Expert Systems With Applications 78 (2017) 225–241

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring

Yufei Xia[a], Chuanzhe Liu[a,*], YuYing Li[b], Nana Liu[a]

[a] School of Management, China University of Mining and Technology, Xuzhou, Jiangsu 221116, PR China
[b] School of Foreign Studies, China University of Mining and Technology, Xuzhou, Jiangsu 221116, PR China

ARTICLE INFO

Article history:
Received 19 November 2016
Revised 8 February 2017
Accepted 9 February 2017
Available online 10 February 2017

Keywords:
Credit scoring
Boosted decision tree
Bayesian hyper-parameter optimization

ABSTRACT

Credit scoring is an effective tool for banks to properly guide decision profitably on granting loans. Ensemble methods, which according to their structures can be divided into parallel and sequential ensembles, have been recently developed in the credit scoring domain. These methods have proven their superiority in discriminating borrowers accurately. However, among the ensemble models, little consideration has been provided to the following: (1) highlighting the hyper-parameter tuning of base learner despite being critical to well-performed ensemble models; (2) building sequential models (i.e., boosting, as most have focused on developing the same or different algorithms in parallel); and (3) focusing on the comprehensibility of models. This paper aims to propose a sequential ensemble credit scoring model based on a variant of gradient boosting machine (i.e., extreme gradient boosting (XGBoost)). The model mainly comprises three steps. First, data pre-processing is employed to scale the data and handle missing values. Second, a model-based feature selection system based on the relative feature importance scores is utilized to remove redundant variables. Third, the hyper-parameters of XGBoost are adaptively tuned with Bayesian hyper-parameter optimization and used to train the model with selected feature subset. Several hyper-parameter optimization methods and baseline classifiers are considered as reference points in the experiment. Results demonstrate that Bayesian hyper-parameter optimization performs better than random search, grid search, and manual search. Moreover, the proposed model outperforms baseline models on average over four evaluation measures: accuracy, error rate, the area under the curve (AUC) H measure (AUC-H measure), and Brier score. The proposed model also provides feature importance scores and decision chart, which enhance the interpretability of credit scoring model.

© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

Granting loans to potential borrowers is the core business of banks worldwide. The enormous default loss and keen competition require financial intermediations to discriminate applicants accurately and effectively. Thus, banks should determine whether or not to extend credit at the time of application screening. Data are mainly obtained from application tables, customer demographics, and abundant records of past borrowing and repaying actions. Normally, the problem of credit scoring is transformed into binary or multiclass classification. In other words, classifiers are developed with the credit data to construct decision support systems, thereby assisting banks in deciding whether or not to grant loans to specific applications. The predictive models applied to credit scoring can be roughly divided into two groups: statistical approaches and artificial intelligence (AI) methods. Numerous studies have focused on modeling methods that offer a novel algorithm to enhance the accuracy of credit scoring. These methods include statistical methods, such as linear discriminate analysis (LDA) (Altman, 1968) and logistic regression (LR) (Wiginton, 1980), and artificial intelligence methods, such as artificial neural network (NN); (West, 2000), support vector machine (SVM) (Huang, Chen, & Wang, 2007; Min & Lee, 2005), and decision tree (DT) (Lee, Chiu, Chou, & Lu, 2006; Nie, Rowe, Zhang, Tian, & Shi, 2011). Despite the breakthrough of AI-based approaches, simple models, such as LDA and LR, still remain to be popular credit scoring methods because of their easy implementation and accuracy (Finlay, 2011; Lessmann, Baesens, Seow, & Thomas, 2015).

A single complex classifier is not a considerable solution to credit scoring according to the famous “no free lunch theorem” (Wolpert & Macready, 1997); because datasets derived from different banks have their own properties in sizes, data structures, and predictive variables (features), thereby suggesting that a single algorithm cannot solve all problems effectively. Consequently,

* Corresponding author.
E-mail addresses: xiareyoung@cumt.edu.cn (Y. Xia), rdean@cumt.edu.cn (C. Liu), cumtlyy@163.com (Y. Li), liunana1004@163.com (N. Liu).

http://dx.doi.org/10.1016/j.eswa.2017.02.017
0957-4174/© 2017 Elsevier Ltd. All rights reserved.

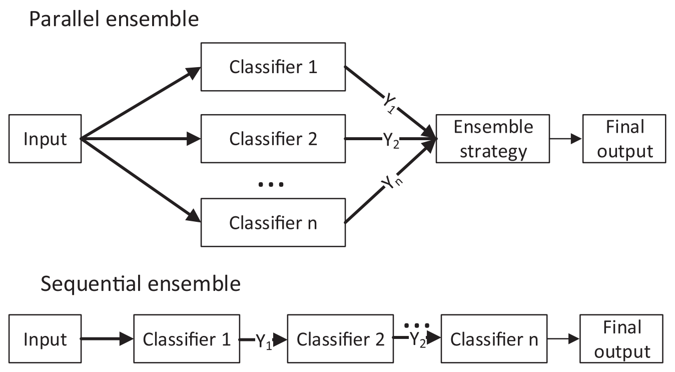

Fig. 1. Diagrams of Parallel ensemble and sequential ensemble.

applying ensemble methods in credit scoring is needed. Ensemble learning combines multiple algorithms that process different hypotheses to form a better hypothesis, thus making good predictions (Nascimento, Coelho, & Canuto, 2014). According to the study of Nanni and Lumini (2009), and Lessmann et al. (2015), ensemble methods perform better than single AI and statistical methods. The promising results encourage further exploration of its structures, combination strategies, and mechanisms.

Ensemble methods can be roughly categorized into two groups according to their structures: parallel and sequential (See Fig. 1) (Duin & Tax, 2000). The parallel ensemble is the combination of different learning algorithms, each of which generates an independent model in parallel. By contrast, the first algorithm in the sequential ensemble learns to generate a model, and then the second algorithm learns to correct the former model, and so on. Despite the active trend of applying ensemble learning to credit scoring, researchers have mainly focused on parallel ensemble methods, such as bagging (Paleologo, Elisseeff, & Antonini, 2010; Wang, Ma, Huang, & Xu, 2012), random forest (RF) (Yeh, Lin, & Hsu, 2012), and multiple classifier systems (MCS) (Ala'raj & Abbod, 2016a; Finlay, 2011; Twala, 2010). The overwhelming parallel ensemble approaches can be attributed to the following: first, these ensemble approaches are simple but accurate. The parallel ensemble is understandable because it can be regarded as a process of consensus decision making. In parallel ensemble, each member (i.e., base model) decides independently on a specific issue, and then the decisions are combined to make final decisions. In most studies, simple sequential ensemble techniques, such as adaptive boosting (AdaBoost), were introduced as benchmark models, whereas parallel ensemble approach was reported to achieve better overall performances (Brown & Mues, 2012). However, advanced sequential ensemble method is seldom considered. Second, parallel ensemble is usually noise-resistant. Dietterich (2000) demonstrated that boosting outperformed bagging and randomization in situations with little classification noise while it performed worse than bagging under substantial noise. However, discovering noise and redundant features is common in real-world credit scoring. Third, the parallel structure is easily applied to parallel and distributed learning, which speeds the learning process and makes the model practical for large-scale data. With progressive boosting techniques, these drawbacks can be overcome to some extent. Recent novel boosting techniques claim to provide similar performances compared with state-of-the-art parallel ensemble methods (Brown & Mues, 2012). Feature selection algorithms alleviate the effects of noise and redundant features. Distributed learning and multi-core computation, which completely utilize the resources of the computer to speed learning, are also feasible in boosting techniques (e.g., extreme gradient boosting (XGBoost)). Therefore, further exploration on boosting is highly required in the credit scoring domain.

Recent studies show that boosting technique is a promising approach in the domain of insurance loss modeling (Guelman, 2012), travel time prediction (Yanru Zhang & Haghani, 2015), and chemistry (Brillante et al., 2015). Boosting comprises a family of algorithms. In the field of credit scoring, a few research have focused on AdaBoost with DT and NN as base models (Tsai, Hsu, & Yen, 2014; Twala, 2010; Wang, Hao, Ma, & Jiang, 2011). However, NN is criticized as a black box (Hand & Henley, 1997). Other boosting techniques have been introduced as well. Finlay (2011) applied two boosting algorithms, i.e., discrete AdaBoost and error-trimmed boosting to logistic regression. Brown and Mues (2012) compared several credit scoring algorithms using imbalanced data and suggested that RFs and gradient boosting (GB) perform comparatively well. However, details, which include hyper-parameter setting and interpretability, were not emphasized.

Apart from accuracy, interpretability is also a vital component in the credit scoring domain. First, banking supervision authorities usually require banks to subject to comprehensive credit risk models, verifying the soundness of bank choices (Florez-Lopez & Ramon-Jeronimo, 2015). Second, comprehensive and understandable models are required to persuade managers to replace simple models with complex but accurate ensembles, because the idea of building various models and then combining their results into the final decision is occasionally counter-intuitive (Lessmann et al., 2015). Third, combining expert knowledge with established models for comprehensive models is easy (Finlay, 2011). However, most literature only focused on the accuracy and tended to ignore interpretability. Among the limited number of studies that have tried to balance the accuracy and interpretability of credit scoring models, Paleologo et al. (2010) introduced a novel bagging algorithm (subagging) to several classifiers in the credit scoring domain. The authors discovered that the use of subagging improved the performances of base learners and subagging DTs achieved better performances while maintaining reasonable interpretability. However, exact decision rules were not extracted, except for the tree sizes and feature importance scores. Florez-Lopez and Ramon-Jeronimo (2015) proposed a correlation-based decision forest model aiming to enhance both accuracy and interpretability. They measured interpretability with rules or trees in a model and compared the proposal with RF and GB. The authors also suggested that the decision forest model outperformed baseline models in both accuracy and interpretability. Wu and Hsu (2012) developed a decision support model based on DT for decision making in credit risk assessment. They suggested that the model had promising forecasting performance, general applicability, and explanatory power because it generated various rules with relevance vector machine and DT. However, the decompositional knowledge acquisition structure was inefficient, because relevance vector machine is complex and costly.

In this paper, the entire process of modeling, including feature selection, training, automatic hyper-parameter tuning, and evaluation, is considered. The main part of our model applies XGBoost (Chen & He, 2015), an advanced GB algorithm that obtains state-of-the-art results in Kaggle machine learning competitions. This paper aims to explore an accurate and comprehensive credit scoring model based on XGBoost. The study of Zięba, Tomczak, and Tomczak (2016) also applied XGBoost in bankruptcy prediction. However, their study ignored the hyper-parameter tuning of XGBoost, and interpretability was not highlighted in their work. Our experimental results demonstrate that the proper setting of hyper-parameters has substantial influences on the performances of XGBoost. Consequently, tree-structured Parzen estimator (TPE) (Bergstra & Bengio, 2012), which is a Bayesian hyper-parameter optimization method, is employed to tune XGBoost. The performances of XGBoost tend to be improved further with careful parameter tuning. Moreover, the feature importance scores and decision chart are provided to enhance model interpretation.

