sampling or cost-sensitive) has been proved to be effective in imbalanced data problem [38]. For example, Wang and Yao compared the performance of Adaboost.NC and Adaboost in combination with random oversampling with (or without) using class decomposition for multi-class imbalanced datasets [13]. By applying ensemble learning to each sub-problem, Zhang et al. proposed a method to improve the performance of binary decomposition used in multi-class imbalanced problems [39]. Krawczyk also developed a new approach to handle multi-class imbalanced datasets by combining pairwise OVO decomposition and ensemble learning. He divided the original set into several binary sub-problems in order to simplify the classification task. Then, an ensemble classifier designed for handling such simplified binary problems, and outputs of individual classifiers were combined with pairwise coupling method [40]. Feng et al. proposed a new ensemble margin-based algorithm which emphasizes on

the use of a large number of informative low margin samples compared to margin samples by the aim of handling imbalanced data problem [41].

Boosting methods

Boosting as the most popular category of ensemble learning, builds a strong classifier by combining many weak classifiers [12]. The superiority of boosting is in its serial learning nature, which results in excellent approximation and generalization. In other words, weak classifiers are learned sequentially, aiming to reduce the errors of the previously modeled classifier. Many boosting approaches have been proposed so far. Each improves the classification performance by varying some steps of the general boosting scheme. Among the various kinds of boosting approaches, we have participated 14 well-known methods in our experimental comparisons. The methods involve AdaBoost.MH, SAMME, LogitBoost, GradientBoost, XGBoost, LightGBM, CatBoost, SMOTEBoost, RUSBoost, MEBoost, AdaCost, AdaC1, AdaC2 and AdaC3. We have chosen these approaches due to their high-citation count and popularity. We have selected representative approaches from all the categories of boosting. AdaBoost.MH, SAMME, LogitBoost, GradientBoost, XGBoost, LightGBM and CatBoost are general boosting algorithms applicable for all applications. Besides, SMOTEBoost, RUSBoost, MEBoost, AdaCost, AdaC1, AdaC2 and AdaC3 are specifically developed for imbalanced data classification problem. Among these methods, SMOTEBoost and RUSBoost are data-level approaches and the rest are algorithm-level. The detailed descriptions of these approaches are provided in "General Boosting approaches" section. Furthermore, a brief description for some other known boosting methods is provided in Table 1. The description of boosting approaches involves their exclusive features, discriminations and proposal objectives. The presented techniques are sorted based on their publication year and cover most of the methods from 1997 to 2019.

General Boosting approaches

AdaBoost.MH

AdaBoost.MH, as a boosting approach proposed in 2000, is an extension of the AdaBoost algorithm. In order to deal with multi-class classification, AdaBoost.MH decomposes a multi-class problem into $K(K-1)/2$ binary problems (K is the number of classes) and applies a binary AdaBoost procedure to each of the binary datasets [42]. In order to better understand the AdaBoost.MH, binary AdaBoost algorithm is explained by details.

AdaBoost (Adaptive Boosting), which is one of the primary and well-known boosting approaches, is proposed by Freund and Schapire in 1997. In its procedure, an ensemble of M classifiers is learned sequentially [43]. The main idea of AdaBoost is bolding the effect of misclassified samples by increasing their weights in the iterations of boosting. Hence, in all iterations of the algorithm, the sample set is fixed, and only their weights are changed. By doing so, classifiers can learn from the errors of the current classifiers and improve the accuracy. AdaBoost uses exponential loss function as Eq. (1).

$$L_{Adaboost}(y,f) = \exp\left(-y^{\mathrm{T}}f\right) \tag{1}$$

The boosting procedure in AdaBoost is as follows. At first, all n samples get the same weight $w_i^1 = 1/n$. Then in each iteration of the boosting, a classifier $f^m(x)$ is selected with

Tanha *et al. J Big Data* (2020) 7:70

Page 7 of 47

Table 1 Overview of the boosting approaches

Approach	Brief description	Years
AdaBoost.M1 [43]	A multiclass variation of AdaBoost which uses multiclass base classifier Weight of each base classifiers is a function of error rate	1997
AdaBoost.M2 [43]	A multiclass variation of AdaBoost Weight of each base classifiers is a function of pseudo-loss	1997
GentleBoost [45]	Extended version for AdaBoost which uses Newton steps Using weighted least-squares regression for fitting the base classifiers	2000
CSB1 [58]	A Cost-sensitive variation of AdaBoost proposed for handling imbalanced data Adding cost item into the weight update formula of AdaBoost Removing step size coefficient from the weight update formula of AdaBoost	2000
CSB2 [58]	A Cost-sensitive variation of AdaBoost proposed for handling imbalanced data Adding cost item into the weight update formula of AdaBoost The step size is considered in the weight update formula, like AdaBoost	2000
MAdaBoost [59]	Proposed with the goal of solving the AdaBoost's sensitivity to noise Modifying the weight update formula of AdaBoost	2000
RareBoost [60, 61]	An improvement for AdaBoost Using different weight update scheme for positive and negative predictions Considering False Positive, True Positive, True Negative and False Negative in step size calculation	2001
Modest AdaBoost [62]	An improvement of GentleBoost Using different weight update formula for misclassified and truly classified samples Using inverted distribution to assign larger weights to truly classified samples	2005
JOUSBoost [63]	Proposed with the goal of handling imbalanced data in AdaBoost algorithm Combining the jittering of the data and sampling techniques with AdaBoost	2007
ABC-LogitBoost [47]	An improvement of LogitBoost for multiclass classification Solving the difficulties of dense Hessian Matrix in Logistic loss	2009
AdaBoost.HM [64]	A multiclass variation of AdaBoost which uses hypothesis margin Using multiclass base classifiers instead of decomposing the multiclass classification problem into multiple binary problems	2010
RAMOBoost [65]	Proposed with the goal of imbalanced data handling Combining Ranked Minority Oversampling with AdaBoost.M2 Using the sampling probability distribution for ranking the minority class samples	2010
AOSO-LogitBoost [48]	One versus one version of LogitBoost for multiclass classification Solving the difficulties of dense Hessian Matrix in Logistic loss by utilizing vector tree and adaptive block coordinate descent techniques	2011
CD-MCBoost [66]	Performing coordinate descent on multiclass loss function Concentration of each base classifier on margin maximization of a single class	2011
EUSBoost [67]	An improvement of RUSBoost which uses evolutionary undersampling Using different subsets of majority class samples in the training phase of each base classifier to ensure diversity	2013
RB-Boost [68]	Combining Random Balance with AdaBoost.M2 Using SMOTE sampling to deal with imbalanced data problem The difference with SMOTEBoost is using random proportion of classes in each iteration of booting to ensure the diversity of base classifiers	2015
LIUBoost [69]	Proposed with the goal of imbalanced data handling Using undersampling in order to solve the imbalanced data problem Adding a cost term to the weight update formula of the samples	2019
TLUSBoost [70]	Proposed with the goal of imbalanced data handling Using Tomek-linked and redundancy-based undersampling for removing outlier samples	2019

respect to the loss function and is fitted to the data points considering their current weights. Then, the error rate of the selected classifier is computed as Eq. (2).

$$err^{(m)} = \frac{\sum_1^n w_i^m . \lessdot \left(f^m(x_i) \neq y_i \right)}{\sum_1^n w_i^m} \tag{2}$$

Tanha *et al. J Big Data*   (2020) 7:70

Page 8 of 47

In which $f^m(x_i)$ and y_i indicates the predicted value and actual output of the sample x_i, respectively. $<\left(f^m(x_i) \neq y_i\right)$ is the indicator function with a value equal to 1 when the statement is true; otherwise, it is equal to zero.

Then the step size (or the coefficient of the $f^m(x_i)$) is calculated according to the error rate value as Eq. (3).

$$\alpha^m = \log\frac{1 - \text{err}^{(m)}}{\text{err}^{(m)}} \tag{3}$$

According to this equation, the fitted classifier of each iteration will get a high impact in the ensemble if it has a lower error rate on the training data. After calculating the error rate and step size of the classifier. The misclassified samples are reweighted according to Eq. (4).

$$w_i^{m+1} = w_i^m.\exp\left(\alpha^m.<\left(f^m(x_i) \neq y_i\right)\right) i = 1, \ldots, n \tag{4}$$

When M classifiers are produced, the final ensemble model is constructed by Eq. (5).

$$F(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha^m f^m(x)\right) \tag{5}$$

The pseudo-code of the AdaBoost algorithm is provided in Algorithm 1.

Algorithm1: AdaBoost algorithm

i.   **Input:** Training set $(x_1,y_1),...,(x_n,y_n)$, where $x_i \in X$, $y_i \in Y = \{-1, +1\}$, Base-learner algorithm, Number of iterations M.

ii. **Initialization:** Weight the training samples $w_i^1 = 1/n$, $i = 1,...,n$.

iii. **Iteration:** For $m = 1,...,M$

    (1)   Use the Base-learner algorithm to fit a classifier $f^m(x)$ to the training data using weights w_i^m.

    (2)   Calculate the training error $err^{(m)}$ of the classifier f^m:

$$err^{(m)} = \frac{\sum_1^n w_i^m.\mathbb{I}\left(f^m(x_i) \neq y_i\right)}{\sum_1^n w_i^m}$$

    (3)   Calculate the weight α^m for the classifier f^m:

$$\alpha^m = \log\frac{1 - err^{(m)}}{err^{(m)}}$$

    (4)   Update the weight of the training samples:

$$w_i^{m+1} = w_i^m.\exp\left(\alpha^m.\mathbb{I}\left(f^m(x_i) \neq y_i\right)\right) \quad i = 1,..., n$$

iv. **Output:** The final ensemble model:

$$F(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha^m f^m(x)\right)$$

SAMME

SAMME [44] is a multi-class variant of AdaBoost, <mark>which uses a multi-class exponential loss function as follows.</mark>

$$L_{SAMME}(y,f) = \exp\left(-\frac{1}{K}Y^T f\right) \tag{6}$$

In which the target variables are encoded using the codewords technique to the form $y_i = (y_1, \ldots, y_K)^T, i = 1, 2, \ldots, N$, where

$$y_k = \begin{cases} 1 & if y_i = C_k \\ -\frac{1}{K-1} & otherwise \end{cases} \tag{7}$$

In this algorithm, the boosting procedure is exactly the same as the AdaBoost with minor differences. The error rate calculation for the weak learners and the reweighting phase for the misclassified samples are same as the AdaBoost. The difference is in the extra term $log(K-1)$ in the step size calculation equation as following equation.

$$\alpha^m = \log\frac{1 - \text{err}^{(m)}}{\text{err}^{(m)}} + \log(K-1) \tag{8}$$

The final ensemble model has the following form

$$F(x) = \underset{k}{\text{argmax}} \sum_{m=1}^{M} \alpha^m . \prec \left(f^m(x) = k\right). \tag{9}$$

LogitBoost

LogitBoost is one of the state-of-the-art approaches proposed by Friedman et al. in 1998 [45, 46]. The LogitBoost utilizes adaptive Newton steps to fit an additive logistic model. So, LogitBoost is a boosting approach with logistic loss function (negative conditional log-likelihood) as Eq. (10).

$$L_{LogitBoost}(y,f) = -\sum_{k=1}^{K} \prec \left(k = y\right) \log P_k \tag{10}$$

where $P_k(x) = \frac{\exp(F_k(x))}{\sum_{j=1}^{K} \exp(F_j(x))}$ and $\sum_{k=1}^{K} F_k(x) = 0$.

The boosting procedure in K-class LogitBoost is as follows. At first, all n samples get the same weight $w_i^1 = 1/n$. The initial probability for each class for samples is set to $P_k(x) = 1/K, k = 1, \ldots, K$.

At each iteration of the LogitBoost, K independent classifiers that each minimizes the overall loss function with respect to the kth class, are fitted. So, an ensemble of classifiers is considered for each class. The final ensemble model is chosen by Eq. (11).

$$F(x) = \underset{k}{\text{argmax}} F_k(x). \tag{11}$$

where $F_k(x) = \sum_{m=1}^{M} f_k^m(x) = \sum_{m=1}^{M} \frac{K-1}{K}\left(f_k^m(x) - \frac{1}{K}\sum_{j=1}^{K} f_j^m(x)\right)$.

LogitBoost algorithm was designed as an alternative solution to address the limitations of AdaBoost in handling noise and outliers. It is less sensitive to outliers and noise due to using a logistic loss function that changes the loss function linearly. In contrast, AdaBoost uses an exponential loss function that changes exponentially with the classification error, which makes AdaBoost more sensitive to noise and outliers.

LogitBoost supports both binary and multi-class classification. This algorithm can handle multi-class problems by considering multi-class logistic loss. Also, it can address multi-class classification problems by using a parametric method. Some other multi-class variants of LogitBoost, have been proposed too, e.g., ABC-LogitBoost [47] and AOSO-LogitBoost [48] (which is a one-vs-one LogitBoost for multiclass problems).

GradientBoost

GradientBoost is based on trees which is applicable for various kinds of loss functions. Like other boosting algorithms, it builds a stage-wise additive model. Unlike traditional boosting in which weak-learners fit a model to the output values of samples, in each iteration of this algorithm, the decision trees are generated by fitting the negative gradients [49]. Negative gradient is also called residual error that is a function of the difference between the predicted value and the real value of the output. Generally, in Gradient-Boost the model is initialized with a constant value γ (A tree with just one leaf node) that minimizes the loss function over all the samples as Eq. (12).

$$F^0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma) \tag{12}$$

Then, in each iteration, the negative gradient of the loss function is computed with respect to the current ensemble as following equation.

$$r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F^{m-1}(x)}, \quad i = 1 \, to \, n \tag{13}$$

A decision tree is fitted on the negative gradients (residuals) and then the values of the leaf nodes are computed with the goal of loss function minimization as Eq. (14).