The remainder of our work is organized as follows. Section 2 introduces the mechanism of GB, XGBoost, and Bayesian hyper-parameter optimization method. Section 3 explains the proposed tree boosting model. Section 4 specifies the experiment set-up from three aspects: dataset, benchmark models, and evaluation measure. Section 5 presents the experimental results. Finally, conclusions and future work are described in Section 6.

2. Methodology

2.1. Gradient boosting

Unlike the bagging algorithm that fits the base models in parallel, the boosting approach sequentially builds models. The basic idea of boosting is combining a series of weak base learners, which are normally regression trees (see Appendix A for details), into a strong one. The weak learner herein refers to a model that only performs slightly better than a random guess. Boosting fits additive base learners to minimize the loss function provided. Loss function measures how well the model fits the current data. The process of boosting continues until the loss function reduction becomes limited.

Given a dataset $\{x_i, y_i\}$ with N samples, where x denotes a set of features and y denotes the response variable, the focus is on the estimation of a true but unknown function $F^*(x)$ that maps features to the response variable, such that a specific loss function $\Psi(y, F(x))$ is minimized:

$$F^*(x) = \arg min_F \ \Psi(y, F(x)). \tag{1}$$

Please note that $F^*(x)$ is a special member of function $F(x)$ that minimizes a certain loss function. Friedman (2001) provided a solution to function estimation and proposed the GB method. GB approximates $F^*(x)$ in an "additive" form: GB is sequentially updated by taking the sum of previous single base models. GB starts from an initial guess $f_0(x)$ and for number of boosts $k = 1, 2, \ldots, K$

$$F_k^*(x) = \sum_{k=0}^{K} f_k(x) = \sum_{k=0}^{K} \beta_k h(x; \theta_k), \tag{2}$$

where $f_k(x)$ is the optimal base models in each boost. A common practice is to represent the optimal base learner $f_k(x)$ as a parameterized function $\beta_k h(x; \theta_k)$, where β_k and θ_k denote the best expansion coefficient and best parameters in base learner, respectively. Thus, determining an optimal model $F^*(x)$ can be solved by changing the function optimization problem (Eq. (1)) into a "greedy-stagewise" parameter optimization problem:

$$(\beta_k, \theta_k) = \arg min \sum_{i=1}^{N} \Psi(y_i, F_{k-1}(x_i) + \beta h(x_i; \theta)), \tag{3}$$

Combine Eqs. (2) and (3), the following equation can be derived:

$$F_k(x) = F_{k-1}(x) + \beta_k h(x; \theta_k). \tag{4}$$

The basic idea of greedy-stagewise optimization approach is that for every boost, the currently optimal β_k and θ_k are searched to build the optimal base learner. By summing these base learners, $F^*(x)$ can thus be approximated. To approximately solve Eq. (3), the function is optimized in a parametric space and solved by steepest descent step: given the current approximation $F_{k-1}(x)$ at the k-th boost, we aim to find a specific base learner $\beta_k h(x; \theta_k)$ that minimizes the loss function. The $h(x; \theta_k)$, which is a possible choice of function $h(x; \theta)$, can be interpreted as the step "direction" here. As the negative gradient of loss function is the steepest descent step direction given $F_{k-1}(x)$, it is intuitive to search the optimal direction $h(x; \theta_k)$ in the members of $h(x; \theta)$ that most parallel to the negative gradient in the infinite data space. Particularly, GB employs a two-stage optimization procedure as follows (Friedman, 2001). First, the function $h(x; \theta)$ is fitted by the least squares to the negative gradient of loss function (or the pseudo-residuals):

$$\theta_k = \arg min \sum_{i=1}^{N} [-g_k(x_i) - \beta h(x_i; \theta)]^2, \tag{5}$$

where the gradient of loss function at point x_i (i.e., $g_k(x_i)$) is computed as

$$g_k(x_i) = \left[\frac{\partial \Psi(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{k-1}(x)}. \tag{6}$$

Then, given the optimal base learner $h(x; \theta_k)$, the optimal scale coefficient β_k is determined as follows:

$$\beta_k = \arg min \sum_{i=1}^{N} \Psi(y_i, F_{k-1}(x_i) + \beta h(x_i; \theta_k)). \tag{7}$$

Therefore, the model in Eq. (4) can be updated.

GB specializes to the case where the base learner is a J-terminal node regression tree. For the k-th boost, the regression tree partitions the input space into J disjoint regions (i.e., terminal nodes) $\{R_{jk}\}_{j=1}^{J}$. Then the base learner can be denoted as follows.

$$h_k\left(x; \left\{R_{jk}\right\}_{j=1}^{J}\right) = \sum_{j=1}^{J} b_{jk} I(x \in R_{jk}), \tag{8}$$

Where b_{jk} is the prediction on the j-th terminal node (See Appendix A for details). $I(\cdot)$ is an indicator function. Notably, $I = 1$ if and only if $x \in R_{jk}$; else $I = 0$. The $\{R_{jk}\}_{j=1}^{J}$ in Eq. (8), which is determined by the splitting feature and the splitting point, is a special form of θ_k. In GB, the $\{R_{jk}\}_{j=1}^{J}$ is obtained using a least-squares splitting criterion (see Appendix A for details). Instead of estimating a single β_k for the each base learner, Friedman (2001) suggested to solve the Eq. (7) separately within each node of regression tree at the k-th boost. Thus, we define γ_{jk} as a set of optimal expansion coefficients with one parameter corresponding to each node in the tree at k-th boost. Because regression tree predicts a constant value within each terminal node, the solution to Eq. (7) can be converted to:

$$\gamma_{jk} = argmin_{\gamma} \sum_{x_i \in R_{jk}} \Psi(y_i, F_{k-1}(x_i) + \gamma). \tag{9}$$

By this means, separate optimal value γ_{jk} for each terminal node is chosen. Then the current approximation $F_{k-1}(x)$ is separately updated in each terminal node.

$$F_k(x) = F_{k-1}(x) + \gamma_{jk} I\left(x \in R_{jk}\right). \tag{10}$$

GB may also suffer from over-fitting because it subsequently adds new base learners for minimizing a certain loss function. GB employs several mechanisms to prevent over-fitting. The first mechanism is the number of iteration (or the number of trees) and learning rate. Learning rate smooths the contribution of each iteration by shrinking its value to (0, 1). Thus, Eq. (4) can be updated to:

$$F_K(x) = F_{K-1}(x) + \tau \beta_K h(x; \theta_K). \tag{11}$$

Parameter τ, which functions as the learning rate, makes the boosting process more conservative and robust to over-fitting. A trade-off exists between the learning rate and the number of iteration. A small value of learning rate generally requires a larger number of iteration to ensure sufficient convergence. The empirical results of Friedman (2001) have revealed that small value of learning rate ($\tau < 0.1$) leads to better generalization error.

Tree complexity is another important parameter. In GB, tree complexity is controlled by two alternative hyper-parameters: the maximum tree depth[1] and the tree size[2] (See Appendix A for an example). Either maximum tree depth or tree size is used to control tree complexity. If the maximum tree depth of a decision tree is defined as D, the tree size J is at most 2^D. Small trees are preferred in GB unlike RF, which grows trees without pruning, because large trees easily over-fit and are not interpretative. An optimal tree depth ensures that the base learner is simple but captures the important details of features in the training set. A trade-off also exists between tree complexity and the number of iterations. For the given learning rate, fitting complex trees result in fewer iterations to ensure convergence. Hastie, Tibshirani, and Friedman (2009) suggested that tree size between 4 and 8 worked well in GB and reported that 6 was a preferable candidate because other choices seldom provided significant improvement. In the user guide of scikit-learn,[3] the authors suggested that model with "tree size = J" achieved similar performance to the one with "maximum tree depth = J − 1″. Thus, a maximum tree depth between 3 and 7 is a considerable option.

In a working guide on gradient boosting ((Elith, Leathwick, & Hastie, 2008), the aforementioned relationships among learning rate, tree complexity, and the number of iteration are summarized. The experimental results suggested that without consideration to the computing time, the best strategy was employing high tree complexity and low learning rate, in which a substantial number of iterations is necessary for convergence. However, the authors also indicated that building a substantial number of trees was time-consuming for real-world applications.

Randomization also controls over-fitting and further increases the accuracy of GB. Specifically, a subset of the training set at each iteration is drawn at random with replacement. Then, the randomly drawn subset is used to fit the base learner. This model is known as stochastic GB. Friedman (2002) claimed that both the approximation accuracy and computation speed can be improved by subsampling.

2.2. Extreme gradient boosting

XGBoost, which is an advanced GB system, was recently proposed by Chen and He (2015). XGBoost optimizes the objective function and its estimation. The fast, efficient, and scalable system achieves promising results on numerous standard classification benchmarks. In machine learning competitions of Kaggle, numerous winning solutions employ XGBoost to train models. In the following paragraphs, the framework of XGBoost is briefly discussed.

With regard to a given dataset with n samples and m features $\mathrm{D} = \{x_i, y_i\}$, where x and y denotes the features and response variable, respectively; similar to GB, XGBoost uses K additive function $f_k(x)$ to approximate the function $F_K(x)$ (i.e., $F_K(x) = \sum_{k=1}^{K} f_k(x)$), where $F_K(x)$ denotes the prediction on K-th boost. Notably, XGBoost uses a specific form for a base learner: $f_k(x)$ is a CART and can be denoted as $\omega_{q(x)}, \mathrm{q} \epsilon \{1, 2, \ldots, \mathrm{T}\}$, where T denotes the number of leaves. Specifically, q represents the decision rules of the tree, and ω is a vector denoting the sample weight of leaf nodes (i.e., leaf scores). In other words, q describes how the samples are classified into leaves, and ω denotes the leaf weight on each node of the tree.

In XGBoost, the loss function is expanded to the objective function by adding a regularization term:

$$\mathcal{L}_K(F(x_i)) = \sum_{i=1}^{n} \Psi(y_i, F_K(x_i)) + \sum_{k=1}^{K} \Omega(f_k), \tag{12}$$

where $F_K(x_i)$ is the prediction on the i-th sample at the K-th boost and $\Omega(f) = \gamma T + 0.5 \times \lambda \|\omega\|^2$. In the regularization term, γ is the complexity parameter. λ is a fixed coefficient, and $\|\omega\|^2$ is the L2 norm of leaf weights. The $\Psi(*)$ herein is a specified loss function that measures the differences between the prediction and the real target. $\Omega(*)$ is the regularization term that penalizes the complexity of the model. The regularized objective function, which is inspired by the regularized greedy forest (Johnson & Zhang, 2014), tends to smooth the contributions of base learners to avoid over-fitting. The objective function is reduced to the loss function of GB when the regularization term is removed.

The objective function represented in Eq. (12) can also hardly be optimized in the function space directly. Similarly, XGBoost is trained in an additive form at the step k:

$$\mathcal{L}_k = \sum_{i=1}^{n} \Psi(y_i, F_{k-1}(x_i) + f_k(x_i)) + \Omega(f_k). \tag{13}$$

The goal of XGBoost is to find the f_k that minimizes the objective function. In GB, the objective function is optimized with gradient descent where only the first order gradient statistics are used. While in XGBoost, Eq. (13) is quickly approximated with Taylor expansion. For simplicity, after removing the constant term, Eq. (13) can be transformed as follows:

$$\mathcal{L}_k \cong \sum_{i=1}^{n} \left[g_i f_k(x_i) + \frac{1}{2} h_i {f_k}^2(x_i) \right] + \Omega(f_k), \tag{14}$$

where g_i and h_i denote the first and second order gradient statistics on the loss function, respectively. Concretely, $g_i = \partial_{F_{k-1}(x_i)} \Psi(y, F_{k-1}(x_i))$ and $h_i = \partial^2_{F_{k-1}(x_i)} \Psi(y, F_{k-1}(x_i))$. Let I_j denote as the sample set of leaf j. Eq. (14) can be transformed as follows:

$$\mathcal{L}_k = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) \omega_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) \omega_j^2 \right] + \gamma T. \tag{15}$$

Given a fixed tree structure $q(x)$, the optimal leaf weight scores on each leaf node ω_j^*, and the extreme value of $\mathcal{L}_K$ can be solved by a simple quadratic programming:

$$\omega_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}, \tag{16}$$

$$\mathcal{L}_K^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{17}$$

Thus, the best base learner $f_k = \omega^*_{q(x)}$ is achieved. The $\mathcal{L}_K^*$ can also be regarded as a structure scoring function that measures how suitable a given vector of leaf scores is. A smaller value is preferred because it better fits the data. Note that both score in XGBoost and Gini impurity (see Appendix A) evaluate a certain tree structure. However, the difference lies on that score measures the structure of entire decision tree, while Gini impurity only measures the splitting point for one node.

The core problem of XGBoost is to determine the optimal tree structure. An intuitive searching algorithm is summarized as follows: enumerate all possible tree structures, calculate the scores according to Eq. (17), and then finds the optimal one. However, infinite tree structures exist; thus, enumeration is computationally costly. XGBoost employs a greedy search algorithm in finding an optimal tree structure. The algorithm is the so called "exact greedy

[1] Maximum tree depth is defined as the number of edges along the longest path from the root node to the farthest terminal node.

[2] Tree size is defined as the number of terminal nodes in a tree.

[3] http://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting.

algorithm for splitting finding" (Chen & Guestrin, 2016). XGBoost starts from the root node and iteratively splits the features. Suppose I_L and I_R denote the sample set of the left and right branch, respectively. For each node on the tree, XGBoost attempts to add a split. The change of objective function, i.e., the gain after adding the split, is described as follows:

$$G = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{18}$$

Eq. (18) obviously comprises four parts, denoting the score of left branch, score of right branch, score without splitting and complexity, respectively. XGBoost stops splitting if $G < 0$, implying that the parameter γ herein penalizes the complexity and additional splits. The splitting point with the largest G is considered as the optimal splitting on the node.

The exact greedy search algorithm is still costly for processing high-scale and high-dimensional data. Consequently, XGBoost provides an approximate greedy algorithm for finding optimal tree structure. The approximate algorithm first proposes the splitting point according to the percentiles of feature distribution. Subsequently, the approximate algorithm maps the variables into buckets split by these splitting, calculate the first- and second-order of gradient statistics on the loss function and finds the best candidate points among proposals based on the scores computed according to Eq. (18). Chen and Guestrin (2016) suggested that approximate algorithm is an efficient searching method and is accurate as the greedy searching algorithm given sufficient iterations.

XGBoost follows the idea of GB and employs the learning rate and boosting numbers, maximum tree depth, and subsampling to avoid over-fitting. Moreover, three additional hyper-parameters are proposed in XGBoost. Hyper-parameter $\gamma \in (0, +\infty)$ defines the minimum loss reduction for further partition, as shown in Eq. (17). A substantial γ makes splitting more conservative, thus controlling the tree depth. $\omega_{mc} \in (0, +\infty)$ is defined as the minimum sum of the necessary instance weight in further splitting. If the sum of the instance weight after splitting is less than ω_{mc}, then the tree will not provide further partition. Similar to γ, a substantial ω_{mc} also makes the model more conservative in partitioning. XGBoost also introduces column subsampling techniques, i.e., feature-based subsampling. The hyper-parameter enables XGBoost to randomly collect parts of features in growing trees, thereby speeding learning and avoiding over-fitting. Similar to GB, a tradeoff also exists between the number of iteration and learning rate. Assume that tree complexity is controlled, lower learning rate requires a substantial number of iteration to ensure convergence.

2.3. Bayesian hyper-parameter optimization

Classification algorithms are rarely parameter-free. The commonly used credit scoring models referred in this paper, such as SVM, NN, CART, and RF, as well as GB and XGBoost, all have several hyper-parameters which dramatically influence the accuracy of models. Therefore, careful tuning of these hyper-parameters, i.e., hyper-parameter optimization, is important. However, tuning is often regarded as a "black art," which mainly depends on subjective judgments, experience, and trial-and-error method (Bergstra, Bardenet, Bengio, & Kégl, 2011). Although grid search (GS) has been applied to the hyper-parameter tuning of SVM and NN, the two algorithms may be infeasible to XGBoost because of the substantial number of hyper-parameters included in XGBoost. Consequently, Bayesian optimization, which achieves state-of-the-art results in a few global optimization problems, is a better solution to hyper-parameter tuning. In this subsection, Bayesian hyper-parameter optimization is roughly introduced.

```
Algorithm.1 Sequential model-based global optimization algorithm (SMBO)
1: Initialization $S_0; H = \emptyset$
2: for $t = 1$ to $T$ do
3:     $\lambda^* = argmin S_{t-1}(\lambda)$
4:     $c = L(\lambda^*)$
5:     $H = H \bigcup (\lambda^*, c)$
6:     fit new model $S_t$ according to updated $H$
7: end for
8: Return $\lambda$ with minimum $c$ in $H$
```

Fig. 2. Pseudocode of sequential model-based global optimization (SMBO).

2.3.1. Parameters and hyper-parameters

The final goal of a machine learning problem is to determine a mapping $y = G(\mathbf{x}, \theta)$, where y is the output, $\mathbf{x}$ is the input vector, and the map G is parameterized by a vector $\boldsymbol{\theta}$. For example, the parameters in a backpropagation (BP) neural network are the weights of the nodes, which can be estimated by an optimization method, such as gradient descent (Simon, 2008). However, the learning algorithm itself may also have parameters that cannot be directly learned from the input and requires fixing before model training. These parameters are called "hyper-parameters" because they are assumed to be "higher-level," relative to parameters estimated from data. The number of hidden layers and the corresponding number of neurons are typical hyper-parameters for the BP neural network. Generally speaking, learning algorithms learn parameters to fit the data well, whereas hyper-parameters mainly control the complexity or regularization to refine the model.

2.3.2. Sequential model-based global optimization (SMBO)

Bayesian hyper-parameter optimization assumes the presence of a noisy black-box function mapping from hyper-parameters to a specific objective (e.g., model performances). The methodology gathers hyper-parameters (input) and the corresponding model performances (observations) to infer information on the unknown function. Particularly, an SMBO algorithm is proposed, as shown in Fig. 2 (Hutter, Hoos, & Leyton-Brown, 2011). SMBO first builds a model S to map the hyper-parameter settings λ to the loss function L (line 1 in Fig. 2), and an H (or trails) to record the settings and the corresponding loss. The loss function L is used to evaluate a specific hyper-parameter setting. The trails assist in the inspection inside the Bayesian hyper-parameter optimization, and display the parameter settings and the corresponding evaluations. Then, SMBO iterates the following steps: find the local optimal hyper-parameter settings λ^* according to the current model S_{t-1}; calculate the loss c under the settings λ^*; store λ^* and the corresponding loss c in H; and fit a new model S_t according to the updated record H. T is the pre-defined maximum iteration. When the loop ends, the SMBO outputs the global optimal hyper-parameter settings with the minimum c.

Thereafter, the core problem is to determine a local optimal hyper-parameter settings λ^* according to the current model S_t for the tth loop. SMBO employs an acquisition function to tackle the issue. Particularly, the acquisition function uses the predictive information of S_t at any possible hyper-parameter settings. Afterward, SMBO finds the most promising settings by maximizing the acquisition function. Two forms of acquisition function are suggested: expected improvement (EI) and the probability of improvement. In this paper, EI is chosen as the criterion because it performs well and is intuitive (Bergstra et al., 2011). For simplicity, let c_{min} denote the current minimum loss in H, and $c(\lambda)$ denote the loss under the hyper-parameter setting λ. Then, the improvement can be described as follows:

$$I(\lambda) = \max\left(c_{min} - c(\lambda), 0\right). \tag{19}$$

Table 1
Searching space of hyper-parameter optimization approach.

Parameter	Symbol	TPE	Random search	Grid search	Manual search
Learning rate	τ	0.1	0.1	0.1	0.1
Number of boost	N	60	60	60	60
Maximum tree depth	D_{max}	(1,12)	(1,12)	1,2,3	0.2*$N_{feature}$
Subsample ratio	r_s	(0.9,1)	(0.9,1)	0.9,0.95,1	0.9
Column subsample ratio	r_c	(0.9,1)	(0.9,1)	0.9,0.95,1	0.9
Minimum child weight	ω_{mc}	(0,4)	(0,4)	0,1,2,3,4	1
Maximum delta step	δ	(0,1)	(0,1)	0.4,0.6,0.8,1	1
Gamma	γ	(0,0.01)	(0,0.01)	0,0.01	0

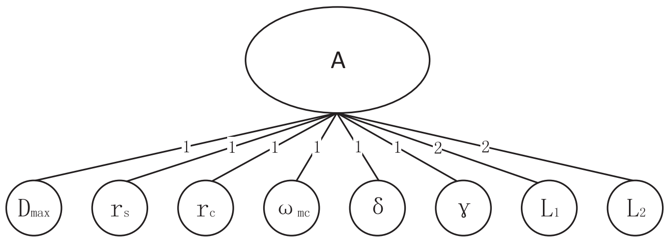

Fig. 3. An example of tree structure in TPE.