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L\left(y_i, F^{m-1}(x_i) + \gamma\right), \quad j = 1, \ldots, J_m \tag{14}$$

where, J_m is the number of the leaf nodes in the mth iteration tree, and γ_{jm} is the value of jth leaf node at iteration m. Note that the summation of loss values is calculated over the samples that belong to that leaf nodes region R_{jm}. Finally, the new model is added to the ensemble as the following equation.

$$F(x) = F^{m-1}(x) + \nu f^m(x) \tag{15}$$

Besides the effective process of boosting in GradientBoost, a shrinkage metric ν is also for controlling overfit to training data. In other words, in each boosting iteration, the selected model is multiplied by a coefficient between 0 and 1, which indicates the learning rate (shrinkage value). GradientBoost supports both binary and multi-class classifications.

CatBoost

CatBoost is a novel algorithm for gradient boosting on decision trees, which has the ability to handle the categorical features in the training phase. It is developed by Yandex researchers and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in some other

companies. CatBoost has both CPU and GPU implementations which accelerates the training process [50].

Like all gradient-based boosting approaches, CatBoost consists of two phases in building trees. The first is choosing the tree structure and the second is setting the value of leaves for the fixed tree. One of the important improvements of the CatBoost is doing unbiased gradient estimation in order to control the overfit. To this aim, in each iteration of the boosting, for truly estimating the gradient of each sample, it excludes that sample from the training set of the current ensemble model. The other improvement is the automatic transformation of categorical features to numerical features without any preprocessing phase. CatBoost is applicable for both binary and multi-class problems.

XGBoost

XGBoost is an optimized version of GradientBoost, which is introduced by Chen in 2016. XGBoost has improved the traditional GradientBoost by adding up some efficient techniques to control overfitting, split finding and handling missing values in the training phase [51]. In order to control the overfitting, the objective (minimization) function consists of two parts: loss function and regularization term, which controls the complexity of the model as Eq. (16).

$$Obj = \sum_{i=1}^{n} l(y_i, f_i) + \sum_{m=1}^{M} \Omega(f^m) \tag{16}$$

where, $\Omega(f^m)$ is the regularization term. In XGBoost, a second-order approximation is used to optimize the objective function. Accordingly, in each iteration, the best tree is selected by using Eq. (17).

$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T \tag{17}$$

where, T is the number of leaf nodes. G_j and H_j are the summations of the first and second-order gradient statistics on the loss function over the samples of the j-th leaf node, respectively. λ and γ are the regularization coefficients. Thus, in this approach, the complexity of the tree is chosen and controlled separately in each iteration and the number of the leaves is not a fixed value during all the iterations. After the best tree selection in each iteration, the values of the leaf nodes are calculated by the use of gradient statistics of each leaf by the use of Eq. (18).

$$w_j^* = -\frac{G_j}{H_j + \lambda} \tag{18}$$

In addition, an approximate technique is used for split finding in each node of the tree. To this aim, for each feature, all the instances are sorted by that feature's value, then a linear search is done to find the best split along that feature. The best split is chosen by comparison among the best split of all the features. Furthermore, after constructing the tree, in each node the direction with maximum score is marked to make a default path on the tree for classification of data with missing values.

XGBoost algorithm has made excellent improvements in the implementation level. It utilizes parallel learning, out-of-core computation and cache-aware access techniques to minimize the time complexity of learning, thus it is applicable for very large datasets.

LightGBM

LightGBM, as an efficient implementation of GradientBoost, is proposed by Ke et al. in order to solve the efficiency and scalability problem of GradientBoost in the case of high dimensional feature space or large data size. In the GradientBoost, for each feature, all the data samples are needed to be scanned and all the possible split points are examined. Therefore, the most time-consuming part of the GradientBoost is the splitting procedure in each tree node. In the this algorithm, two novel techniques called Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) are proposed. In the GOSS technique, only the small set of samples which have large gradients are considered in the split point selection. This works, because samples with larger gradients play a more important role in the computation of the information gain. In the FEB technique, in order to reduce the number of features, mutually exclusive features are bundled with the use of a greedy approach. LightGBM only supports binary problems [52].

Boosting for imbalanced data
SMOTEBoost

SMOTEBoost, proposed by Chawla et al., is a data-level method to deal with the imbalanced data problem. The main steps of the proposed approach are SMOTE sampling and boosting. This algorithm uses SMOTE technique as a data level solution. SMOTE adds new minority class examples to a training dataset by finding the *k*-nearest neighbors (KNN) of a minority class example and extrapolating between that example and its neighbors to create new examples. Im addition, it adopts the AdaBoost algorithm as its ensemble approach. In each iteration of the SMOTEBoost, the sampling is done based on the original data, but the training process is done based on all the original and synthetic data. Like AdaBoost, after training a classifier the weights of the misclassified samples are updated, but here the update is done only for the original data. Thus, after each step of the boosting, the weights of the original data change and new synthetic data are generated. One of the main drawbacks of the SMOTEBoost is generating a large training dataset and, therefore, relatively slow in training phase of the boosting algorithm. SMOTEBoost supports both binary and multi-class problems [53].

RUSBoost

RUSBoost, as a data-level solution, is proposed by Seiffert et al. to reduce the computational requirement of SMOTEBoost. RUSBoost combines data sampling and the AdaBoost algorithm. It uses RUS (Random Under Sampling) sampling approach to balance the initial data. RUS technique randomly removes the examples from the majority class. Each iteration of the boosting consists of two phases. In the first phase of the algorithm, random under-sampling is performed on all the classes except the minority class. Under-sampling continues until all the classes reach the same size (minority class size). In the second phase of the algorithm, AdaBoost is applied on the sampled data. So, the training set is not fixed over all the iterations. One of the

drawbacks of the RUSBoost is its data loss in the under-sampling level. It supports both binary and multiclass problems [54].

MEboost

MEBoost is a novel boosting approach that is proposed for handling imbalanced data with an algorithm level solution. MEBoost mixes two different weak learners with boosting to improve the performance and taking the benefits of both classifiers. In each iteration of the boosting, it either uses a decision tree (which uses information entropy as splitting metric) or an extra tree classifier (Randomized tree classification) as its learner. In this algorithm, the ensemble model is tested according to the AUC (area under Receiver Operating characteristic curve) score. It keeps adding weak learners to the model until there is no significant change in the auROC on the test data. Furthermore, MEBoost has an early stopping criterion for the case that the improvement between two iterations of boosting is not significant. MEBoost only supports binary problems [55].

AdaC

AdaC is proposed by Sun et al. in order to improve the imbalanced data classification. The main idea of this approach is to take an algorithm level solution to handle the imbalanced data by adding a cost item into the learning procedure of the AdaBoost. The cost item $C_i \in [0, +\infty]$ (for each sample i) is added in order to increase the impact of minority class samples in the training phase and improve the performance. To this aim, the cost item is set to higher values for minority class samples and smaller values for majority class samples. The cost value of each sample shows the importance of correctly classifying that sample [56].

AdaC algorithm has three versions, namely AdaC1, AdaC2 and AdaC3. The difference between these three versions is in the way of adding cost to the weight update equation of the AdaBoost. In the AdaC1 algorithm, the cost item is embedded inside the exponent part. The weight update equation of AdaC1 is provided in Eq. (19).

$$w_i^{m+1} = \frac{w_i^m . \exp\left(-\alpha^m C_i f^m(x_i) y_i\right)}{\sum_{j=1}^n w_j^m . \exp\left(-\alpha^m C_j f^m(x_j) y_j\right)} \quad i = 1, \ldots, n \tag{19}$$

Moreover, the step size parameter α^m is modified to the form shown in Eq. (20).

$$\alpha^m = \frac{1}{2} \log \frac{1 + \sum_{i:f^m(x_i)=y_i} C_i w_i^m - \sum_{i:f^m(x_i)\neq y_i} C_i w_i^m}{1 - \sum_{i:f^m(x_i)=y_i} C_i w_i^m - \sum_{i:f^m(x_i)\neq y_i} C_i w_i^m} \tag{20}$$

In the AdaC2 algorithm, the cost item is embedded outside the exponent part. The weight update equation of AdaC2 is shown in Eq. (21).

$$w_i^{m+1} = \frac{C_i w_i^m . \exp\left(-\alpha^m f^m(x_i) y_i\right)}{\sum_{j=1}^n C_j w_j^m . \exp\left(-\alpha^m f^m(x_j) y_j\right)} \quad i = 1, \ldots, n \tag{21}$$

The step size parameter α^m is modified to the form shown in Eq. (21).

$$\alpha^m = \frac{1}{2}\log\frac{\sum_{i:f^m(x_i)=y_i}C_iw_i^m}{\sum_{i:f^m(x_i)\neq y_i}C_iw_i^m} \tag{22}$$

AdaC3 algorithm is the combination of AdaC1 and AdaC2, in which the cost item is embedded both inside and outside the exponent part as Eq. (23).

$$w_i^{m+1} = \frac{C_iw_i^m.\exp\left(-\alpha^m C_if^m(x_i)y_i\right)}{\sum_{j=1}^n C_jw_j^m.\exp\left(-\alpha^m C_jf^m(x_j)y_j\right)} i = 1,\dots,n \tag{23}$$

And the step size parameter α^m has the form provided in Eq. (24).

$$\alpha^m = \frac{1}{2}\log\frac{\sum_i C_iw_i^m + \sum_{i:f^m(x_i)=y_i}C_i^2w_i^m - \sum_{i:f^m(x_i)\neq y_i}C_i^2w_i^m}{\sum_i C_iw_i^m - \sum_{i:f^m(x_i)=y_i}C_i^2w_i^m + \sum_{i:f^m(x_i)\neq y_i}C_i^2w_i^m} \tag{24}$$

AdaC algorithms only support the binary classification.

AdaCost

AdaCost as another algorithm level cost-sensitive AdaBoost approach is proposed by Fan et al. in 1999 [57]. The main idea is to emphasize on the minority class classification accuracy and adding larger cost values to the minority class samples. In this approach, the weight of misclassified minority class samples increases higher than misclassified majority class samples. Moreover, the weight of a truly classified sample decreases less if it belongs to the minority class. As a result, the weight update equation is changed by adding a cost adjustment function β, which is a function of cost item C as Eq. (25).

$$w_i^{m+1} = \frac{w_i^m.\exp\left(-\alpha^m f^m(x_i)y_i\beta_{\text{sign}(f^m(x_i).y_i)}\right)}{\sum_{j=1}^n w_j^m.\exp\left(-\alpha^m f^m(x_j)y_j\beta_{\text{sign}(f^m(x_j).y_j)}\right)} i = 1,\dots,n \tag{25}$$

where, β_+ refers to the case $\text{sign}\left(f^m(x_i).y_i\right) = +1$ and β_- refers to the case $\text{sign}\left(f^m(x_i).y_i\right) = -1$. In the main reference, the optimum value for β is recommended as $\beta_+ = -0.5C_i + 0.5$ and $\beta_- = 0.5C_i + 0.5$. Hence, the step size parameter α^m is modified to Eq. (26).

$$\alpha^m = \frac{1}{2}\log\frac{1 + \sum_{i=1}^n w_i^m.\exp\left(-\alpha^m f^m(x_i)y_i\beta_{\text{sign}(f^m(x_i).y_i)}\right)}{1 - \sum_{i=1}^n w_i^m.\exp\left(-\alpha^m f^m(x_i)y_i\beta_{\text{sign}(f^m(x_i).y_i)}\right)} \tag{26}$$

AdaCost algorithm only supports binary classification.

Experimental study

Dataset characteristics

The aforementioned boosting algorithms are evaluated on 15 conventional and 4 big multi-class imbalanced datasets from UCI[1] and OpenML[2] repositories. The characteristics of these datasets are given in Table 2.

---

[1] https://archive.ics.uci.edu/ml/index.php

[2] https://www.openml.org/search?type=data

Tanha *et al. J Big Data*   (2020) 7:70

Page 15 of 47

Table 2 Characteristics of the test datasets

Dataset	# of Attributes	Instances	# of classes	IR
Conventional datasets				
Wine	13	178	3	1.47
Hayes-Roth	4	132	3	1.7
Contraceptive	9	1473	3	1.89
Pen-Based	16	1100	10	2.18
Vertebral column	6	310	3	2.5
New thyroid	5	215	3	5
Dermatology	34	366	3	5.6
Balance Scale	4	625	3	5.8
Glass	9	214	7	8.44
Heart (Cleveland)	13	303	5	12.62
Car Evaluation	6	1728	4	18.61
Thyroid	21	7200	3	40.15
Yeast	8	1484	10	92.5
Page blocks	10	5473	5	175.46
Shuttle	9	58,000	7	4558.6
Big datasets				
FARS	29	100,968	8	4679
KDD Cup'99	41	494,021	5	1870
Covertype	54	581,012	7	103
Poker	10	1,000,000	10	64,212

Performance metrics

Using evaluation metrics is an essential factor to assess the classification performance of a learning algorithm. Accuracy and error rate are the most widely used metrics for this purpose. However, they are biased toward the majority class in imbalanced datasets as described in "Handling imbalanced data problem" section. Therefore, the values of these popular metrics do not show the ability of the classifier to predict examples from minority classes.

Alternative criteria have been used for this scenario that can be roughly categorized into single-class and overall performance measures. The first group category of criteria shows how well a classifier performs in one class and the second one allows to have a clear and intuitive interpretation on the performance of the classifier on all classes.

Although in the literature, many evaluation metrics have been proposed to measure the classifier performance when dealing with imbalanced data problem, Precision, Recall, F-measure, AUC, G-mean, Kappa, and MCC are the most prevalently used metrics in the two above mentioned categories. All of these metrics are calculated from the confusion matrix. A confusion matrix is a two-dimensional matrix that presents a clear explanation of how a classifier works in predicting new samples [71]. One dimension is indexed by the actual class of an object and the other by the class that the classifier predicts. Figure 1 shows the confusion matrix for a multi-class classification task. The entry N_{ij}, with $i \neq j$, shows the number of samples that classified as class C_j but actually belong to class C_i. The best classifier will have zero values in non-diagonal entries.

Like the previous studies, the minority class is considered as positive, while the majority class is considered as negative.