The improvement is difficult to estimate directly, because c(λ) is unknown until the settings are evaluated by a loss function (i.e., line 4 in Fig. 2). However, EI can be computed as

$$\mathrm{EI}(\lambda) = \int_{-\infty}^{c_{min}} \max(c_{min} - c(\lambda), 0) \cdot p_{S_t}(c|\lambda)dc. \tag{20}$$

2.3.3. Tree-structured Parzen estimator (TPE)

Under the framework of SMBO, two different Bayesian optimization algorithms, i.e., Gaussian process (GP) approach (Snoek, Larochelle, & Adams, 2012) and TPE approach (Bergstra et al., 2011) are proposed. Hyper-parameter optimization is fundamentally a problem of optimizing a specific map function over a graph-structured configuration space. As suggested by Bergstra et al. (2011), the configuration space is restricted to tree structure in TPE. A simple example of tree structure in TPE for the hyper-parameter optimization in XGBoost is shown in Fig. 3. Suppose the learning rate and the maximum number of boost is set manually (see step 3 in Section 3 for details), the set of hyper-parameters {A, D_{max}, r_s, r_c, ω_{mc}, δ, γ, $L1$, $L2$} ∈ λ and A is a binary hyper-parameter representing the type of base learner, where A = 1 denotes the regression tree and A = 2 denotes the linear model. A is the root node of the tree structure and the remaining hyper-parameters correspond to the leaf nodes. The hyper-parameters for tree base learner are introduced in Table 1. L1 and L2, denoting the L1 regularization coefficient and L2 regularization coefficient respectively, are designed for linear base learner. If and only if the root node A = 1, the leaf nodes hyper-parameters D_{max}, r_s, r_c, ω_{mc}, δ and γ are valid, while L1 and L2 is invalid. TPE assumes that the leaf nodes hyper-parameters are independent; thus, an edge does not exist among the leaf nodes.

Generally speaking, TPE provides simple solutions to determine the model S (i.e., line 6 in Fig. 2) and find the local optimal hyper-parameter settings (i.e., line 3 in Fig. 2). To approximate the function S, GP directly estimates $p_S(c|\lambda)$, whereas TPE models $p_S(c|\lambda)$ indirectly based on $p_S(\lambda|c)$ and $p_S(c)$. In our proposal, the TPE approach is adopted because it reports better results in several difficult learning problems (Thornton, Hutter, Hoos, & Leyton-Brown, 2013). TPE models $p_S(c|\lambda)$, with an alternative density estimate which is conditional on the value of c relative to a threshold c^*:

$$p_S(c|\lambda) = \begin{cases} \ell(\lambda), & c < c^* \\ \mathfrak{g}(\lambda), & c \geq c^* \end{cases}, \tag{21}$$

where $\ell(\lambda)$ is the density estimate formed by the observations regarding the value of loss, which is lower than c^* in H, whereas $\mathfrak{g}(\lambda)$ is generated by the remaining observations. Normally, c^* is set as a γ-quantile of the best-observed c, where the default is $\gamma = 0.15$. To determine the local optimal hyper-parameter setting, Bergstra et al. (2011) proved that EI in Eq. (20) was proportional to the following expression comprising: γ, $\ell(\lambda)$ and $\mathfrak{g}(\lambda)$

$$\mathrm{EI}(\lambda) \propto \left(\gamma + \frac{\mathfrak{g}(\lambda)}{\ell(\lambda)}(1-\gamma)\right)^{-1}. \tag{22}$$

This equation indicates that hyper-parameter settings with high probability under $\ell(\lambda)$ and low probability under $\mathfrak{g}(\lambda)$ are possible to maximize EI.

A Parzen estimator (or Parzen-window estimator) is employed to estimate $\ell(\lambda)$ and $\mathfrak{g}(\lambda)$. Parzen estimator provides the probability density function (PDF) of a random variable in a non-parametric way. Let $(x_1, x_2, \ldots, x_n)$ be independent and identically distributed samples drawn from an unknown density. The Parzen estimator aims to estimate the function f directly from the data:

$$\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{23}$$

where $K(\cdot)$ is the kernel function and $h > 0$ is the so-called bandwidth. Gaussian kernel $K(\frac{x-x_i}{h}) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}(\frac{x-x_i}{h})^2)$ is one of the most often used kernel. For each node in the tree structure, a 1-D Parzen estimator is built to estimate PDF. Parzen estimators are only created for valid nodes. The estimations of $\ell(\lambda)$ and $\mathfrak{g}(\lambda)$ involve a hierarchical structure with continuous and discrete hyper-parameters. For continuous hyper-parameters, 1-D Parzen estimator is created by assuming the density as Gaussian (i.e., Gaussian kernel), with the bandwidth h set to the greater of the distances to the left and right neighbors of each point (Bergstra et al., 2011; Thornton et al., 2013). Discrete hyper-parameters are estimated with probabilities proportional to the occurrences of the corresponding choice in H. TPE starts from the root node to each valid leaf node to evaluate an arbitrary density estimate of hyper-parameter λ. Given A = 1, $\tau = 0.1$ and N = 60, TPE starts from A and descends into the active nodes D_{max}, r_s, r_c, ω_{mc}, δ and γ to evaluate the possibility density of an arbitrary candidate {D_{max}, r_s, r_c, ω_{mc}, δ, γ} ∈ λ (i.e., $f(\lambda)$) either added to $\ell(\lambda)$ or $\mathfrak{g}(\lambda)$ (Fig. 3). Six Parzen estimators are created on nodes D_{max}, r_s, r_c, ω_{mc}, δ and γ, respectively. The leaf nodes are mutually independent; thus, the joint density function $f(\lambda)$ can be computed by multiplying the individual density estimations $f(D_{max})$, $f(r_s)$, $f(r_c)$, $f(\omega_{mc})$, $f(\delta)$ and $f(\gamma)$.

XGBoost is a powerful machine learning algorithm with numerous hyper-parameters. The setting of XGBoost is critically dependent on the predictive capability. In our paper, TPE is employed

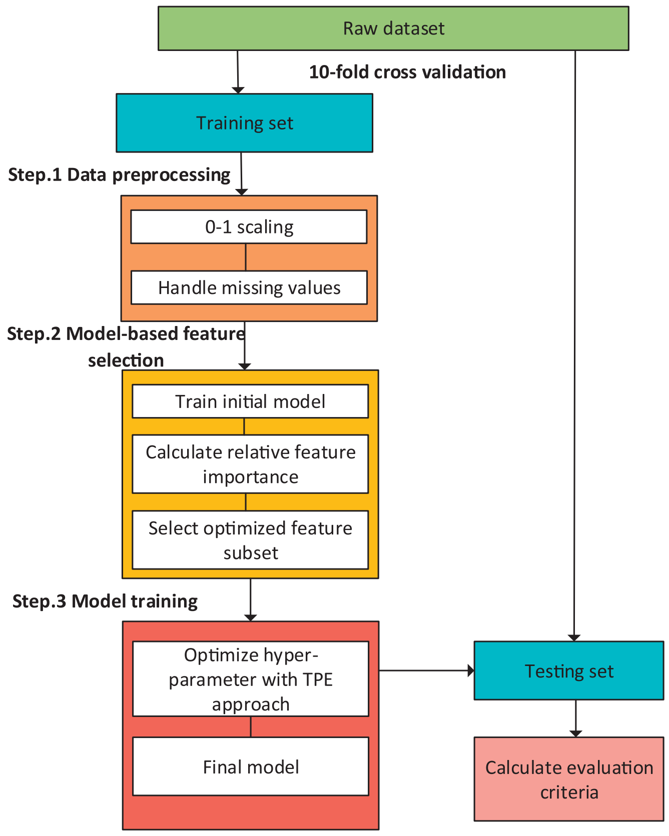
Raw dataset
10-fold cross validation
Training set
Step.1 Data preprocessing
0-1 scaling
Handle missing values
Step.2 Model-based feature selection
Train initial model
Calculate relative feature importance
Select optimized feature subset
Step.3 Model training
Optimize hyper-parameter with TPE approach
Final model
Testing set
Calculate evaluation criteria

Fig. 4. Flowchart of XGBoost-based credit scoring model.

to tune the hyper-parameters in XGBoost carefully. Moreover, grid search (GS), random search (RS), and manual search (MS) are also introduced and compared to demonstrate the efficiency of TPE. The searching spaces of corresponding hyper-parameters are summarized in Table 1.

3. XGBoost-based credit scoring model

In this paper, an XGBoost-based credit scoring model is proposed. The flowchart of the proposal is depicted in Fig. 4. The model can be divided into three steps: data pre-processing, model-based feature selection, and model training. Data pre-processing includes 0–1 scaling and missing value marking. Afterward, a model-based feature selection algorithm is applied to filter unrepresentative features. The feature importance scores are derived from the initial model and sorted. Then, a wrapper feature selection algorithm is utilized according to the calculated feature importance scores. This stage finally outputs a feature subset which is expected to remove redundant features. In the final step, TPE is used to tune the hyper-parameters, and the final model is built with selected features and optimized parameters. The process is discussed below.

Step 1. Data pre-processing

0–1 scaling and missing values marking.

Despite the fact that tree-based algorithms are hardly affected by scaling, distance- or margin-based classifiers for comparisons are heavily dependent on feature standardization. Thus, 0–1 scaling is employed in the data pre-processing step to make more convincing model comparisons. The process of scaling can be omitted for real-world applications of the proposed XGBoost-based model. Let $D = \{X, Y\}$ denote the entire training set, where $X = (x_1, x_2, \ldots x_m)$ is the m-dimension feature space, and $Y \in \{0, 1\}$ denotes the target variables, representing bad and good applicants, respectively. Suppose x is a specific feature in the space, then the 0–1 scaling of x is computed as follows:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{24}$$

where x' represents the normalized value. Missing values are also common in credit scoring. Normally, these missing values are replaced by specific imputation approaches (e.g., average, median, and complex algorithms). XGBoost provides an inherent sparsity-aware splitting algorithm, which learns the best direction to handle these missing values. In data pre-processing, missing values are marked as *Nan* to guide XGBoost to limit the enumeration solely to the expected direction.