		Predicted		
		$\mathbf{C_1}$	$\mathbf{C_2}$	$\cdots$	$\mathbf{C_n}$
Actual	$\mathbf{C_1}$	N_{11}	N_{12}	$\cdots$	N_{1n}
	$\mathbf{C_2}$	N_{21}	N_{22}	$\cdots$	N_{2n}
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$\mathbf{C_n}$	N_{n1}	N_{n2}	$\cdots$	N_{nn}

Fig. 1 Confusion matrix for multi-class classification

Single-class measures

Single-class metrics are calculated for each class and are less sensitive to class imbalance, therefore, naturally are better suited to evaluate classifiers in skew data domains. In this section, the used single-class metrics are briefly reviewed.

Precision metric measures the correctly classified positive class samples and defines as

$$Precision = \frac{TP}{TP + FP} \tag{27}$$

where *TP* and *FP* indicate the counts of true-positive and false-positive, respectively.

Recall measures the proportion of correctly identified of all real positive samples and computed by Eq. (28):

$$Recall = \frac{TP}{TP + FN} \tag{28}$$

where *FN* indicates the counts of false-negative.

In general, there is a trade-off between precision and recall. To have an intuitive view of the performance of the classifier based on these metrics, *F*-measure [72] can be used. It is a harmonic mean of precision and recall in which the user can adjust the relative importance to recall as to precision by the β parameter. Usually, the value of this parameter is taken as 1. The following equation show how F-measure is computed:

$$F-measure = \frac{(1 + \beta)PrecisionRecall}{\beta^2 Precision + Recall} \tag{29}$$

F-measure can be extended by micro-averaging or macro-averaging methods for multi-class cases. In micro-averaging, *F*-measure is computed globally over all classes and its elements (Precision and Recall) are obtained by summing over all classes, while in macro-average, it is obtained for each class locally, and then by averaging those, a single value can be computed as a *F*-measure value for whole classes.

Due to the independence of the imbalanced distribution, Geometric-mean (*G*-mean) is another suitable single metric for imbalanced data problems [19] that is defined as Eq. (30)

$$G-mean = \sqrt{\frac{TP}{TP + FN} \frac{TN}{TN + FP}} \tag{30}$$

Like *F*-measure, this metric can be extended by micro-averaging or macro-averaging methods for the multi-class problems, however, in this study the method that was proposed

by Sun et al. [19] is used to extend it to multi-class domains by the geometric mean of recall values of all classes as defined in Eq. (31).

$$G{-}mean = \left(\prod_{i=1}^{C} Recall_i \right)^{\frac{1}{c}} \tag{31}$$

Although both *G*-mean and *F*-measure measures can be used to evaluate the performance of a learning method in class imbalanced classification problems, *F*-measure may get infinity when classifier all samples were classified as negative class. Therefore, in this study we will use *G*-mean as a single class performance evaluation criterion.

Overall metrics

The area under the curve (AUC) of the receiver operating characteristic (ROC) is extremely used as an overall evaluation technique, especially for ranking classifiers in the presence of class imbalanced for binary classifiers. The ROC curve shows all possible conflicts between true-positive rate (TPR) and false-positive rate (FPR) across various decision thresholds and AUC evaluation metric converts this curve to a value in the range of [0.5, 1], where the value 1 shows a perfect classifier and the lower value 0.5 means the classifier does not work better than random guess [73].

Although extending the AUC to multi-class problems is an open research topic, the MAUC metric [20] which averages the AUC value of all pairs of classes, are mostly used in the researches multi-class imbalanced data learning and defined as

$$MAUC = \frac{2}{C(C-1)} \sum_{i<j} A(i,j) \tag{32}$$

where *C* denotes the number of classes, *A(i, j)* is the AUC between class *i* and class *j*.

Matthews Correlation Coefficient (MCC) is the third overall evaluation metric used in this study to evaluate the performance of boosting algorithms, which is especially utilized in biomedical imbalanced data analysis [74]. MCC contributes all classes' values in the confusion matrix for computing a measure of the correlation between actual and predicted samples. Its value is located between -1 and $+1$. The value $+1$ means perfect prediction and -1 inverse prediction [75]. This metric can be directly calculated from the confusion matrix using Eq. (33)

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{33}$$

Furthermore, MCC is a binary classification evaluation metric and how it performs in the multi-class imbalanced dataset has not yet been well studied. However, it has been shown that the behavior of MCC remains consistent in multi-class settings [76]. Based on these facts and inspired by the MAUC metric, we extend the MCC metric to multi-class case, called MMCC, by employing the OVO decomposition strategy. As MAUC, MMCC average the MCC value of all pairs of classes and it is defined as Eq. (34).

$$MMCC = \frac{2}{C(C-1)} \sum_{i<j} MCC(i.j) \tag{34}$$

Only the performance parameter of algorithms are given

Parameter settings

The Performance of the algorithms is investigated with default parameter settings, except the nearest neighbor parameter for SMOTEboost is set to 3, and the population of the minority class C is increased to be approximately equal to the size of the majority class. In addition, since in gradient boosting methods the value between 8 and 32 is usually used for the number of leaves in the tree, the C4.5 decision tree is employed with depth 5 as a base learner in all boosting algorithms. Each boosting consists of 100 trees with the learning rate of 0.3.

For binary boosting algorithms, One-Versus-One (OVO) decomposition method is obtained to split multi-class datasets to multiple binary problems and then the voting aggregation strategy is used in order to detect the class of new data samples from the aggregated classifiers [77].

In the experiments, the means and standard deviations of three performance criteria, which were described in the previous section, are calculated for the algorithms, and the procedures are repeated 10 times by employing fivefold cross-validation to reduce any bias because of the randomness. For creating the folds, the stratified sampling method is used instead of simple random sampling to guarantee that each fold contains a number of samples from each class especially minority class.

Statistical comparison of boosting algorithms

For comparing more rigorously the difference between classification algorithms, a statistical test suite is carried out on the experimental results to validate their results further and determine whether there exists a significant difference among them.

The statistical test procedure can be categorized into parametric (e.g., paired t-test and ANOVA) and nonparametric methods (e.g., the Wilcoxon and Friedman tests) from a methodological perspective. Although parametric tests are more robust than nonparametric, they are based on strict assumptions that are often violated in machine learning studies and make their results to be unreliable [78]. Therefore, nonparametric tests are preferred for comparing the results of classifiers.

According to the recommendation made by Demšar [79], in this study, the Friedman test is used to compare multiple classifiers over multiple datasets. The null-hypothesis (there is no significant difference between the performance of all algorithms) is rejected at a specified level of significance when Friedman test statistic (χ_F^2), computed as Eqs. (35) and (36), is greater than the critical value from the chi-square table. The rejection of the null-hypothesis implies the existence of at least two classification algorithms that are significantly different.

$$\chi_F^2 = \frac{12D}{M(M+1)} \left[\sum_{j=1}^{M} AR_j^2 - \frac{M(M+1)^2}{4} \right] \tag{35}$$

$$AR_j = \frac{1}{D}\sum_{i=1}^{D} r_i^j \tag{36}$$

where D and M indicate the number of datasets and classification algorithms, AR_j is the rank of j-th classifier, and r_i^j is the rank of jth classifier on ith dataset.

After the rejection of null-hypothesis, a post-hoc test should be conducted to determine whether the control a(usually, the best one) presents significantly statistical differences than the rest. In this study, the Holm post-hoc test [79] is used for this purpose. In this test, the test statistic is computed as Eq. (37), and the Z value is used to find the p-value from the table of normal distribution based on a level of significance of $\alpha = 0.05$. The null-hypothesis is rejected if the p-value is lower than the Holm's critical value.

$$Z_i = (R_i - R_j)/\sqrt{\frac{M(M+1)}{6D}} \tag{37}$$

where R_i and R_j are the rank of ith and jth classifier, respectively.

However, the Holm test does not detect the difference between a pair of algorithms. For pairwise comparison and determining the existence of significant differences between a pair of classifiers, the Wilcoxon rank-sum test [80] is adopted.

Comparison of evaluation metrics

One of the key challenges in imbalanced data classification is selecting an appropriate evaluation metric for a better comparison of classification algorithms. To this aim, the statistical test proposed in [21] is carried out for comparing the performance of evaluation metrics. Firstly, the consistency of the evaluation metrics with each other are examined. Secondly, the degree of discriminancy of the metrics is investigated. Furthermore, the degree of Indifferency of metrics is tested as proposed in [22].

Degree of consistency

For a better comparison of two evaluation metrics on two algorithms, at least their manner should be consistent [74]. Two evaluation metrics are consistent with each other in the evaluation of two algorithms A and B, when metric f states that aA is better than aB, metric g doesn't indicate that aB is better than aA. The degree of consistency (DoC) of two evaluation metrics f and g is calculated as follows.

$$C = \frac{|R|}{|R| + |S|} \tag{38}$$

where R and C are defined as Eqs. (39) and (40) in the domain Ψ.

$$R = \{(a,b)|a, b \in \Psi, f(a) > f(b), g(a) > g(b)\} \tag{39}$$

$$S = \{(a,b)|a, b \in \Psi, f(a) > f(b), g(a) < g(b)\} \tag{40}$$

The value C shows the degree of consistency of two metrics f and g on algorithms A and B that its value is located in the interval $[0, 1]$.

Degree of discriminancy

Besides the consistency of two evaluation metrics, there exist some cases that metric f can distinguish between two algorithms and states that for example, aA is better than aB, but the metric g cannot distinguish between these two algorithms. In this case, metric f is more discriminating than metric g. The degree of discriminancy (DoD) of metric f over metric g is calculated as follows.

$$D = \frac{|P|}{|Q|} \tag{41}$$

where P and Q are defined as Eqs. (42) and (43) in the domain Ψ.

$$P = \{(a,b)|a,b \in \Psi, f(a) > f(b), g(a) = g(b)\} \tag{42}$$

$$Q = \{(a,b)|a,b \in \Psi, f(a) = f(b), g(a) > g(b)\} \tag{43}$$

The value D shows the degree of discriminancy of metric f over metric g on algorithms A and B. In order to claim that one metric is better than another one, the condition $C > 0.5$ and $D > 1$ should be satisfied. In this case, we can conclude that two metrics f and g are statistically consistent with each other and metric f can D times better tell the difference between two algorithms.

Degree of indifferency

There exist some cases in comparison that none of the two metrics can report the difference between two algorithms A and B. The degree of Indifferency (DoI) of two evaluation metrics f and g is calculated as follows.

$$I = \frac{|V|}{|U|} \tag{44}$$

where V and U are defined as Eqs. (45) and (46) in the domain Ψ.

$$V = \{(a,b)|a,b \in \Psi, a \neq b, f(a) = f(b), g(a) = g(b)\} \tag{45}$$

$$U = \{(a,b)|a,b \in \Psi, a \neq b\} \tag{46}$$

The value I shows the degree of Indifferency for metrics f and g on algorithms A and B.

Experimental results and analysis

In this section, which consists of two parts, we empirically compare the boosting algorithms that have been reviewed in "General Boosting approaches" section. In the first part, the performance of the algorithms and their computational time are compared on 15 multi-class imbalanced conventional datasets through various statistical tests. Then, the best evaluation metric is determined based on the obtained results. In the second part, the performance of the algorithms is investigated on 4 multi-class imbalanced big datasets.

The experiments were conducted using a computer system with Intel Core i3 2.13 GHz CPU, 8 GB RAM running Microsoft Windows 10 64-bit operating system. The overall

results were obtained by averaging the values of evaluation criteria over 10 runs of the algorithms (10 × 5 cross-validation). For the sake of clarity, it should be noted that the library of all algorithms were installed using the pip Python installer, e.g., *sudo pip install xgboost*, except MEBoost, SMOTEBoost, and AdaCosts, which their implemented python source codes are freely available at GitHub[3] repository.

Experiments on conventional datasets

The performance of the boosting algorithms are studied on 15 multi-class imbalanced conventional datasets as follows.

MAUC as a measure

Table 3 shows the average and standard deviation of the results for MAUC obtained for the boosting algorithms. The best results are in bold for each dataset. By looking at the results in this table, although it can be seen that CatBoost presents the highest average MAUC, but for the assurance we need to conduct a non-parametric statistical test on the results. By conducting the Friedman Test, the value of 0.043 was computed for *p*-value, which rejects the null hypothesis. Moreover, the average ranks from this test are presented in Fig. 2. It is observable that CatBoost and AdaC1 achieved the lowest and highest rank, respectively. Therefore, we select this aas a control method and continue with the Holm post-hoc test. Based on the presented results for *p*-values in Table 4, it can be seen that CatBoost statistically outperforms AdaBoost, SAMME, MEBoost, and RUSBoost.

For comprehensive analysis and understanding of the difference between pairs of algorithms the Wilcoxon signed-rank test is used. The results of this test are presented in Tables 5 and 6. Table 5 shows the computed ranks by the test. Each number below the main diagonal is the sum of the ranks for the datasets that the row aoutperforms the corresponding column a(R_+), while each number above the main diagonal is the sum of the ranks for the datasets on which the column ais worse than the corresponding row a(R_-). Two classifiers are significantly different and reject the null hypothesis if the test statistic, min(R_+, R_-), is equal or less than the critical value which is 25 for the confidence level of $\alpha = 0.05$ and $N = 15$, according to the table of Wilcoxon test.