Step 2. Model-based feature selection

Calculate the relative feature importance of the initial model and filter redundant features with a wrapper feature selection algorithm.

Similar to CART and RF, tree boosting techniques can output feature importance score, which is a metric for measuring the importance of features in splitting. Moreover, XGBoost provides three alternative feature importance scores: “weight,” “gain,” and “cover.” Specifically, weight denotes the number of times a feature is used for splitting; gain is the average error reduction when a feature is used for splitting; and cover is the average number of samples affected by the split. Therefore, the gain scores of all features in the initial model because it is in line with the original feature importance measure in CART (See Appendix A). Features with high importance scores are regarded as critical variables. Afterward, sequential forward search (SFS) is applied to select features. Particularly, SFS generates a series of feature subsets by putting the most important feature into the subset and iteratively adding the remaining feature with the highest score to the subset. For example, suppose four features, x_1, x_2, x_3, and x_4, are in the initial model with importance scores computed as 0.4, 0.3, 0.2, and 0.1, respectively. Therefore, four feature subsets, namely $\{x_1\}$, $\{x_1, x_2\}$, $\{x_1, x_2, x_3\}$, and $\{x_1, x_2, x_3, x_4\}$ can be sequentially generated. A 10-fold cross-validation logarithmic loss is calculated to select the best subset, while the feature subset corresponding to the lowest logarithmic loss is selected as the best feature set. Despite tree-based model can automatically select important features due to the splitting criterion (e.g., Gini impurity, see Appendix A for details), the motivation of applying feature selection as a separate step is twofold: first, feature selection improves the performance of XGBoost empirically. The phenomenon may be attributed to the fact that feature selection removes the redundant features, which may be used for splitting when building model with all the features. Second, feature selection ensures a fair comparison between different methods because benchmark models such as support vector machine and neural network is highly dependent on the feature selection (Ala’raj & Abbod, 2016b). The initial parameter settings will obviously have non-trivial influences on the relative importance of features. We follow the default setting of most parameters: learning rate = 0.1, subsample ratio, minimum child weight, and maximum delta step set as 1. With regard to maximum tree depth, the empirical results suggest that max depth set as $0.2^*N_{feature}$ performs better. $N_{feature}$ herein is defined as the number of features. In this step, an optimal feature subset is an output to build the model in the final step.

Step 3. Model training

Train XGBoost model with optimized parameters and selected features.

As previously illustrated, XGBoost is a powerful classifier with numerous hyper-parameters that require careful tuning. These hyper-parameters are summarized in Table 1. With regard to parts of the hyper-parameters, we manually set them with reasonable explanations: The loss function used is chosen as logistic loss (or logarithmic loss) function (Bishop, 2006) because the problem of credit scoring is normally simplified to a binary classification problem. When modeling label y as $+1/-1$, the logistic loss is as follows:

$$L_{Logistic} = \log(1 + \exp(-yp)), \tag{25}$$

where y is the true value, and p is the prediction. The remaining parameters are optimized with TPE. The loss function L in TPE is critical to the optimization performance; thus, we try several metrics and finally decide to use 10-fold cross-validation logistic loss to measure the performances of model under corresponding parameters. Whereas in this paper, we model label y as 1/0 because the default and non-default loans are denoted as 0 and 1, respectively. Thus, the logistic loss for $y \in \{0, 1\}$ is as follows (Bishop, 2006):

$$L_{Logistic} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)), \tag{26}$$

where y_i and p_i denote the true value and the probability prediction, respectively. A perfect classifier will have a logarithmic loss of approximately zero. In TPE optimization, hyper-parameter settings with lower logarithmic loss are regarded as the superior setting.

The application of hyper-parameter optimization in XGBoost follows the framework of SMBO (Fig. 2). The model that initializes random values of the hyper-parameters in the specific searching space are summarized in Table 1. The hyper-parameter setting and the corresponding 10-fold cross-validation logistic loss are stored in H. Then, the model iterates the following steps:

(1) estimate PDF for each hyper-parameter with the existing observations in H through Parzen estimator;
(2) compute the joint density function of the combination of hyper-parameters;
(3) select the most promising hyper-parameters settings to maximize EI (See Eq. (22)); and
(4) record hyper-parameters and its evaluation (i.e., 10-fold cross-validation logistic loss) in H.

The iteration stops when the maximum number of iteration is reached.

As described in Sections 2.1 and 2.2, a trade-off exists among maximum tree depth, learning rate and the number of iteration in GB and XGBoost, which violates the independence assumption in the TPE algorithm. Therefore, we introduce a stepwise training process. First, learning rate and the number of boosts are manually determined. We follow the default learning rate 0.1, which is also suggested value in GB (Friedman, 2001). For number of boosts, we set the hyper-parameter as 60 for a trade-off between model performance and computational cost. By this determination, the optimal values for the remaining hyper-parameters, except learning rate and the number of iteration, are acquired. Afterwards, the learning rate is fixed as 0.1 and the search for optimal number of boost is one-dimensional given that other hyper-parameters have been determined. The searching space *is integer interval [1..100]*. The 10-fold cross-validation logarithmic loss under different numbers of boost are computed, and the number corresponding to the lowest logarithmic loss is maintained as the best iteration number. At the end of this stage, an XGBoost model is built with an optimal feature subset and hyper-parameters. This model is used to predict the test set and calculate the evaluation criteria, such as average accuracy, type I error, type II error, AUC-H measure, and Brier score.

Table 2
Description of the datasets in the study.

Dataset	#Samples	#Features	Good/Bad
German	1000	24	700/300
Australian	690	14	307/383
Taiwan	6000	23	3000/3000
P2P-A	2642	11	1322/1320
P2P-B	1421	17	1072/349

4. Experimental set-up

4.1. Credit datasets

In this experiment, five real-world credit datasets are utilized to verify the performances of our proposal. Three credit datasets, i.e., German, Austrian, and Taiwan, from the UCI machine learning repository are used. However, finding other public datasets with numerous samples is difficult. The booming peer-to-peer (P2P) lending provides us with a feasible and innovative solution to the problem because the trading documents in a few lending platforms are transparent and freely downloaded. P2P lending involves identifying the creditworthiness of a certain borrower, which provides a new data source for credit scoring. Moreover, credit scoring for P2P lending has its unique features. First, soft information is highlighted, because hard information is insufficient for P2P lending (Pötzsch & Böhme, 2010). Therefore, soft information, such as social media information and social capital, demonstrates a relation with default risk and can be used as features in a credit scoring model (Chen, Zhou, & Wan, 2016; Zhang et al., 2016). Second, lenders should not only know which loan to invest on but also the specific amount, because a partial fund is possible to each lender. A few interesting studies in this field include Serrano-Cinca and Gutiérrez-Nieto (2016), and Guo, Zhou, Luo, Liu, and Xiong (2016), which aim to provide practical decision support system for lenders in P2P lending. Therefore, two P2P lending datasets, i.e., Lending Club[4] in the U.S. and We.com[5] in China, are employed in the experiments. For simplification, we denote the datasets as P2P-A and P2P-B. The two datasets are derived from raw lending documents of two P2P lending platforms. The features used in this paper conforms with that in traditional credit scoring datasets, which include demographic variables, solvency, and creditworthiness of the borrower, because soft information is unavailable in the trading documents of the Lending Club nor the websites of We.com. The summary of all the datasets is depicted in Table 2.

4.2. Benchmark models

To evaluate the performance of the proposed XGBoost-based credit scoring model, it is compared with other current techniques, including logistic regression (LR), neural network (NN), decision tree (DT), support vector machine (SVM), bagging-NN, bagging-DT, AdaBoost, AdaBoost-NN, random forest (RF), gradient boosting decision tree (GBDT), XGBoost optimized with manual search (TPE-MS), random search (TPE-RS) and grid search (TPE-GS).

Among the benchmark models, two popular ensemble methods, namely bagging (Breiman, 1996) and AdaBoost (Freund & Schapire, 1995), are included. Given a training set with m samples, bagging generate multiple new datasets (or bootstrap samples), by randomly sampling with replacement from the training set. Then for each bootstrap sample, a model is built and the predictions of all the models are combined by a specific fusion strategy such as majority voting. AdaBoost is a boosting algorithm that sequentially

[4] www.lendingclub.com.
[5] www.we.com.

combines weak learners to a strong one. In each iteration, AdaBoost modifies the weights of the training samples based on the errors of previously created classifiers. Misclassified samples in the training set are assigned with greater weights. A weighted voting scheme is then applied to build the final model. The outperforming ensemble methods can be explained by the famous bias-variance tradeoff. For a training set, the expected error of a classifier can be decomposed into the sum of three terms: bias, variance, and noise (Bauer & Kohavi, 1999). Bias measures how close the target is approximated by the prediction; variance reflects the variability of the prediction against changes in the training set; and noise measures the irreducible part of the error that results from the noise of classification itself. Bias and variance are major contributors to the error. An empirical study on bias–variance decomposition (Bauer & Kohavi, 1999) revealed that the error reduction for bagging is primarily due to variance reduction. Bagging only slightly affects (either increases or decreases) bias. However, the error reduction for AdaBoost is due to both variance and bias reduction. Thus, with a new set of samples, ensemble method can reduce the expected error, thereby increasing the accuracy of the model. The mechanisms of the benchmark models are briefly described as follows.

Logistic regression (LR): Although AI-based credit scoring models have become mainstream tools, LR is regarded as an industry standard and is widely applied in practice because of its simplicity and balanced error distribution (Lessmann et al., 2015). For a binary target variable (e.g., good or bad borrowers), the formula of LR can be expressed as follows:

$$\ln\left(\frac{F(x)}{1-F(x)}\right)=\beta_0+\sum_{i=1}^{n}\beta_i x_i, \tag{27}$$

where $F(x)$ denotes the probability prediction, β_0 is the constant coefficient, and β_i is the coefficient corresponding to the feature x_i. In other words, for a certain input (features), LR outputs the conditional probability of a sample that belongs to a specific class.

Neural network (NN): This AI-based technique was inspired by the structure of the human brain. NN typically consists of three or more layers: an input layer containing the input nodes, at least one hidden layer containing the weighted nodes, and an output layer that makes decisions on the input instances (West, 2000). Nodes in neighboring layers are linked, whereas those in the same layer are not. NN is started by pouring the data into the input layer. Then, the data pass through the network via nodes from the hidden layers to the output layer (Simon, 2008). The temporary output and its mean-of-squares error in the validation set are calculated. Thus, the results are returned to the hidden layer to adjust weights of nodes; this process is known as backpropagation. Finally, the optimized weights of nodes are determined, and the final decisions are realized.