The summary of the comparison results is presented in Table 5, in which the mark ● indicates that the row asignificantly outperforms the corresponding column algorithm. As can be observed from this table, CatBoost outperforms most of the algorithms. SMOTEBoost outperforms four algorithms and we saw that it has the second rank from Fig. 2. We can see that binary classification boosting algorithms could not outperform any algorithms except AdaCost that outperforms just AdaC1 algorithm. Therefore, we can conclude that binary boosting algorithms do not show better performance in multi-class classification problems based on the MAUC criteria. Another interesting analytic finding is that the under-sampling method for encountering class imbalanced problems is not a satisfactory solution based on the obtained results for the RUSBoost algorithm, while SMOTEBoost which uses the oversampling method

---

[3] MEBoost: https://github.com/farshidrayhanuiu/MEBoost
SMOTEBoost, AdaC1, AdaC2, AdaC3, AdaCost: https://github.com/gkapatai/MaatPy

Table 3 The obtained results for MAUC from boosting algorithms using decision tree as a base learner

Dataset	Algorithm													
	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
Wine	0.9034 ±0.0183	0.9139 ±0.0128	0.9783 ±0.0054	0.9574 ±0.0098	0.9917 ±0.0025	0.9659 ±0.0129	0.9838 ±0.0077	0.9499 ±0.0167	0.9621 ±0.0091	0.9693 ±0.0152	0.9294 ±0.0168	0.9164 ±0.0162	0.9176 ±0.0134	0.932 ±0.0139
Hayes-Roth	0.8064 ±0.0155	0.7951 ±0.0104	0.7588 ±0.0154	0.8025 ±0.0087	0.8032 ±0.0138	0.7979 ±0.0151	0.7941 ±0.1901	0.7956 ±0.0153	0.7734 ±0.0292	0.5339 ±0.0343	0.8013 ±0.0107	0.8107 ±0.0115	0.8162 ±0.0197	0.8095 ±0.0173
Contraceptive	0.5281 ±0.0048	0.4944 ±0.0071	0.5305 ±0.0046	0.4904 ±0.0061	0.4939 ±0.0100	0.5366 ±0.0072	0.5032 ±0.0051	0.4801 ±0.0121	0.4458 ±0.0074	0.5005 ±0.0060	0.5455 ±0.0056	0.3479 ±0.0044	0.4422 ±0.0114	0.5524 ±0.0093
Pen-based	0.9012 ±0.0055	0.9581 ±0.0055	0.962 ±0.0045	0.9554 ±0.0053	0.9513 ±0.0030	0.9327 ±0.0057	0.9726 ±0.0037	0.9503 ±0.0027	0.9592 ±0.0047	0.9526 ±0.0020	0.9035 ±0.0096	0.9038 ±0.0056	0.9029 ±0.0061	0.9031 ±0.0079
Vertebral column	0.7758 ±0.0218	0.7758 ±0.0218	0.7999 ±0.0230	0.7814 ±0.0165	0.7794 ±0.0162	0.7824 ±0.0155	0.7814 ±0.0184	0.8069 ±0.0218	0.7851 ±0.0169	0.7957 ±0.0278	0.793 ±0.0089	0.7973 ±0.0135	0.7911 ±0.0119	0.7881 ±0.0145
New thyroid	0.8879 ±0.0217	0.8713 ±0.0269	0.8899 ±0.0126	0.9098 ±0.0206	0.9063 ±0.0117	0.8894 ±0.0273	0.6225 ±0.0217	0.9071 ±0.0354	0.884 ±0.0210	0.9097 ±0.0109	0.8987 ±0.0296	0.8728 ±0.0415	0.8855 ±0.0431	0.8833 ±0.0351
Dermatology	0.9551 ±0.0103	0.9657 ±0.0050	0.9783 ±0.0033	0.9681 ±0.0087	0.9715 ±0.0057	0.9700 ±0.0123	0.9187 ±0.01691	0.9656 ±0.0062	0.9438 ±0.0113	0.9024 ±0.0121	0.9558 ±0.0107	0.9592 ±0.0087	0.9517 ±0.0101	0.9526 ±0.0108
Balance scale	0.6131 ±0.0118	0.628 ±0.0180	0.6719 ±0.0246	0.6079 ±0.0099	0.6103 ±0.0150	0.6204 ±0.0091	0.6268 ±0.0085	0.6071 ±0.0100	0.6618 ±0.0209	0.7116 ±0.0179	0.3603 ±0.0086	0.6086 ±0.0064	0.566 ±0.0194	0.6893 ±0.014
Glass	0.6381 ±0.0248	0.6644 ±0.0324	0.6857 ±0.0244	0.7187 ±0.0229	0.6455 ±0.0485	0.7087 ±0.0246	0.6235 ±0.0388	0.6662 ±0.0192	0.5334 ±0.0437	0.4205 ±0.0042	0.6407 ±0.0324	0.6479 ±0.0183	0.5928 ±0.0356	0.6412 ±0.0169
Heart	0.2839 ±0.0136	0.2887 ±0.0112	0.3054 ±0.0131	0.2673 ±0.0147	0.2838 ±0.0171	0.2783 ±0.0181	0.272 ±0.0136	0.3216 ±0.0202	0.2949 ±0.0144	0.3182 ±0.0389	0.2853 ±0.0152	0.2781 ±0.0257	0.2923 ±0.0145	0.298 ±0.0165
Car evaluation	0.9848 ±0.0102	0.9528 ±0.0115	0.985 ±0.0034	0.998 ±0.0025	0.9851 ±0.0075	0.8471 ±0.0098	0.9909 ±0.0046	0.9589 ±0.0086	0.8439 ±0.0196	0.9843 ±0.0036	0.7711 ±0.0278	0.8967 ±0.0046	0.9478 ±0.0047	0.9698 ±0.0049
Thyroid	0.9817 ±0.0033	0.9754 ±0.0023	0.9889 ±0.0026	0.9769 ±0.0038	0.9765 ±0.0029	0.987 ±0.0022	0.9718 ±0.0070	0.9867 ±0.0021	0.9612 ±0.0168	0.8899 ±0.0325	0.6667 ±0.0000	0.9905 ±0.0022	0.9907 ±0.0010	0.6667 ±0.0000

Tanha *et al. J Big Data*   (2020) 7:70

Page 23 of 47

Table 3 (continued)

Dataset	Algorithm													
	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
Yeast	0.5088 ±0.0270	0.4092 ±0.0098	0.4757 ±0.0139	0.4958 ±0.0174	0.4619 ±0.0200	0.539 ±0.0159	0.4397 ±0.0099	0.4889 ±0.0116	0.2946 ±0.0133	0.4209 ±0.0063	0.4107 ±0.0161	0.4806 ±0.0153	0.5055 ±0.0237	0.4846 ±0.0237
Page blocks	0.8183 ±0.0149	0.7858 ±0.0198	0.8645 ±0.0095	0.8131 ±0.0173	0.3038 ±0.0462	0.8238 ±0.0213	0.8117 ±0.0188	0.825 ±0.0194	0.8322 ±0.0185	0.5566 ±0.0274	0.7457 ±0.0293	0.7391 ±0.0090	0.8895 ±0.0058	0.7455 ±0.0074
Shuttle	0.9479 ±0.0177	0.9734 ±0.0119	0.9607 ±0.0121	0.9608 ±0.0213	0.2682 ±0.0404	0.9714 ±0.0158	0.8121 ±0.0149	0.9863 ±0.0127	0.9727 ±0.0165	0.3947 ±0.0317	0.9457 ±0.0146	0.9502 ±0.0168	0.9467 ±0.0116	0.9508 ±0.0170
Average	0.76896	0.7634	0.789	0.7802	0.6954	0.7767	0.7416	0.7797	0.7432	0.684	0.7102	0.7466	0.7625	0.7511

The best performance is shown in italic for each dataset
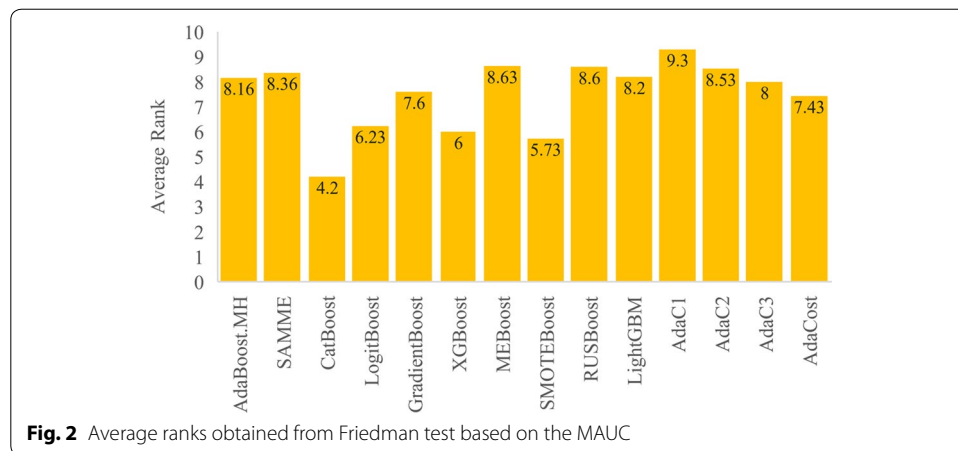
Fig. 2 Average ranks obtained from Friedman test based on the MAUC

Table 4 Holm post-hoc test results based on the MAUCs

Control method (CatBoost)

Algorithm	Z	*p*-value	Holm	Hypothesis (α = 0.05)
AdaBoost.MH	3.3387	0.0008	0.0038	Rejected
SAMME	2.7277	0.0045	0.0055	Rejected
LogitBoost	1.3311	0.1831	0.0008	Not rejected
GradientBoost	2.2258	0.026	0.01	Not rejected
XGBoost	1.1783	0.2386	0.025	Not rejected
MEBoost	2.9022	0.0037	0.0041	Rejected
SMOTEBoost	1.0038	0.3154	0.05	Not rejected
RUSBoost	2.8804	0.0039	0.0045	Rejected
LightGBM	2.6186	0.0088	0.0062	Not rejected
AdaC1	3.3387	0.0008	0.0038	Rejected
AdaC2	2.8368	0.0045	0.005	Rejected
AdaC3	2.4876	0.0128	0.0083	Not rejected
AdaCost	2.1167	0.0342	0.0125	Not rejected

to handle imbalanced data, got the second Friedman rank and could outperform four algorithms.

By comparing a family of binary classification boosting algorithms that OVO strategy was taken to them to handle multi-class problems, in addition to get higher Friedman rank, none of them could outperform other algorithms, except AdaCost that outperforms just AdaC1 in terms of MAUC criterion.

From the cost-sensitive boosting approaches, in overall AdaCost shows better performance; however, AdaC3 presents better average MAUC value than the others. Moreover, AdaC3 works better than all algorithms for 3 datasets.

MMCC as a measure

Table 7 shows the average results of cross-validation for MMCC measurement. The form of the reported results is similar to Table 3. It is clear that, except AdaBoost.MH, SAMME, RUSBoost, and AdaC1, the rest of the algorithms present the best MMCC

Tanha *et al. J Big Data*   (2020) 7:70

Page 25 of 47

Table 5 Ranks computed by the Wilcoxon test based on the MAUCs

Algorithms	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
AdaBoost.MH	–	61	20.5	39.5	64	26	85	36	73	85	74	75	64	55
SAMME	44	–	14	31	57	35	65	23	76	75	86	83	62	57
CatBoost	99.5	106	–	76	95	79	106	79	109	101	107.5	107	95	90
LogitBoost	80.5	89	44	–	81	46	73	64	83	89	102	98	88	83
GradientBoost	56	63	25	39	–	45	64	43	75	62	76	64	73	53
XGBoost	94	85	41	74	75	–	87	58	81	85	105	96	80.05	80
MEBoost	35	55	14	32	56	33	–	32	62	85	72	60	52	51
SMOTEBoost	84	97	41	56	77	62	88	–	102	88	109	111	94	88
RUSBoost	47	44	11	37	45	39	58	18	–	68	70	64	43	47
LightGBM	35	45	19	31	58	35	35	32	52	–	58	43	42	41
AdaC1	46	34	12.5	18	44	15	48	11	50	62	–	43.5	41	25
AdaC2	45	37	13	22	65	24	60	9	56	77	76.5	–	43	40
AdaC3	56	58	25	32	47	39	68	26	77	78	79	77	–	52
AdaCost	65	63	30	37	67	40	69	32	73	79	80	80	68	–

Tanha *et al. J Big Data*   (2020) 7:70

Page 26 of 47

Table 6 Summary of Wilcoxon test based on the MAUCs

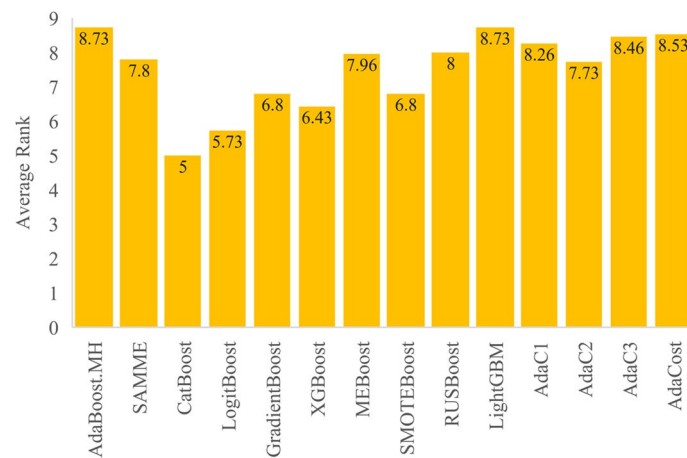| Algorithms	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
AdaBoost.MH	–													
SAMME		–												
CatBoost	●	●	–				●		●	●	●	●	●	
LogitBoost				–							●	●		
GradientBoost			●		–									
XGBoost						–					●	●		
MEBoost							–							
SMOTEBoost		●						–	●		●	●		
RUSBoost									–					
LightGBM										–				
AdaC1											–			
AdaC2												–		
AdaC3													–	
AdaCost											●			–

Fig. 3 Average ranks obtained from Friedman test based on the MMCC

for at least one dataset. However, the *p*-value computed by the Friedman test is 0.0439, which rejects the null hypothesis. Furthermore, according to Fig. 3 CatBoost algorithm achieves the lowest average rank. By selecting the CatBoost algorithm as a control method and conducting the Holm post-hoc test, we can see that it fails to find a significant difference between CatBoost and others, as stated in Table 8. On the contrary, the Wilcoxon rank-sum test shows that CatBoost outperforms ten algorithms as presented in Tables 9 and 10.

From the MMCC viewpoint, according to Table 10, both XGBoost and SMOTEBoost can be considered as the second best algorithm due to outperforming four other algorithms, although, they are significantly better than the cost-sensitive boosting methods and the old AdaBoost.MH algorithm.

Another interesting point is that we can see the superiority of most of the gradient-based boosting algorithms (CatBoost, GradientBoost, XGBoost) and SMOTEBoost algorithm over other methods, therefore, integrating the used sampling method in SMOTEBoost (Synthetic Minority Oversampling Technique) with the gradient-based boosting algorithm will improve their performance in front of imbalanced datasets.