Support vector machine (SVM): SVM is another AI-based technique used in credit scoring. The core of SVM is to map the original training set into a high-dimensional feature space, in which non-linear separated features are replaced by a linear discriminant function in high-dimension (Cortes & Vapnik, 1995). The function used to map the data is also called kernel function. Some popular kernel functions are linear, polynomial and radial basis function (RBF). Then, according to the structural risk minimization principle, SVM searches for a separating hyperplane to maximize the margin among different classes to ensure a sound generalization capability. SVM requires appropriate tuning of parameters (Huang et al., 2007), and tasks are handled by GS (Baesens et al., 2003).

Random forest (RF): RF is an ensemble tree-based method that utilizes bagging to generate diversified subsets of the entire training set to build individual trees. Then, during the splitting of each node, RF randomly selects a subset of features rather than consider all of the available features. In this way, given the same training samples, RF is likely to grow different trees, thereby offering additional diversity among trees. Tree pruning is prohibited in RF (Breiman, 2001).

Table 3
Confusion matrix.

Observed	Predicted		
	Good	Bad	
Good	TP	FN	TP+FN
Bad	FP	TN	FP+TN
	TP+FP	FN+TN	TP+FP+FN+TN

Some of the benchmark models require prior setting up of parameters. The base model used for tree-based models is the CART, which is further described in the Appendix. For NN, a multilayer perceptron (MLP) model is built with one hidden layer. In accordance with the suggestion of Ala'raj and Abbod (2016b), a small learning rate of 0.01 is set. The default maximum of epochs is set to 1000. The number of hidden neurons are tuned by GS based on the 10-fold cross-validation ACC and determined as 5, 9, 10, 4, and 7 for the German, Australian, Taiwan, P2P-A, and P2P-B dataset, respectively. A RBF kernel SVM trained with LIBSVM (Chang & Lin, 2011) is employed for SVM. For RBF-SVM, two hyper-parameters, i.e., C and γ, require tuning. In the present work, the optimal pair (C, γ) is determined by GS. Specifically, (32, 2), (32, 8), (0.5, 32), (32, 0.5), and (1, 0.1) are calculated as the optimal combinations of C and γ for the German, Australian, Taiwan, P2P-A, and P2P-B dataset, respectively. For bagging, AdaBoost, and RF, the number of iterations is fixed at 100. The hyper-parameters involved in GB slightly differ from those in XGBoost; the default setting in the scikit-learn package is adopted (Pedregosa et al., 2011).

4.3. Evaluation measures

An appropriate evaluation criteria must be selected to perform reasonable model comparisons. In credit scoring, one of the most popular evaluation measures is average accuracy (ACC), which is defined as the ratio of correctly classified samples to the test set. Using ACC cannot be considered the sole criterion because it cannot discriminate between good and bad applicants. Therefore, type I and type II error rates are also employed as performance measures. For credit scoring, a type I error occurs when bad applicants are misclassified as good, whereas a type II error appears if good applicants are misclassified as bad ones. Specifically, ACC, type I error, and type II error are defined as the confusion matrix in Table 3. In credit scoring, TP and TN represent the numbers of correctly classified good and bad borrowers, respectively. FP and FN denote the numbers of misclassified bad and good borrowers, respectively.

$$\mathrm{ACC}=\frac{TP+FN}{TP+FP+FN+TN} \tag{28}$$

$$\mathrm{Error\ I}=\frac{FP}{FP+TN} \tag{29}$$

$$\mathrm{Error\ II}=\frac{FN}{TP+FN} \tag{30}$$

These two types of error typically incur different misclassification costs. Given its high probability to cause huge loss, type I error is more costly than type II error.

Area under the curve (AUC) is an alternative discrimination capability measure based on the receiver operating characteristic (ROC) curve. The ROC curve is a complete sensitivity/specificity report for model evaluation. In the ROC curve, the true positive rate (TPR) is plotted against the false positive rate (FPR) at various cut-off values (Fig. 5), where TPR $=\frac{TP}{TP+FN}$ and FPR $=\frac{FP}{FP+TN}$. Each point

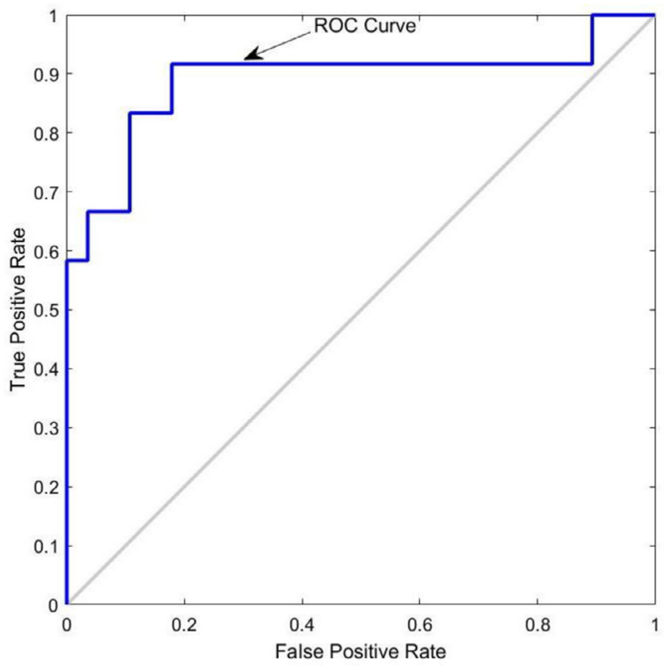

Fig. 5. Example of ROC curve.

on the ROC curve signifies a pair of TPR and FPR that corresponds to a certain threshold. The AUC measures how well the model distinguishes between the two classes. In other words, AUC indicates the probability that a randomly drawn positive case receives higher relief than a negative one. AUC ranges from 0 (no discriminative power) to 1 (perfect discriminative power).

Nevertheless, Hand (2009) stated that AUC is hindered by inherent drawbacks, among which the most critical was the assumption that the misclassification cost distribution was dependent on the various classifiers rather than on the datasets. In response, the AUC-H measure is proposed to overcome the disadvantages by applying a beta distribution to estimate a cost weight function that satisfies both requirement of objectivity (produce same results given same classifier and data) and of being independent of score distribution. The cost weight function can be expressed as follows:

$$\mathrm{u}(\mathrm{c}) \triangleq \mathrm{beta}(\mathrm{c}; \alpha, \beta) = \frac{\mathrm{c}^{\alpha-1}(1-\mathrm{c})^{\beta-1}}{B(1; \alpha, \beta)}, \tag{31}$$

where c is the cost ratio. $\alpha > 1$ and $\beta > 1$ are parameters in the beta distribution. Hand (2009) suggested that the suitable parameters are $\alpha = 2$ and $\beta = 2$. $B(1;\ \alpha,\ \beta)$ serves as a normalizing constant. In this way, a single cost weight distribution is acquired for all classifiers. Given an $\mathrm{u}_{\alpha,\ \beta}(\mathrm{c})$, the AUC-H measure can be obtained from a loss perspective as follows:

$$\begin{aligned} \mathrm{AUC-H} &= 1 - \frac{L_{\alpha,\beta}}{L_{Max}} \\ &= 1 - \frac{\int^{\mathrm{Q}} (T(c); b, c)\mathrm{u}_{\alpha,\beta}(c)dc}{\pi_0 \int_0^{\pi_1} c\mathrm{u}_{\alpha,\beta}(c)dc + \pi_1 \int_{\pi_1}^{1} (1-c)\mathrm{u}_{\alpha,\beta}(c)dc}, \end{aligned} \tag{32}$$

where $Q(T(c);\ b,\ c)$ is the overall loss for the cost pair (b, c). π_0 and π_1 are prior possibilities of classes 0 and 1, respectively. $L_{\alpha,\ \beta}$ can be interpreted as the general loss over the weight distribution $\mathrm{u}_{\alpha,\ \beta}(\mathrm{c})$, where as L_{Max} denotes the maximum loss results by the worst case (i.e., the class score distributions are the same). By this means, AUC-H provides a single-threshold distribution for all classifiers (Ala'raj & Abbod, 2016a). Thus, AUC-H is employed in this paper for evaluating the model.

Brier score (BS) is a score function that measures the accuracy of probabilistic prediction. BS ranges from 0 (perfect probabilistic prediction) to 1 (poor prediction). The formulation of BS is as follows:

$$\mathrm{BS} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2, \tag{33}$$

where N is the number of samples. p_i and y_i denote the probability prediction and true label of sample i, respectively.

Given that each evaluation measure has its merits and limitations, reliable comparisons can be realized through an integration of these criteria. The abovementioned measures may provide misleading results for the comparisons of the various models. To test the statistical significance of differences in performance, Harris (2015) and Twala (2010) used parametric tests, such as paired t-test, in the model comparisons, even though this approach was considered unsuitable, because the assumptions of parametric tests are likely to be violated in the comparisons of classification models (Demšar, 2006b). In our work, Friedman test, which is a rank-based non-parametric test, is employed to compare the different models. The statistic of Friedman test is computed as follows:

$$\chi_F^2 = \frac{12D}{K(K+1)}\left[\sum_{k=1}^{K} AvR_j^2 - \frac{K(K+1)^2}{4}\right], \tag{34}$$

where $AvR_j^2 = \frac{1}{D}\sum_{i=1}^{D} r_i^j$; D and K denote the numbers of the datasets and the classifiers, respectively. r_i^j is the averaged rank of classifier j on dataset i. If χ_F^2 is larger than a critical value, the null hypothesis (i.e., there is no difference among models) is rejected. For pair comparisons, post hoc test is applied to evaluate the differences among individual models. The post hoc test adopted in our experiment is the Nemenyi test, which indicates the difference when the averaged ranks differ by at least a critical difference (CD). CD is calculated as follows (Demšar, 2006a):

$$CD = q_{\alpha,\infty,\mathrm{k}}\sqrt{\frac{K(K+1)}{12D}}, \tag{35}$$

where $q_{\alpha,\ \infty,\ \mathrm{k}}$ is computed based on the t-test statistic, and D represents the number of datasets. To compare alternative models with the proposed XGBoost-based model, CDs after Bonferroni-Dunn adjustment is calculated at different significant levels beforehand, and the averaged ranks of the XGBoost-based model and the CD at different significant levels are denoted as border lines in the Nemenyi test diagram.