G-mean as a measure

The obtained results of *G*-mean for algorithms are reported in Table 11. It is observable that CatBoost presents the best results in more datasets than the other algorithms. Furthermore, as shown in Fig. 4, CatBoost achieves the lower Friedman average rank. But, unlike the previous measurement criteria, RUSBoost achieves the worst rank. By conducting Holm post-hoc test, we can see the test cannot find a significant difference between the control method (CatBoost) and the other methods, as shown in Table 12. However, the Wilcoxon test shows the difference between each pair of algorithms.

As presented in Tables 13 and 14, we can see that CatBoost, LogitBoost, SMOTEBoost and XGBoost outperforms 5, 3, 3, and 1 algorithms, respectively, in terms of *G*-mean criterion. In addition, like the previous assessments by using MAUC and MMCC measurements, also in G-mean, no binary boosting algorithm outperforms other methods.

Table 7 The obtained results for MMCC from boosting algorithms using decision tree as a base learner

Dataset	Algorithm													
	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
Wine	0.8134 ±0.0254	0.8318 ±0.0236	0.9546 ±0.0086	0.9155 ±0.0188	0.9823 ±0.0050	0.9343 ±0.0237	0.9664 ±0.0155	0.9013 ±0.0325	0.923 ±0.0178	0.9382 ±0.0280	0.865 ±0.0320	0.8388 ±0.0327	0.9176 ±0.0254	0.8696 ±0.0262
Hayes-Roth	0.6109 ±0.0274	0.5911 ±0.0212	0.5229 ±0.0267	0.6026 ±0.0228	0.611 ±0.0222	0.5967 ±0.0302	0.5916 ±0.0374	0.5876 ±0.0300	0.5401 ±0.0554	0.0649 ±0.0682	0.6027 ±0.0255	0.622 ±0.0199	0.6325 ±0.0288	0.6157 ±0.0325
Contraceptive	0.0562 ±0.0098	−0.0104 ±0.0142	0.0615 ±0.0089	−0.0181 ±0.0119	−0.0111 ±0.0199	0.073 ±0.0143	0.0078 ±0.0099	−0.0395 ±0.0239	−0.1068 ±0.0074	0.0018 ±0.0086	0.0894 ±0.0111	−0.3029 ±0.0044	−0.1027 ±0.0237	0.1025 ±0.0185
Pen-based	0.8073 ±0.0106	0.918 ±0.0108	0.9252 ±0.0087	0.9125 ±0.0105	0.9049 ±0.0056	0.8684 ±0.0116	0.946 ±0.0073	0.9026 ±0.0053	0.92 ±0.0091	0.9076 ±0.0035	0.8119 ±0.0183	0.8121 ±0.0109	0.8105 ±0.0113	0.8111 ±0.0151
Vertebral column	0.5894 ±0.0386	0.5914 ±0.0412	0.6373 ±0.0449	0.6061 ±0.0292	0.6072 ±0.0251	0.604 ±0.0282	0.604 ±0.0385	0.6468 ±0.0412	0.6148 ±0.0361	0.6348 ±0.0464	0.6167 ±0.0156	0.6249 ±0.0251	0.6048 ±0.0194	0.6095 ±0.0204
New thyroid	0.7741 ±0.0335	0.7586 ±0.04565	0.8003 ±0.0190	0.8215 ±0.0339	0.8317 ±0.0222	0.792 ±0.0421	0.2416 ±0.0385	0.8057 ±0.0499	0.7983 ±0.0311	0.823 ±0.0205	0.7842 ±0.0521	0.7431 ±0.0690	0.7684 ±0.0707	0.7587 ±0.580
Dermatology	0.9089 ±0.0150	0.9301 ±0.0091	0.9559 ±0.0068	0.9344 ±0.0170	0.9434 ±0.0100	0.9386 ±0.0233	0.8412 ±0.0321	0.9317 ±0.0119	0.8889 ±0.0204	0.8172 ±0.0176	0.9115 ±0.0141	0.9171 ±0.0135	0.9024 ±0.0206	0.9063 ±0.0173
Balance scale	0.2229 ±0.0239	0.256 ±0.0352	0.3643 ±0.0706	0.2154 ±0.0217	0.2083 ±0.0293	0.2247 ±0.0178	0.2516 ±0.0180	0.2116 ±0.0189	0.3379 ±0.0492	0.429 ±0.0364	−0.2735 ±0.0178	0.1405 ±0.0110	0.1339 ±0.0320	0.3307 ±0.0214
Glass	0.2494 ±0.0496	0.301 ±0.0596	0.3623 ±0.0439	0.4125 ±0.0512	0.2796 ±0.0915	0.3909 ±0.0489	0.2321 ±0.0805	0.3011 ±0.0389	0.0595 ±0.0894	−0.1694 ±0.0081	0.2539 ±0.0624	0.2713 ±0.0398	0.1628 ±0.0666	0.2529 ±0.0314
Heart	−0.4132 ±0.0261	−0.3978 ±0.0269	−0.3753 ±0.0278	−0.444 ±0.0327	−0.4106 ±0.0376	−0.4353 ±0.0393	−0.4389 ±0.02901	−0.3518 ±0.0429	−0.3794 ±0.0357	−0.3506 ±0.08237	−0.4077 ±0.0301	−0.3952 ±0.0476	−0.3878 ±0.0302	−0.3841 ±0.0324
Car evaluation	0.9731 ±0.0166	0.8248 ±0.0172	0.9697 ±0.0085	0.9921 ±0.0070	0.9701 ±0.0139	0.6457 ±0.0230	0.9843 ±0.0072	0.8061 ±0.0208	0.5592 ±0.0273	0.9767 ±0.0057	0.3903 ±0.0382	0.6436 ±0.0084	0.7439 ±0.0122	0.8122 ±0.0160
Thyroid	0.9494 ±0.0044	0.943 ±0.0049	0.9569 ±0.0065	0.9471 ±0.0062	0.9431 ±0.0065	0.9537 ±0.0023	0.9358 ±0.01143	0.9603 ±0.0038	0.8774 ±0.0306	0.7566 ±0.0351	0.3333 ±0.0000	0.9604 ±0.0049	0.9561 ±0.0029	0.3335 ±0.0004

Tanha *et al. J Big Data*   (2020) 7:70

Page 29 of 47

Table 7 (continued)

Dataset	Algorithm													
	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
Yeast	−0.0095	−0.1297	−0.0283	−0.0228	−0.0796	*0.0415*	−0.0822	−0.0437	−0.2991	−0.1124	−0.2593	−0.0542	−0.0061	−0.0483
	±0.0361	±0.0132	±0.0192	±0.0244	±0.0266	±0.0195	±0.0148	±0.0194	±0.0269	±0.0098	±0.0293	±0.0224	±0.0333	±0.0361
Page blocks	0.5489	0.4927	*0.6272*	0.5448	−0.3273	0.572	0.5375	0.5423	0.4592	0.1063	0.4042	0.4137	0.5912	0.3993
	±0.0244	±0.0325	±0.0170	±0.0239	±0.0693	±0.0341	±0.0273	±0.0325	±0.0337	±0.0125	±0.0112	±0.0192	±0.0129	±0.0235
Shuttle	0.8751	0.9385	0.9176	0.9266	−0.2663	0.9156	0.6635	*0.9682*	0.7336	−0.1741	0.8661	0.873	0.8784	0.8767
	±0.0302	±0.0190	±0.0170	±0.0244	±0.0498	±0.0262	±0.0248	±0.0139	±0.0256	±0.0282	±0.0226	±0.0250	±0.0154	±0.0283
Average	0.5304	0.5226	*0.5768*	0.5564	0.4124	0.5411	0.4855	0.5420	0.4618	0.3766	0.3992	0.4739	0.5071	0.4831

The best performance is shown in italic for each dataset

Table 8 Holm post-hoc test results based on the MMCCs

Control method (CatBoost)				
Algorithm	Z	*p*-value	Holm	Hypothesis (α = 0.05)
AdaBoost.MH	2.444	0.0145	0.0038	Not rejected
SAMME	1.833	0.0667	0.0083	Not rejected
LogitBoost	0.48	0.6311	0.05	Not rejected
GradientBoost	1.1783	0.2386	0.0125	Not rejected
XGBoost	0.9383	0.348	0.025	Not rejected
MEBoost	1.9421	0.0521	0.0071	Not rejected
SMOTEBoost	1.1783	0.2386	0.0166	Not rejected
RUSBoost	1.9639	0.0495	0.0062	Not rejected
LightGBM	2.444	0.0145	0.0041	Not rejected
AdaC1	2.1385	0.0324	0.0055	Not rejected
AdaC2	1.7893	0.0735	0.01	Not rejected
AdaC3	2.2694	0.0232	0.005	Not rejected
AdaCost	2.3131	0.0207	0.0045	Not rejected

Computational time study

Computation time analysis was performed to determine which boosting algorithm shows lower computation time, and the results are presented in Table 15. It should be noted that the times are recorded just for one fold of cross-validation in the training phase.

From Table 15, it is obvious that the LightGBM algorithm presents the lowest average computational time. This is an interesting observation, because LightGBM is a binary boosting algorithm and by using the OVO decomposition mechanism for handling multi-class problems, it should be generated boosting process for $K(K-1)/2$ pairs of classes (K is the number of classes). But, by comparing the computational times of algorithms for each dataset, one can see that XGBoost is better than LightGBM in most of the datasets. However, considering the classification performance of the algorithms, we can see CatBoost shows fair computational times.

SMOTEBoost shows the highest computational time due to generating synthetic samples for minority classes. Furthermore, although it presents the best performance for the largest dataset (Shuttle), it needs a lot of training time.

Metric evaluation study

In this section, we aim to investigate which metric would turn out to be the most suitable measure for evaluating the predictable performance of the algorithm on multi-class imbalanced problems by comparing MAUC, MMCC and *G*-mean metrics by using the comparison framework on the 15 imbalanced conventional datasets.

By obtaining the results of the boosting algorithms on these datasets with 5-Fold cross-validation, 75 test sets (15 × 5) are obtained for each pairwise comparison between evaluation metrics (MAUC vs. MMCC, MAUC vs. *G*-mean and MMCC vs. *G*-mean). In this test, all the possible pairs of 14 boosting algorithms (91 pairs) are considered. First of all, regardless of which metric is the best, the degree of

Table 9 Ranks computed by the Wilcoxon test based on the MMCCs

Algorithms	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
AdaBoost.MH	–	63	16	36	62	25	82	45	84	82	76	75	61	59
SAMME	57	–	12	30	55	43	62	29	90	74	93	87	70	64
CatBoost	104	108	–	82	95	84	105	89	116	99	110	110	107	104
LogitBoost	84	90	38	–	73	58	84	79	96	85	104	106	89	89
GradientBoost	58	65	25	47	–	50	62	50	75	66	79	71	74	59
XGBoost	95	77	36	62	70	–	75	61	95	83	106	104	85	83
MEBoost	38	58	15	36	58	30	–	43	75	83	75	65	59	57
SMOTEBoost	75	91	31	41	70	59	77	–	104	82	109	113	95	89
RUSBoost	36	30	4	24	45	25	45	16	–	68	68	54	39	48
LightGBM	38	46	21	35	54	37	37	38	52	–	63	49	44	50
AdaC1	44	27	10	16	41	14	45	11	52	57	–	32	31	35
AdaC2	45	33	10	14	49	16	55	7	66	71	88	–	35	50
AdaC3	59	50	13	31	46	35	61	25	81	76	89	85	–	59
AdaCost	61	56	16	31	61	37	63	31	72	70	85	70	61	–

Tanha *et al. J Big Data* (2020) 7:70

Page 31 of 47

Tanha *et al. J Big Data*   (2020) 7:70

Page 32 of 47

Table 10 Summary of Wilcoxon test based on the MMCCs

Algorithms	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
AdaBoost.MH	—													
SAMME		—												
CatBoost	●	●	—		●		●		●	●	●	●	●	●
LogitBoost				—					●					
GradientBoost					—									
XGBoost	●					—			●		●	●		
MEBoost							—				●	●		
SMOTEBoost								—	●		●	●	●	
RUSBoost									—					
LightGBM										—				
AdaC1											—			
AdaC2												—		
AdaC3													—	
AdaCost														—

Tanha *et al. J Big Data*   (2020) 7:70

Page 33 of 47

Table 11 The obtained results for *G-mean* from boosting algorithms using decision tree as a base learner

Dataset	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
Wine	0.9058 ±0.0170	0.9149 ±0.0125	0.9768 ±0.0058	0.9565 ±0.0102	0.9908 ±0.0026	0.9665 ±0.1160	0.9832 ±0.0077	0.9501 ±0.0165	0.962 ±0.0091	0.9688 ±0.0138	0.9325 ±0.0173	0.9193 ±0.0171	0.8776 ±0.0133	0.9347 ±0.0132
Hayes-Roth	0.7948 ±0.0179	0.7829 ±0.0116	0.7558 ±0.0129	0.7901 ±0.0110	0.7915 ±0.0162	0.786 ±0.0156	0.7896 ±0.0194	0.7823 ±0.0164	0.7793 ±0.0225	0.3203 ±0.1187	0.7896 ±0.0127	0.8003 ±0.0129	0.8059 ±0.0225	0.7982 ±0.0204
Contraceptive	0.5243 ±0.0048	0.4841 ±0.0079	0.5052 ±0.0055	0.4819 ±0.0085	0.4849 ±0.0101	0.5335 ±0.0080	0.4971 ±0.0063	0.4704 ±0.0126	0.4327 ±0.0081	0.4926 ±0.0064	0.5314 ±0.0051	0.1711 ±0.0240	0.3828 ±0.0129	0.539 ±0.0088
Pen-based	0.9006 ±0.0057	0.9576 ±0.0056	0.9615 ±0.0046	0.9552 ±0.0051	0.9509 ±0.0031	0.932 ±0.0055	0.9724 ±0.0038	0.9496 ±0.0027	0.9586 ±0.0048	0.9524 ±0.0020	0.9031 ±0.0101	0.9035 ±0.0053	0.9023 ±0.0063	0.9027 ±0.0081
Vertebral column	0.7678 ±0.0217	0.7669 ±0.0239	0.7921 ±0.0245	0.7696 ±0.0163	0.7704 ±0.0193	0.7733 ±0.0159	0.7713 ±0.0191	0.7907 ±0.0239	0.7765 ±0.0215	0.7835 ±0.0252	0.7815 ±0.0113	0.7801 ±0.0166	0.7674 ±0.0146	0.7318 ±0.0184
New thyroid	0.8971 ±0.0176	0.8904 ±0.02311	0.9099 ±0.0106	0.9131 ±0.0140	0.9244 ±0.0098	0.9065 ±0.0176	0.2727 ±0.01659	0.9055 ±0.0240	0.9131 ±0.01361	0.9156 ±0.0091	0.8979 ±0.0268	0.8869 ±0.0327	0.8389 ±0.0322	0.8852 ±0.0238
Dermatology	0.9555 ±0.0067	0.9625 ±0.0040	0.9786 ±0.0039	0.9665 ±0.0096	0.9713 ±0.0050	0.9693 ±0.0125	0.8723 ±0.0854	0.9653 ±0.0061	0.9433 ±0.0109	0.9000 ±0.0097	0.9566 ±0.0070	0.9602 ±0.0065	0.9522 ±0.0103	0.9546 ±0.0075
Balance scale	0.2461 ±0.0632	0.4344 ±0.0731	0.2898 ±0.1676	0.2381 ±0.0515	0.0378 ±0.0650	0.0098 ±0.0310	0.2856 ±0.0730	0.4161 ±0.0544	0.4877 ±0.1068	0.6372 ±0.0641	0.073 ±0.0521	0.5198 ±0.0074	0.4031 ±0.0447	0.5881 ±0.0196
Glass	0.3097 ±0.1445	0.4095 ±0.0571	0.4591 ±0.1209	0.5517 ±0.1101	0.3131 ±0.1880	0.4252 ±0.1218	0.2413 ±0.0957	0.2414 ±0.1493	0.2014 ±0.1506	0.0000 ±0.0000	0.3252 ±0.1040	0.3382 ±0.1197	0.2782 ±0.1381	0.3474 ±0.1161
Heart	0.0000 ±0.0000	0.0000 ±0.0000	0.0119 ±0.0250	0.0000 ±0.0000	0.0000 ±0.0000	0.0000 ±0.0000	0.0000 ±0.0000	0.0993 ±0.0801	0.0058 ±0.0185	0.0000 ±0.0000	0.0173 ±0.0282	0.0000 ±0.0000	0.0307 ±0.0324	0.0066 ±0.0210
Car evaluation	0.9878 ±0.0091	0.9474 ±0.0065	0.9861 ±0.0045	0.9967 ±0.0036	0.9852 ±0.0081	0.8401 ±0.0154	0.9920 ±0.0042	0.9304 ±0.0072	0.8219 ±0.0202	0.9882 ±0.0036	0.603 ±0.0377	0.8337 ±0.0060	0.8852 ±0.0089	0.8686 ±0.0103
Thyroid	0.9771 ±0.0031	0.9779 ±0.0017	0.9809 ±0.0036	0.9767 ±0.0014	0.974 ±0.0025	0.9789 ±0.0014	0.9688 ±0.0070	0.9826 ±0.0020	0.9517 ±0.0121	0.8551 ±0.0052	0.0000 ±0.0000	0.9827 ±0.0031	0.9796 ±0.0089	0.0041 ±0.0129

Table 11 (continued)

Dataset	Algorithm													
	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
Yeast	0.0236	0.0000	0.0000	0.0193	0.0000	0.0000	0.0000	0.0513	0.0000	0.0000	0.0000	*0.1649*	0.0937	0.155
	±0.0499	±0.0000	±0.0000	±0.0408	±0.0000	±0.0000	±0.0000	±0.0543	±0.0000	±0.0000	±0.0000	±0.0957	±0.0929	±0.0766
Page blocks	0.8275	0.8142	*0.8593*	0.8302	0.1030	0.8411	0.8290	0.8180	0.6992	0.0473	0.0000	0.3131	0.7762	0.0000
	±0.0119	±0.0131	±0.0080	±0.0093	±0.0851	±0.0173	±0.0118	±0.0163	±0.0351	±0.0035	±0.0000	±0.0283	±0.0095	±0.0000
Shuttle	0.9195	0.9422	0.8999	0.9707	0.0028	0.9446	0.3690	*0.9881*	0.8915	0.0000	0.9400	0.9039	0.9205	0.9432
	±0.0766	±0.0826	±0.0931	±0.0165	±0.0064	±0.0805	±0.0521	±0.0088	±0.0080	±0.0000	±0.0577	±0.0871	±0.0726	±0.0591
Average	0.6691	0.6857	0.6911	*0.6944*	0.5533	0.6605	0.5896	0.6894	0.6550	0.5241	0.5167	0.6318	0.6596	0.5773

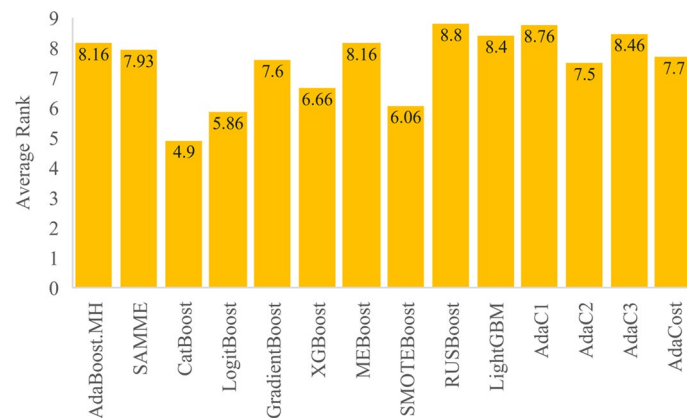The best performance is shown in italic for each dataset

Fig. 4 Average ranks obtained from Friedman test based on the *G*-mean

Table 12 Holm post-hoc test results based on the *G*-means

Control method (CatBoost)

Algorithm	Z	*p*-value	Holm	Hypothesis (α = 0.05)
AdaBoost.MH	2.1385	0.0324	0.0062	Not rejected
SAMME	1.9857	0.047	0.0071	Not rejected
LogitBoost	0.6328	0.5268	0.05	Not rejected
GradientBoost	1.7675	0.0771	0.01	Not rejected
XGBoost	1.1565	0.2474	0.0166	Not rejected
MEBoost	2.1385	0.0324	0.0055	Not rejected
SMOTEBoost	0.7637	0.445	0.025	Not rejected
RUSBoost	2.5531	0.0106	0.0038	Not rejected
LightGBM	2.2912	0.0219	0.005	Not rejected
AdaC1	2.5313	0.0113	0.0041	Not rejected
AdaC2	1.7021	0.0887	0.0125	Not rejected
AdaC3	2.3349	0.0195	0.0045	Not rejected
AdaCost	1.833	0.0667	0.0083	Not rejected

consistency of evaluation metrics are examined. The results of the DoC comparison between metrics are reported in Table 2. The *C* value shown in the table for each pair of metrics is resulted by averaging the C values of 91 pairs of algorithms. As discussed in "Comparison of evaluation metrics" section, the value of *C* larger than 0.5 indicates the consistency of the two metrics. According to Table 16, The *C* value obtained for all the metric pairs is larger than 0.9, which satisfies the condition of being larger than 0.5. Thus, it can be claimed that MAUC is consistent with MMCC with a consistency degree of 0.9853. Also, MAUC is consistent with *G*-mean with a consistency degree of 0.9334 and finally MMCC and *G*-mean are constant with each other with the consistency degree of 0.9403. But among these three pairwise comparisons, MAUC and MMCC are more consistent with each other.

In order to select the best metric, the degree of discriminancy (DoD) of each metric over another metric is examined. The results of the DoD comparison between metrics are reported in Table 17. Exceptionally, in Table 17, instead of average *D* values, the ratio

Table 13 Ranks computed by the Wilcoxon test based on the *G*-means

Algorithms	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
AdaBoost.MH	–	48	31	26	68	38	70	41	78	74	68	51	70	55
SAMME	57	–	32	20	66.5	40.5	71.5	44	72	72.5	72	65	93	71.5
CatBoost	89	73	–	67	82	75	85	63	83	92	86	85	86	78
LogitBoost	79	85	53	–	75	75	70	60	82	79	103	76	85	84
GradientBoost	41	53	23	30	–	57.5	52.5	47	60	81.5	67	46	59	66
XGBoost	67	79	30	30	62.5	–	74.5	45.5	75	77.5	82	72	73	79
MEBoost	35	48	20	35	67.5	45.5	–	36	54	80.5	64.5	40	47	52
SMOTEBoost	79	76	57	60	73	74.5	84	–	100	89	99	80	101	72
RUSBoost	42	33	22	23	45	30	51	20	–	69	69	59	54	53
LightGBM	31	42.5	13	26	38.5	42.5	39.5	31	36	–	56	35	43	55
AdaC1	52	33	19	17	38	23	55.5	21	36	49	–	44	48	31
AdaC2	54	40	35	29	59	33	65	40	61	70	76	–	65	56
AdaC3	50	27	34	35	61	47	73	19	66	77	72	55	–	49
AdaCost	65	48	42	36	54	41	68	48	67	65	74	64	71	–

Table 14 Summary of Wilcoxon test based on the G-means

Algorithms	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
AdaBoost.MH	—													
SAMME		—												
CatBoost			—		●		●		●	●	●			
LogitBoost		●		—					●		●			
GradientBoost					—									
XGBoost						—					●			
MEBoost							—							
SMOTEBoost								—	●		●		●	
RUSBoost									—					
LightGBM										—				
AdaC1											—			
AdaC2												—		
AdaC3													—	
AdaCost														—

Tanha *et al. J Big Data*   (2020) 7:70

Page 38 of 47

Table 15 Computational time (seconds) of the boosting algorithms

Dataset	Algorithm													
	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
Wine	0.01	0.01	0.79	0.85	0.35	0.03	1.37	0.02	0.72	0.06	0.02	0.01	0.01	0.01
Hayes-Roth	0.01	0.34	0.17	0.81	0.66	0.02	1.45	1.07	0.50	0.06	0.01	0.02	0.01	0.01
Contraceptive	1.11	0.57	0.39	4.53	1.81	0.08	7.41	16.11	1.17	0.16	0.45	0.57	0.57	0.71
Pen-based	0.27	1.04	2.65	7.19	3.62	0.20	27.90	1.26	1.74	1.40	0.32	0.36	0.32	0.41
Vertebral column	0.02	0.54	0.65	1.21	0.54	0.03	3.08	5.45	0.72	0.13	0.02	0.02	0.01	0.02
New thyroid	0.01	0.01	0.48	0.87	0.38	0.02	0.56	4.91	0.52	0.05	0.01	0.01	0.02	0.07
Dermatology	0.08	0.71	0.47	2.63	1.02	0.07	2.42	7.20	1.18	0.51	0.05	0.06	0.33	0.08
Balance scale	0.75	0.40	0.63	1.13	1.22	0.04	5.59	14.67	0.89	0.11	0.62	0.55	0.46	0.55
Glass	0.05	0.36	0.71	1.65	0.87	0.05	2.95	3.46	0.65	0.11	0.05	0.06	0.06	0.06
Heart	0.17	0.44	0.44	1.74	1.00	0.53	14.87	9.10	0.69	0.09	0.17	0.30	1.20	0.13
Car evaluation	0.41	0.70	0.45	1.91	1.85	0.08	5.93	89.18	0.99	0.13	0.07	0.11	0.80	0.08
Thyroid	9.63	2.69	2.48	6.08	2.18	0.35	1.86	Hours	1.44	0.48	1.35	2.58	0.93	2.91
Yeast	1.26	0.79	1.22	6.35	4.79	0.32	24.84	20.33	0.91	0.90	0.27	0.23	0.32	0.41
Page blocks	2.89	3.94	2.75	12.19	4.05	0.40	14.66	116.07	1.22	0.31	1.57	1.63	1.66	8.49
Shuttle	0.14	9.40	8.64	58.01	74.24	3.31	331.38	Hours	10.02	0.15	0.06	0.06	0.06	0.06
Average	1.12	1.46	1.53	7.14	6.57	0.37	29.75	Hours	1.56	0.31	0.33	0.44	0.45	0.93

The best comptional time is shown in italic for each dataset

Table 16 Experimental results for verifying degree of consistency (C) of evaluation metrics

	MAUC	MMCC	G-mean
MAUC	–	0.9583	0.9334
MMCC	–	–	0.9403
G-mean	–	–	–

Table 17 Experimental results for verifying degree of discriminancy (D) of evaluation metrics

	MAUC (%)	MMCC (%)	G-mean (%)
MAUC	–	12	100
MMCC	7	–	100
G-mean	0	0	–

Table 18 Experimental results for verifying degree of indifferency (E) o evaluation metrics

	MAUC	MMCC	G-mean
MAUC	–	0.0872	0.0853
MMCC	–	–	0.0829
G-mean	–	–	–

of the cases with $D > 1$ are reported. The reason for taking this approach is the existence of NaN and Infinite values, hence the averaging would not indicate the real results. According to the table, in the comparison between MAUC and *G*-mean, in all the cases (100% of cases), the value of D for MAUC over *G*-mean is larger than 1. So, MAUC always acts better than *G*-mean and in none of the 91 pairs of algorithms, *G*-mean wins (0%). Also, in the comparison between MMCC and *G*-mean, in all the cases (100% of the cases), the value of D for MMCC over *G*-mean is larger than 1 and for *G*-mean over MMCC, $D > 1$ is not observed in any of the 91 pairs. Until now, the results prove that both MAUC and MMCC are more discriminant than *G*-mean. The final step in selecting the best metric is comparing the D values between MAUC and MMC metrics. According to Table 3, in only 12% of the cases (11 out of 91), the DoD of MAUC over MMCC is larger than 1. But in 70% of the cases (64 out of 91), the DoD of MMCC over MAUC is larger than 1. Thus, it can be claimed that MMCC is more discriminant than MAUC and can better distinguish between classification algorithms. The conclusion of the DoC test between evaluation metrics is that MMCC and MAUC are totally better than *G*-mean and MMCC act slightly better than MAUC.