5. Experimental results

In this section, the proposed model is compared with various benchmark models in terms of five credit datasets. To simulate real-world credit scoring with the available datasets, the datasets are partitioned into two parts, i.e., training and test sets. The former is used to train models, and the latter is for evaluating the performances. In accordance with the common practice suggested by Ala'raj and Abbod (2016a), and Finlay (2011), 80%/20% training/testing partition was implemented; thus, 80% of the samples are utilized to build the model, and the remaining samples are used for evaluation. The experiments are looped 100 times to enhance their robustness, and the evaluation criteria for these experiments are averaged to alleviate the influence of individual partition. LR, NN, DT, SVM, bagging-NN, bagging-DT, AdaBoost-DT, AdaBoost-NN, and RF are performed on WEKA 3.6.12, with Matlab 2015a as API. GBDT and XGBoost are operated with Sklearn and XGBoost package on Python 2.7. Bayesian hyper-parameter optimization is performed using the hyperopt package for Python.

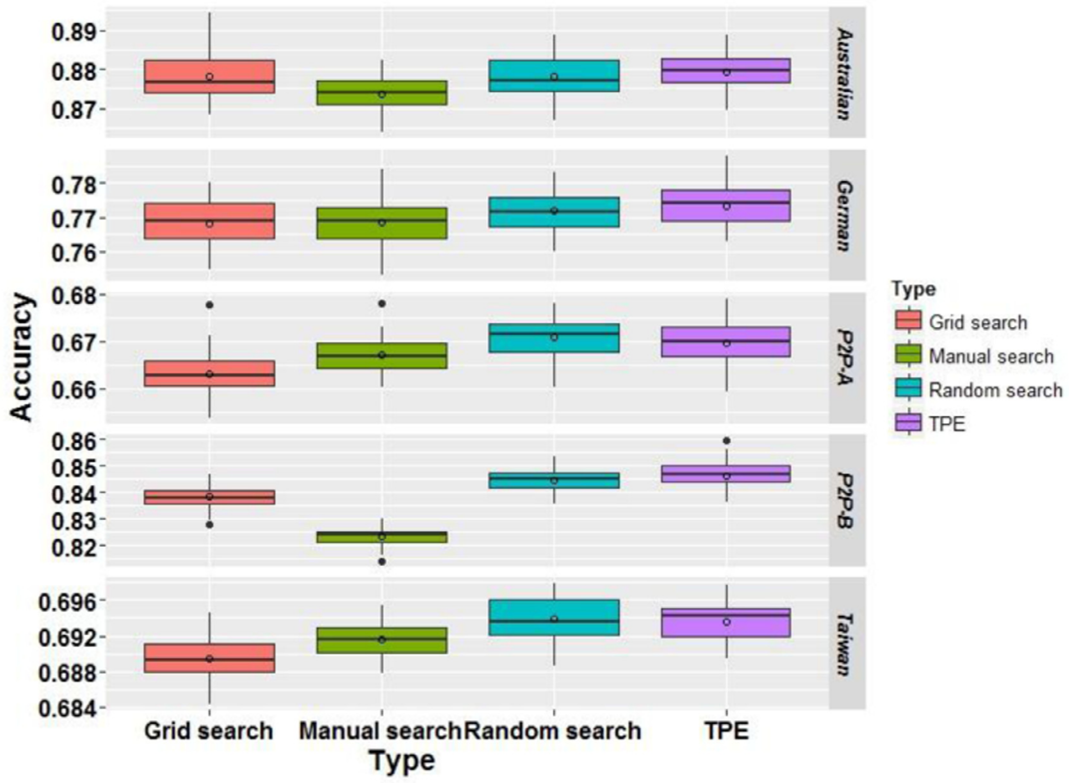

Fig. 6. Boxplot of accuracy over four hyper-parameter optimization approach.

All the experiments are performed on a desktop computer with 3.0 GHz Intel i5 CPU, 8GB RAM, and Windows 7 operating system.

5.1. Comparisons among hyper-parameter optimization methods

Learning algorithms used in credit scoring are seldom parameter-free. Thus, these hyper-parameters must be optimized to enhance the predictive capability of the algorithms. GS and MS are two popular strategies for hyper-parameter optimization in credit scoring. Bergstra and Bengio (2012) argued that RS, which is a random trial-and-error process, is more effectively than GS. In our paper, a Bayesian hyper-parameter optimization approach is employed, and the aforementioned hyper-parameter optimization methods are introduced as baselines.

The primary goal of the comparisons is to evaluate the effectiveness of the TPE hyper-parameter optimization approach. RS and TPE optimizer are iterative processes, and the number of iterations is set to 100 to achieve a good trade-off between accuracy and complexity. These hyper-parameter optimization methods are evaluated in terms of two aspects: ACC after optimization and speed of optimization. The details of the searching space are summarized in Table 1.

With regard to accuracy, Fig. 6 shows the boxplot of the ACC of different optimizers over five datasets. In the figure, the X-axis denotes the adopted methods, and the Y-axis represents the ACC. As shown in Fig. 6, TPE, which is a Bayesian hyper-parameter optimization approach, achieves a higher accuracy than the other approaches, except in the P2P-A dataset. Table 4 presents the results of a pairwise Wilcoxon rank sum test between TPE and the benchmark methods. If the p-value for the test is lower than a specific significance rate $\alpha = 0.1$, then the null hypothesis is rejected. Therefore, TPE performs significantly better than the corresponding methods. As shown in Table 4, TPE significantly outperforms GS and MS in most datasets but it significantly exceeds the accuracy of RS only in the P2P-B dataset. Combining these findings with the average ACC depicted in Fig. 6 suggests that TPE is marginally better than RS.

Table 4
Results of Wilcoxon rank sum test between TPE and benchmark methods for German, Australian, Taiwan, P2P-A and P2P-B dataset.

Method	*p*-value	Hypothesis ($\alpha = 0.1$)
German		
TPE vs. RS	0.1452	Not rejected
TPE vs. GS	0.0003	Rejected for TPE
TPE vs. MS	0.0004	Rejected for TPE
Australian		
TPE vs. RS	0.2146	Not rejected
TPE vs. GS	0.1429	Not rejected
TPE vs. MS	0.0001	Rejected for TPE
Taiwan		
TPE vs. RS	0.8173	Not rejected
TPE vs. GS	0.0001	Rejected for TPE
TPE vs. MS	0.0001	Rejected for TPE
P2P-A		
TPE vs. RS	0.3538	Not rejected
TPE vs. GS	0.0001	Rejected for TPE
TPE vs. MS	0.0011	Rejected for TPE
P2P-B		
TPE vs. RS	0.0704	Rejected for TPE
TPE vs. GS	0.0001	Rejected for TPE
TPE vs. MS	0.0001	Rejected for TPE

In Fig. 6, TPE also displays a low variability in the Austrian and P2P-B datasets. By contrast, TPE is average in the other datasets. The different variabilities of the TPE approach for different datasets must be further investigated. RS exhibits the second-best performance after TPE, achieving the second-highest accuracy (77.18%, 87.82%, 69.41%, 67.08%, and 84.47% for the German, Austrian, Taiwan, P2P-A, and P2P-B datasets, respectively) on average within 100 iterations.

The superiority of TPE over RS is due to its heuristic structure that exploits the information in existing trails. In addition, RS tends to be more unstable than TPE, partly because RS leads to greater randomness in the search for optimal parameters. GS is sub-optimal and inefficient. GS is slower than any RS methods and TPE optimizers, because the number of possible combinations increases with the number of hyper-parameters. GS may also fail due to the potential ignorance of optimal configurations as a result of limited alternatives on every parameter. The performance of GS can improve by adding enough candidates for each hyper-parameter. However, the number of possible combinations also increases exponentially and dramatically increases the computational costs. Under current configurations (21,600 loops under 10-fold cross-validation), accuracy is slightly improved but it is still worse than that of RS and TPE. With regard to MS, the performances of XGBoost under our manual parameter-settings is not satisfactory demonstrating worst performance among parameter optimization methods. This finding is consistent with Bergstra et al. (2011), which criticizes the difficulty of MS to reproduce results and the high dependence on expert knowledge. These drawbacks make MS undesirable for non-professional users.

In term of training time, MS spends the shortest time because it does not employ a trial-and-error method and uses only one group of parameter setting. The training time per sample (TTPS) of MS is 0.33 s. In striking contrast to MS, GS is the slowest with a TTPS of 6.27 s, implying that GS is unsuitable for large-scale credit datasets. The TTPS of TPE and RS is 1.35 s and 1.08 s, respectively. TPE is slower because the creation of Parzen estimators incurs additional computational costs. The increased computational costs might hinder the application of Bayesian hyper-parameter optimization, especially when computational feasibility is considered. Nonetheless, the Bayesian method is promising for the following three reasons.

First, among the existing methods, Bayesian hyper-parameter optimization approach is an advanced technique that has demonstrated better performances in several experiments when compared with traditional GS (Bergstra et al., 2011; Hutter et al., 2011; Snoek et al., 2012). The experimental results reported that Bayesian hyper-parameter optimization improved model performances within a fast convergence. Our experiment also suggests that Bayesian methods behave better than benchmark methods without excessive computational costs. A few generic optimization approaches, such as genetic algorithm and particle swarm optimization, have been applied for the hyper-parameter optimization of SVM-based credit scoring models (Huang et al., 2007; Lin, Ying, Chen, & Lee, 2008). However, these methods suffer from the curse of dimensionality (Chen, Montgomery, & Bolufé-Röhler, 2015).

Second, Bayesian hyper-parameter optimization is capable of parallel and distributed computing, suggesting that optimization can be realized with a cluster of computers (Bergstra, Yamins, & Cox, 2013). Moreover, XGBoost also supports multi-thread and distributed computing. These properties indicate that the proposed model is suitable for real-world applications.

Third, from a cost-benefit perspective, even one percentage of incorrect answer can result in a huge loss to financial institutions (Ala'raj & Abbod, 2016b). Thus, the ultimate goal of the study of credit scoring is to enhance the predictive capability of models even at the expense of large computational costs and loss of interpretability (Florez-Lopez & Ramon-Jeronimo, 2015). Bayesian hyper-parameter optimization is slow, and requires additional software and hardware investment to accelerate. Nonetheless, a well-tuned and accurate credit scoring model exhibits long-term benefits (Lessmann et al., 2015).

To summarize, the following conclusions can be drawn from the comparisons.

1. Bayesian optimization is an effective and fast method for hyper-parameter optimization. The heuristic structure enables the TPE approach to produce a promising parameter setting within a limited number of iterations.
2. The accuracy of RS is comparable to that of TPE and better than that of GS and MS.
3. Despite its popularity in tuning classifiers in credit scoring, GS exhibits worse performance than RS and TPE approach. Moreover, GS is slow, because all possible combinations of parameters are enumerated and evaluated.
4. MS is an inefficient method. The poor performance and the dependence on expert knowledge make MS an unsatisfactory method for hyper-parameter optimization in credit scoring.