Finally, the degree of indifferency (DoI) of evaluation metrics is tested. The results of DoI comparison are reported in Table 18 by averaging the results of 91 pairs of algorithms. According to this table, by comparing MAUC and MMCC, in nearly 8% of the cases, none of the metrics can distinguish between the classification algorithms. Furthermore, by comparing MAUC and *G*-mean, the average DoC is equal to 0.0853, so, in 8% of the cases, both MAUC and *G*-mean cannot tell the difference between classification algorithms. This result is nearly true for the comparison between MMCC and

G-mean. Also, in this case, in 8% of the comparisons, they both cannot distinguish the difference between classification algorithms. In all the pairwise comparisons of DoI, the value of *E* is approximately 8%.

Experiments on big datasets

According to study [16], a dataset consisting at least 100,000 instances can be considered as a big dataset. Therefore, 4 big multi-class imbalanced datasets, as shown in Table 2, were used to investigate the performance of the boosting algorithms [17, 18]. The obtained results for the used three performance metrics on these datasets are presented in Table 19.

Based on the previous experiments that were determined MMCC is the best evaluation metric, we performed Friedman test on the obtained results of this metric for ranking the algorithms. Figure 5 shows the outcome of this statistical test. It is obvious that the LogitBoost algorithm performs better than the other algorithms on big datasets in overall. However, in particular big datasets with high imbalanced ratio (FARS and Poker), the LogitBoost could not present good results. For instance, according to Table 19, AdaC3 performs well in FARS dataset based on the MMCC metric. Moreover, all of the boosting methods perform worse than random prediction in Poker dataset and most of them present very low performance on FARS dataset. By considering the obtained values for *G*-mean metric, we can conclude that all of the algorithms could not learn one or some of the classes.

Result discussion

If we rank the algorithms based on all performance metrics, then CatBoost is the first due to outperforming more algorithms than the others. The second best algorithm is SMOTEBoost that outperforms 4, 4, and 3 other boosting methods in terms of MAUC, MMCC, and G-mean, respectively. This observation indicates that oversampling is an effective method in imbalanced data classification, while we can see RUSBoost, which uses under-sampling method, could not outperform any other algorithm.

Considering all evaluation metrics, XGBoost and LogitBoost outperform 7 and 6 algorithms, respectively. Thus, XGBoost can be considered as the third best algorithm. It should be noted that it is possible to get better results from XGBoost by finding the optimal values of its parameters, but it is not an easy task and having many parameters increases the complexity of this issue.

One of the interesting observations is that no binary boosting algorithm with decomposition mechanism is not better than the others, except AdaCost which is better than AdaC1 in terms of MAUC. Thus, multi-class boosting algorithms can deal with imbalanced data problem in a more satisfactory way due to using a proper margin loss function for multi-class problems. However, from Tables 3, 7 and 11, it can be seen that there are some datasets in which binary boosting methods show the best results.

In addition, according to the obtained ranks from the Freidman test, the algorithm with lower rank (CatBoost) was selected as a control algorithm for post-hoc test, but it failed to reject the null hypothesis, especially in the results for MMCC and G-means metrics. Therefore, there are some cases that the Holm post-hoc test fails
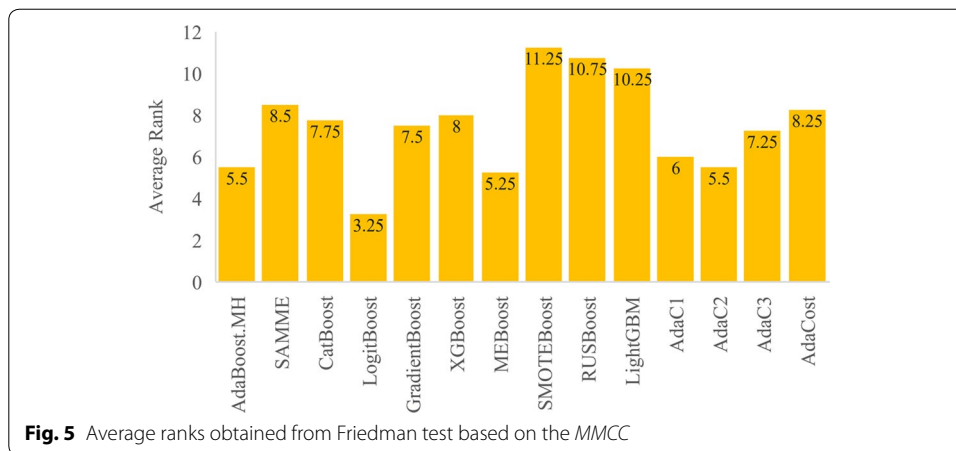
Table 19 The obtained results for evaluation metrics from running boosting algorithms on multi-class imbalanced big datasets

Dataset	Algorithm	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
FARS															
	MAUC	0.5382 ±0.0022	0.4754 ±0.0129	0.5330 ±0.0030	0.5428 ±0.0039	0.2112 ±0.2228	0.5108 ±0.0029	0.5543 ±0.0029	0.4546 ±0.0153	0.3708 ±0.0195	0.4755 ±0.0089	0.5970 ±0.0117	0.3327 0.0138±	0.5190 ±0.0093	0.4306 ±0.130
	MMCC	0.1803 ±0.0027	0.0907 ±0.0134	0.1772 ±0.0043	0.1962 ±0.0052	−0.3315 ±0.0401	0.1384 ±0.0037	0.2078 ±0.0034	0.0580 ±0.0207	−0.1258 ±0.0202	0.1084 ±0.0166	0.2107 ±0.0070	0.4295 0.0261±	0.3771 ±0.0032	−0.1482 ±0.0243
	G-mean	0.0000 ±0.0000	0.0000 ±0.0000	0.0000 ±0.0000	0.0000 ±0.0000	0.0031 ±0.0072	0.0000 ±0.0000	0.0000 ±0.0000	0.0000 ±0.0000	0.0368 ±0.0422	0.0000 ±0.0000	0.1013 ±0.0700	0.0000 ±0.0000	0.1215 ±0.0579	0.0608 ±0.0159
KDD Cup'99															
	MAUC	0.9251 ±0.0158	0.9267 ±0.0109	0.9333 ±0.0155	0.9637 ±0.0112	0.3057 ±0.0218	0.8755 ±0.0078	0.9468 ±0.0036	0.9485 ±0.0172	0.7874 ±0.0294	0.4601 ±0.0274	0.8380 ±0.0169	0.8553 ±0.0250	0.9440 ±0.0042	0.6618 ±0.0468
	MMCC	0.8523 ±0.0000	0.8320 ±0.0124	0.8106 ±0.0180	0.9119 ±0.0161	0.0236 ±0.0312	0.7120 ±0.0109	0.8873 ±0.0066	0.8122 ±0.0268	0.3392 ±0.0423	0.0171 ±0.0496	0.3678 ±0.0186	0.4477 ±0.0309	0.6201 ±0.0090	0.2050 ±0.0428
	G-mean	0.9398 ±0.0156	0.9411 ±0.0055	0.0000 ±0.0000	0.9728 ±0.0101	0.0649 ±0.0179	0.0000 ±0.0000	0.9535 ±0.0058	0.8696 ±0.0292	0.5072 ±0.0375	0.0000 ±0.0000	0.2637 ±0.0294	0.3621 ±0.0431	0.6894 ±0.0124	0.0564 ±0.0222
Covtype															
	MAUC	0.6533 ±0.0000	0.4085 ±0.0033	0.6624 ±0.0026	0.8367 ±0.0022	0.7003 ±0.9982	0.5732 ±0.0008	0.7722 ±0.00002	0.4737 ±0.0101	0.4002 ±0.0172	0.7886 ±0.0120	0.7416 ±0.0014	0.4301 ±0.0000	0.6978 ±0.0006	0.6379 ±0.0020
	MMCC	0.2046 ±0.0012	−0.1357 ±0.0052	0.2276 ±0.00004	0.5189 ±0.0033	0.2874 ±0.0117	0.0791 ±0.0012	0.3963 ±0.0032	0.0757 ±0.0159	−0.1335 ±0.0247	0.4593 ±0.0151	0.3395 ±0.0023	−0.1344 ±0.0035	0.2995 ±0.0010	0.2146 ±0.0030
	G-mean	0.6839 ±0.0018	0.3885 ±0.0156	0.6967 ±0.0033	0.8594 ±0.0024	0.7003 ±0.0082	0.6038 ±0.0020	00.796 ±0.0013	0.7237 ±0.0089	0.3313 ±0.0132	0.8138 ±0.0154	0.5551 ±0.0027	0.0000 ±0.0000	0.5033 ±0.0009	0.0666 ±0.0544
Poker															
	MAUC	0.0524 ±0.0014	0.1557 ±0.0073	0.2376 ±0.0042	0.1765 ±0.0005	0.1168 ±0.0116	0.1350 ±0.0023	0.2585 ±0.0201	0.1659 ±0.0018	0.1117 ±0.0045	0.2593 ±0.0157	0.2217 ±0.0041	0.09357 ±0.0136	0.2087 ±0.0162	0.06224 ±0.0216

Table 19 (continued)

Dataset	Algorithm													
	AdaBoost.MH	SAMME	CatBoost	LogitBoost	GradientBoost	XGBoost	MEBoost	SMOTEBoost	RUSBoost	LightGBM	AdaC1	AdaC2	AdaC3	AdaCost
MMCC	− 0.4545	− 0.3696	− 0.3011	− 0.3889	− 0.3556	− 0.5599	0.2879−	− 0.2821	− 0.3352	0.275−0	− 0.3790	− 0.8569	− 0.2284	− 0.4425
	± 0.0035	± 0.0140	± 0.0038	± 0.0126	± 0.0083	± 0.0082	± 0.0162	± 0.0076	± 0.0086	± 0.0241	± 0.0089	± 0.0272	± 0.0098	± 0.0155
G-mean	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0018	0.0000
	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0000	± 0.0040	± 0.0000
Average over MMCC	0.1957	0.1044	0.2286	0.3095	− 0.0940	0.0924	0.3009	0.1660	− 0.0638	0.0775	0.1348	− 0.0285	0.2671	− 0.0428

The best performance is shown in italic for each dataset

Tanha *et al. J Big Data*   (2020) 7:70

Page 43 of 47

Fig. 5 Average ranks obtained from Friedman test based on the *MMCC*

in detecting differences between algorithms and it is necessary to use pairwise statistical tests.

The average computational times of algorithms on all datasets indicate that the LightGBM which belongs to the binary boosting set is faster than the other algorithms, although it should be run for each pair of classes. The reason for this observation is that binary methods show the lower computational time in large datasets. However, among the multi-class boosting methods, XGBoost has the lowest computation time, which is due to designing for speed and performance. Eventually, SMOTEBoost has the highest computation time where in large datasets, it takes lots of hours. This is due to generating extra samples for minority classes.

Finally, by comparing the metrics through a comparison framework, it can be seen that MMCC is the most suitable metric among the used evaluation metrics in the classification task with multi-class imbalanced datasets. The conducted test shows that although all 3 metrics are consistent with each other and have the same degree of indifference, MAUC is more discriminant than G-mean and MMCC is more discriminant than both MAUC and G-mean. The better discriminacy of MMCC can be explained by its computational formula. MMCC contributes all elements of confusion matrix in its formula, but G-mean is calculated by using just the recall values of classes in its formula and MAUC is computed by the area of the ROC curve, which plots the true-positive rate against false-positive rate of a classifier.

The obtained results for big datasets are different. In overall, although the LogitBoost algorithm performs well, but no algorithm can present better performance in highly imbalanced big datasets. By presenting zero or very small values for G-mean metric, it can be observed that these boosting methods cannot learn some classes in this kind of datasets. Therefore, the boosting algorithms studied in this work are not suitable for multi-class imbalanced big datasets with higher imbalanced ratio.

Conclusion

In this paper, the most significant boosting ensemble algorithms to deal with multi-class imbalanced data have been reviewed. Many real-life problems are naturally multi-class imbalanced and in recent years, this issue has been handled by using ensemble learning

techniques. However, there was a lack of study in the literature to investigate and compare the performance of boosting algorithms as a class of ensemble learning methods on this type of datasets.

According to the analysis, 14 binary and multi-class boosting algorithms investigated on 19 multi-class conventional and big datasets with various imbalanced ratios. By comparing multiple algorithms on multiple datasets, it is observed that CatBoost algorithm outperforms most of the other boosting methods in terms of MAUC, G-mean, and MMCC on 15 conventional datasets. SMOTEBoost can be considered as a second best algorithm that outperforms 4, 4, and 3 algorithms in terms of MAUC, MMCC and G-mean metrics, respectively in conventional datasets. Therefore, it can be concluded that oversampling is an effective approach for tackling the imbalanced data problem. Both XGBoost and LogitBoost algorithms can be expressed as a third algorithm that could perform other algorithms. One of the interesting observations was that just multi-class algorithms could outperform others in which the training process took place without using class decomposition.

The computational time was performed as a part of this study to show which algorithm is faster. The results indicate that binary algorithms present low computational time although they should run for each pair of classes.

The used evaluation metrics were compared together based on the results of algorithms in conventional datasets and it was observed that MMCC is the most suitable metric in the multi-class imbalanced data domain. However, *G*-means metric gives this useful information that the learner could not classify the samples of one or more of the classes if it gets the zero value.

Finally, the results for big datasets were different. It was observed that LogitBoost algorithm performs better than the other algorithms in overall. But generally, boosting methods cannot present good prediction performance in multi-class big datasets with high imbalanced ratio. Therefore, one open research issue is how to modify the studied boosting methods so that they can perform well on highly imbalanced big datasets.

The other potential future works of this study involve the following: (1) An investigation of the effect of the preprocessing operators on the performance of the boosting algorithms. In this study, no preprocessing operator such as normalization and standardization was applied to the data, but it is most likely to obtain better results with experiences by using this preprocessing operation. (2) The study of parameter values and determining under which values a boosting algorithm, XGBoost for example, could improve the performance of multi-class imbalanced data. (3) In this study, decision tree was employed as a base learner under the boosting procedure, but any other learning model can be used. Hence, the performance of boosting approaches with other base learners can be investigated on multi-class imbalanced datasets. (4) Comparing the performance of binary boosting algorithms in multi-class data using other decomposition ways such as one-versus-all and all-versus-all to decompose the multi-class problem into multiple binary decision problems. (5) Comparing the other evaluation metrics which are suitable for the multi-class imbalanced domain, like Kappa, Weighted Accuracy, and other measures.

References
1. Japkowicz N. Learning from imbalanced data sets: a comparison of various strategies. In: AAAI workshop on learning from imbalanced data sets, Vol. 68. 2000. p. 10–5.
2. Batista GE, Prati RC, Monard MC. A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD Explorations Newsl. 2004;6(1):20–9.
3. Shatnawi R. Improving software fault-prediction for imbalanced data. In: 2012 international conference on innovations in information technology (IIT); 2012. p. 54–9.
4. Di Martino M, Decia F, Molinelli J, Fernández A. Improving electric fraud detection using class imbalance strategies. In: ICPRAM; 2012. p. 135–41.
5. Majid A, Ali S, Iqbal M, Kausar N. Prediction of human breast and colon cancers from imbalanced data using nearest neighbor and support vector machines. Comput Methods Programs Biomed. 2014;113(3):792–808.
6. Liu Y, Loh HT, Sun A. Imbalanced text classification: a term weighting approach. Expert Syst Appl. 2009;36(1):690–701.
7. Kubat M, Holte RC, Matwin S. Machine learning for the detection of oil spills in satellite radar images. Mach Learn. 1998;30(2–3):195–21515.
8. Su P, Mao W, Zeng D, Li X, Wang FY. Handling class imbalance problem in cultural modeling. In: 2009 IEEE international conference on intelligence and security informatics; 2009. p. 251–6.
9. Abdi Y, Parsa S, Seyfari Y. A hybrid one-class rule learning approach based on swarm intelligence for software fault prediction. Innovations Syst Softw Eng. 2015;11(4):289–301.
10. Ganganwar V. An overview of classification algorithms for imbalanced datasets. Int J Emerg Technol Adv Eng. 2012;2(4):42–7.
11. Kotsiantis S, Kanellopoulos D, Pintelas P. Handling imbalanced datasets: a review. GESTS Int Trans Computer Sci Eng. 2006;30(1):25–36.
12. Ferreira AJ, Figueiredo MA. Boosting algorithms: a review of methods, theory, and applications. In: Ensemble machine learning. Boston: Springer; 2012. p. 35–85.
13. Wang S, Yao X. Multiclass imbalance problems: analysis and potential solutions. IEEE Trans Syst Man Cybern. 2012;42(4):1119–30.
14. Bi J, Zhang C. An empirical comparison on state-of-the-art multi-class imbalance learning algorithms and a new diversified ensemble learning scheme. Knowl-Based Syst. 2018;15(158):81–93.
15. Wu K, Zheng Z, Tang S. BVDT: A boosted vector decision tree algorithm for multi-class classification problems. Int J Pattern Recognit Artif Intell. 2017;31(05):1750016.
16. Leevy JL, Khoshgoftaar TM, Bauder RA, Seliya N. A survey on addressing high-class imbalance in big data. J Big Data. 2018;5(1):42.
17. Abu-Salih B, Chan KY, Al-Kadi O, Al-Tawil M, Wongthongtham P, Issa T, Saadeh H, Al-Hassan M, Bremie B, Albahlal A. Time-aware domain-based social influence prediction. J Big Data. 2020;7(1):10.
18. Sleeman IV WC, Krawczyk B. Bagging Using Instance-Level Difficulty for Multi-Class Imbalanced Big Data Classification on Spark. In2019 IEEE International Conference on Big Data (Big Data) 2019 (pp. 2484–2493). IEEE.
19. Sun Y, Kamel MS, Wang Y. Boosting for learning multiple classes with imbalanced class distribution. In: Sixth international conference on data mining (ICDM'06); 2006. p. 592–602.
20. Zhen L, Qiong L. A new feature selection method for internet traffic classification using ml. Phys Procedia. 2012;1(33):1338–455.
21. Ling CX, Huang J, Zhang H. AUC: a statistically consistent and more discriminating measure than accuracy. Ijcai. 2003;3:519–24.

22. Huang J, Ling CX. Using AUC and accuracy in evaluating learning algorithms. IEEE Trans Knowl Data Eng. 2005;17(3):299–310.
23. Singh A, Purohit A. A survey on methods for solving data imbalance problem for classification. Int J Computer Appl. 2015;127(15):37–41.
24. FernáNdez A, LóPez V, Galar M, Del Jesus MJ, Herrera F. Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. Knowl-Based Syst. 2013;1(42):97–110.
25. Krawczyk B. Learning from imbalanced data: open challenges and future directions. Prog Artif Intell. 2016;5(4):221–32.
26. Tahir MA, Asghar S, Manzoor A, Noor MA. A classification model for class imbalance dataset using genetic programming. IEEE Access. 2019;8(7):71013–377.
27. Ramentol E, Caballero Y, Bello R, Herrera F. SMOTE-RSB*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using SMOTE and rough sets theory. Knowl Inf Syst. 2012;33(2):245–65.
28. Liu A, Ghosh J, Martin CE. Generative oversampling for mining imbalanced datasets. In: DMIN; 2007. p. 66–72.
29. Kumari C, Abulaish M, Subbarao N. Using SMOTE to deal with class-imbalance problem in bioactivity data to predict mTOR inhibitors. In: Proceedings of the international conference on adaptive computational intelligence (ICACI), Mysuru, India; 2019. p. 1–12.
30. Colton D, Hofmann M. Sampling techniques to overcome class imbalance in a cyberbullying context. J Computer-Assist Linguistic Res. 2019;3(3):21–40.
31. Esteves VM. Techniques to deal with imbalanced data in multi-class problems: a review of existing methods.
32. Ling CX, Sheng VS. Cost-sensitive learning and the class imbalance problem. Encyclopedia Mach Learn. 2008;2011:231–5.
33. Maheshwari S, Agrawal J, Sharma S. New approach for classification of highly imbalanced datasets using evolutionary algorithms. Int J Sci Eng Res. 2011;2(7):1–5.
34. Błaszczyński J, Stefanowski J. Neighbourhood sampling in bagging for imbalanced data. Neurocomputing. 2015;20(150):529–42.
35. Rokach L. Ensemble-based classifiers. Artif Intell Rev. 2010;33(1–2):1–39.
36. Schapire RE. A brief introduction to boosting. Ijcai. 1999;99:1401–6.
37. Breiman L. Bagging predictors. Mach Learn. 1996;24(2):123–40.
38. Galar M, Fernandez A, Barrenechea E, Bustince H, Herrera F. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. IEEE Trans Syst Man Cybern. 2011;42(4):463–84.
39. Zhang Z, Krawczyk B, Garcìa S, Rosales-Pérez A, Herrera F. Empowering one-vs-one decomposition with ensemble learning for multi-class imbalanced data. Knowl-Based Syst. 2016;15(106):251–63.
40. Krawczyk B. Combining one-vs-one decomposition and ensemble learning for multi-class imbalanced data. In: Proceedings of the 9th international conference on computer recognition systems CORES 2015. Cham: Springer; 2016. p. 27–36.
41. Feng W, Huang W, Ren J. Class imbalance ensemble learning based on the margin theory. Appl Sci. 2018;8(5):815.
42. Schapire RE, Singer Y. BoosTexter: A boosting-based system for text categorization. Mach Learn. 2000;39(2–3):135–68.
43. Freund Y, Schapire RE. A decision-theoretic generalization of on-line learning and an application to boosting. In: European conference on computational learning theory. Heidelberg: Springer; 1995. p. 23–37.
44. Hastie T, Rosset S, Zhu J, Zou H. Multi-class adaboost. Stat Interface. 2009;2(3):349–60.
45. Friedman J, Hastie T, Tibshirani R. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). Ann Stat. 2000;28(2):337–407.
46. Sun P, Reid MD, Zhou J. An improved multiclass LogitBoost using adaptive-one-vs-one. Mach Learn. 2014;97(3):295–32626.
47. Li P. Abc-logitboost for multi-class classification. arXiv preprint: arXiv:0908.4144. 2009.
48. Sun P, Reid MD, Zhou J. Aoso-logitboost: Adaptive one-vs-one logitboost for multi-class problem. arXiv preprint: arXiv:1110.3907. 2011.
49. Friedman JH. Greedy function approximation: a gradient boosting machine. Ann Stat. 2001;1:1189–232.
50. Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. CatBoost: unbiased boosting with categorical features. In: Advances in neural information processing systems. 2018. p. 6638–48.
51. Chen T, Guestrin C. Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining; 2016. p. 785–94.
52. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu TY. Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in neural information processing systems; 2017. p. 3146–54.
53. Chawla NV, Lazarevic A, Hall LO, Bowyer KW. SMOTEBoost: Improving prediction of the minority class in boosting. In: European conference on principles of data mining and knowledge discovery. Springer: Berlin; 2003. p. 107–19
54. Seiffert C, Khoshgoftaar TM, Van Hulse J, Napolitano A. RUSBoost: A hybrid approach to alleviating class imbalance. IEEE Trans Syst Man Cybern Syst Hum. 2009;40(1):185–97.
55. Rayhan F, Ahmed S, Mahbub A, Jani MR, Shatabda S, Farid DM, Rahman CM. MEBoost: mixing estimators with boosting for imbalanced data classification. In: 2017 11th international conference on software, knowledge, information management and applications (SKIMA); 2017. p. 1–6.
56. Sun Y, Kamel MS, Wong AK, Wang Y. Cost-sensitive boosting for classification of imbalanced data. Pattern Recogn. 2007;40(12):3358–78.
57. Fan W, Stolfo SJ, Zhang J, Chan PK. AdaCost: misclassification cost-sensitive boosting. Icml. 1999;99:97–105.
58. Ting KM. A comparative study of cost-sensitive boosting algorithms. In: Proceedings of the 17th international conference on machine learning. 2000.
59. Domingo C, Watanabe O. MadaBoost: A modification of AdaBoost. In: COLT; 2000. p. 180–9.
60. Joshi MV, Agarwal RC, Kumar V. Predicting rare classes: can boosting make any weak learner strong? In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining; 2002. p. 297–306.

Tanha *et al. J Big Data*   (2020) 7:70

Page 47 of 47

61. Joshi MV, Kumar V, Agarwal RC. Evaluating boosting algorithms to classify rare classes: comparison and improvements. In: Proceedings 2001 IEEE international conference on data mining; 2001. p. 257–64.
62. Vezhnevets A, Vezhnevets V. Modest AdaBoost-teaching AdaBoost to generalize better. Graphicon. 2005;12(5):987–97.
63. Mease D, Wyner A, Buja A. Cost-weighted boosting with jittering and over/under-sampling: Jous-boost. J Mach Learn Res. 2007;8:409–39.
64. Jin X, Hou X, Liu CL. Multi-class AdaBoost with hypothesis margin. In: 2010 20th international conference on pattern recognition. 2010. p. 65–8.
65. Chen S, He H, Garcia EA. RAMOBoost: ranked minority oversampling in boosting. IEEE Trans Neural Netw. 2010;21(10):1624–42.
66. Saberian MJ, Vasconcelos N. Multiclass boosting: theory and algorithms. In: Advances in neural information processing systems; 2011. p. 2124–32.
67. Galar M, Fernández A, Barrenechea E, Herrera F. EUSBoost: enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. Pattern Recogn. 2013;46(12):3460–71.
68. Díez-Pastor JF, Rodríguez JJ, García-Osorio C, Kuncheva LI. Random balance: ensembles of variable priors classifiers for imbalanced data. Knowl-Based Syst. 2015;1(85):96–111.
69. Ahmed S, Rayhan F, Mahbub A, Jani MR, Shatabda S, Farid DM. LIUBoost: locality informed under-boosting for imbalanced data classification. In: Emerging technologies in data mining and information security. Singapore: Springer; 2019. p. 133–44.
70. Kumar S, Biswas SK, Devi D. TLUSBoost algorithm: a boosting solution for class imbalance problem. Soft Comput. 2019;23(21):10755–67.
71. Deng X, Liu Q, Deng Y, Mahadevan S. An improved method to construct basic probability assignment based on the confusion matrix for classification problem. Inf Sci. 2016;1(340):250–61.
72. Chicco D, Jurman G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. BMC Genomics. 2020;21(1):6.
73. Saito T, Rehmsmeier M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLoS ONE. 2015;10:3.
74. Halimu C, Kasem A, Newaz SS. Empirical Comparison of Area under ROC curve (AUC) and Mathew Correlation Coefficient (MCC) for evaluating machine learning algorithms on imbalanced datasets for binary classification. In: Proceedings of the 3rd international conference on machine learning and soft computing; 2019. p. 1–6.
75. Rahman MS, Rahman MK, Kaykobad M, Rahman MS. isGPT: An optimized model to identify sub-Golgi protein types using SVM and Random Forest based feature selection. Artif Intell Med. 2018;1(84):90–100.
76. Jurman G, Riccadonna S, Furlanello C. A comparison of MCC and CEN error measures in multi-class prediction. PLoS ONE. 2012;7:8.
77. Zhang ZL, Luo XG, García S, Tang JF, Herrera F. Exploring the effectiveness of dynamic ensemble selection in the one-versus-one scheme. Knowl-Based Syst. 2017;1(125):53–63.
78. Singh PK, Sarkar R, Nasipuri M. Significance of non-parametric statistical tests for comparison of classifiers over multiple datasets. Int J Comput Sci Math. 2016;7(5):410–42.
79. Demšar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res. 2006;7:1–30.
80. Wilcoxon F, Katti SK, Wilcox RA. Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test. Selected Tables Math Stat. 1970;1:171–259.

Publisher's Note