5.2. Evaluation results

To validate our approach and perform reasonable comparisons with current credit scoring models, the proposed XGBoost-based credit scoring model is compared with various baseline models, including LR, NN, DT, SVM, bagging-DT, bagging-NN, AdaBoost, AdaBoost-NN, RF, and GBDT over five credit datasets. The results are summarized in Tables 5–9, which show that the proposed XGBoost-based credit scoring model achieves promising performances across the five credit datasets. The best result for each measure is in bold.

For the German dataset, the XGBoost-TPE model achieves the best ACC (i.e., 77.35%) with 53.71% type I error and 9.35% type II error. The model outperforms the best single classifier (LR) by 0.91% and the best ensemble model (GBDT) by 0.75%. The type I error rate is low in the XGBoost model, implying its advantageous capability to discriminate underlying bad applicants, which is better than its ability to discriminate good ones. The balanced error distribution can be explained by the introduction of parameter δ in XGBoost. The AUC-H measure of XGBoost-TPE is 0.3062, indicating a good discriminatory ability between good and bad applicants. The best baseline model in terms of the AUC-H measure is GBDT, with a score of 0.2929. In term of BS, XGBoost-TPE also achieves the best capability of probability prediction, with a score of 0.0935. The second-lowest baseline model is LR. In general, GBDT outperforms the other baseline models over most evaluation measures, indicating that GBDT is an acceptable alternative in credit scoring.

For the Austrian dataset, XGBoost-TPE improves the accuracy of the baseline single model (LR) and ensemble model (RF) by 1.15% and 0.51%, respectively. The error distribution is balanced, as the number of default and paid samples in the Austrian dataset is remarkably close. Thus, the error rate is relatively balanced. The AUC-H measure verifies that the XGBoost-TPE is better at handling cost assumptions among classes, achieving an AUC-H of 0.6571, which outperforms RF (best baseline models) by 0.0068. In terms of BS, the best baseline model is RF, with a score of 0.0971. However, RF is still worse than XGBoost-TPE, which achieves a BS of 0.0890. In sum, RF is a close alternative to XGBoost-TPE on average, whereas LR is an alternative among traditional single models.

For the Taiwan dataset, XGBoost-TPE achieves an accuracy of 69.36%, outperforming the best single model (LR) and ensemble model (GBDT) by 6.69% and 1.23%, respectively. Given that the dataset is balanced, the error distribution is fairly equal for all the models. XGBoost-TPE scores the second lowest type-I error rate (23.46%), indicating its sound capability to discriminate potential default borrowers. XGBoost-TPE scores an AUC-H of 0.2511, outperforming the baseline models. The best BS (0.2003) is achieved by XGBoost-RS, followed by XGBoost-TPE, which scores 0.2004. The GBDT results suggest that it is a powerful challenger to XGBoost-TPE, although the latter dominates the evaluation criteria over the Taiwan dataset.

In sum, in terms of the three UCI credit scoring datasets, XGBoost-TPE outperforms most of the baseline models, although the improvements of other XGBoost-based models are not notable.

Table 5
Results on German dataset over performance measures.

Model	ACC (%)	Type I error (%)	Type II error (%)	AUC–H	Brier score
AdaBoost	73.61	56.57	13.46	0.2148	0.1779
AdaBoost-NN	73.75	49.95	16.09	0.2317	0.2237
Bagging-DT	75.19	58.19	10.51	0.2232	0.1718
Bagging-NN	76.01	**49.67**	12.98	0.2674	0.1738
DT	72.65	59.37	13.63	0.1291	0.2257
LR	76.43	52.26	11.28	0.2812	0.1627
NN	72.54	50.13	17.75	0.2289	0.2363
RF	75.92	58.59	9.29	0.2664	0.1646
SVM	76.07	63.73	**6.88**	0.2711	0.1631
GBDT	76.59	51.37	11.42	0.2929	0.1615
XGBoost-MS	76.85	52.02	10.77	0.2974	0.1655
XGBoost-GS	76.83	49.79	11.76	0.2977	0.1176
XGBoost-RS	77.18	53.73	9.57	0.3030	0.0957
XGBoost-TPE	**77.34**	53.71	9.35	**0.3062**	**0.0935**

Table 6
Results on Australian dataset over performance measures.

Model	ACC (%)	Type I error (%)	Type II error (%)	AUC–H	Brier score
AdaBoost	85.64	17.40	11.93	0.6202	0.1034
AdaBoost-NN	84.59	14.22	16.36	0.6084	0.1174
Bagging-DT	86.42	13.33	13.78	0.6300	0.0987
Bagging-NN	85.62	11.83	16.42	0.6182	0.1062
DT	84.51	17.41	13.95	0.6065	0.1359
LR	86.77	**8.67**	16.88	0.6350	0.1019
NN	85.27	12.06	16.87	0.6191	0.1111
RF	87.41	13.26	12.05	0.6503	0.0971
SVM	85.54	15.35	13.74	0.6253	0.1012
GBDT	86.14	13.43	14.19	0.6288	0.0991
XGBoost-MS	87.38	13.67	11.79	0.6470	0.0908
XGBoost-GS	87.81	13.92	**10.80**	0.6557	0.0915
XGBoost-RS	87.82	12.64	11.82	0.6559	0.0893
XGBoost-TPE	**87.92**	12.67	11.61	**0.6571**	**0.0890**

Table 7
Results on Taiwan dataset over performance measures.

Model	ACC (%)	Type I error (%)	Type II error (%)	AUC-H	Brier score
AdaBoost	64.08	31.35	40.48	0.1934	0.2232
AdaBoost-NN	57.28	39.41	46.04	0.1691	0.2379
Bagging-DT	65.76	30.47	38.01	0.1936	0.2162
Bagging-NN	60.53	43.92	35.03	0.1982	0.2355
DT	61.65	25.86	50.84	0.1178	0.2493
LR	62.67	28.94	45.72	0.1842	0.2301
NN	56.68	72.36	**14.29**	0.1785	0.2477
RF	66.64	27.64	39.09	0.2048	0.2120
SVM	59.14	35.87	45.86	0.1744	0.2333
GBDT	68.13	26.86	36.89	0.2367	0.2093
XGBoost-MS	69.15	**20.35**	41.34	0.2475	0.2021
XGBoost-GS	68.95	25.11	36.99	0.2429	0.2037
XGBoost-RS	69.35	27.34	37.89	0.2510	**0.2003**
XGBoost-TPE	**69.36**	23.46	37.81	**0.2511**	0.2004

Table 8
Results on P2P-A dataset over performance measures.

Model	ACC (%)	Type I error (%)	Type II error (%)	AUC-H	Brier score
AdaBoost	61.25	40.18	37.32	0.0869	0.2336
AdaBoost-NN	64.09	33.61	38.22	0.1124	0.2251
Bagging-DT	62.43	37.43	37.71	0.1110	0.2328
Bagging-NN	65.34	34.07	35.25	0.1426	0.2198
DT	60.11	46.03	33.74	0.0572	0.2549
LR	64.74	41.37	**29.14**	0.1263	0.2247
NN	63.65	32.22	40.49	0.1284	0.2279
RF	63.20	35.72	37.88	0.1168	0.2277
SVM	60.67	41.29	37.36	0.1023	0.2331
GBDT	66.25	30.90	36.59	0.2120	0.2166
XGBoost-MS	66.70	**28.95**	37.64	0.2176	0.2125
XGBoost-GS	66.31	31.80	35.58	0.2129	0.2143
XGBoost-RS	**67.08**	29.78	36.06	**0.2358**	0.2096
XGBoost-TPE	66.97	29.82	36.23	0.2356	**0.2095**

Table 9
Results on P2P-B dataset over performance measures.

Model	ACC (%)	Type I error (%)	Type II error (%)	AUC-H	Brier score
AdaBoost	75.75	94.99	**1.22**	0.1139	0.1728
AdaBoost-NN	75.25	64.79	11.72	0.1689	0.1703
Bagging-DT	79.42	66.87	5.51	0.2580	0.1465
Bagging-NN	75.83	76.99	6.97	0.1955	0.1504
DT	76.01	63.27	11.21	0.1375	0.1941
LR	75.07	92.30	2.99	0.1603	0.1637
NN	75.40	69.01	10.14	0.1429	0.1599
RF	80.59	60.49	6.03	0.3098	0.1355
SVM	75.17	97.18	1.27	0.1418	0.1813
GBDT	83.52	42.20	8.10	0.5176	0.1212
XGBoost-MS	82.35	52.10	6.43	0.4917	0.1170
XGBoost-GS	83.83	41.19	8.03	0.5005	0.1105
XGBoost-RS	84.46	40.57	7.39	0.5283	0.1070
XGBoost-TPE	**84.65**	**39.68**	7.43	**0.5321**	**0.1067**

XGBoost-TPE performs well in imbalanced datasets, and is hardly influenced by the overwhelming majority samples. The experimental results also indicate that LR remains a competitor in credit scoring because of its simplicity and efficiency. Other single models, such as DT, SVM, and NN perform poorly, implying their unsuitability for credit scoring. For ensemble models, RF and GBDT are close challengers to XGBoost-TPE. GBDT is the best baseline model in the German and Taiwan datasets, whereas RF is the best in the Austrian dataset. The superiority of RF is consistent with the findings of Ala'raj and Abbod (2016a). However, GBDT has received limited attention in existing literature.

In the balanced P2P-A dataset, the XGBoost-based models exhibit nearly the same performances. In terms of ACC, XGBoost-RS outperforms XGBoost-TPE (in second place) by 0.11%. The best single and ensemble models in terms of ACC are LR and GBDT, respectively. XGBoost-TPE achieves the third lowest type I error rate below that of XGBoost-MS and XGBoost-RS, still performing better than the other baseline models. In terms of AUC-H, XGBoost-TPE outperforms the best single (LR) and ensemble (bagging-NN) models. DT and AdaBoost achieve comparatively low AUC-H. In terms of BS, XGBoost-TPE outperforms the other models but is only slightly better than XGBoost-RS by 0.0001. Single models are inferior to the other models over the five criteria. This finding agrees with that of Lessmann et al. (2015), and Ala'raj and Abbod (2016a), and is the direct motivation to apply ensemble methods to credit scoring.

In the imbalanced P2P-B dataset, the ACC of XGBoost-TPE reaches 84.65%, which is 8.64% and 1.13% better than DT (best single model) and GBDT (best ensemble model). XGBoost-based models are hardly affected by the imbalanced distribution, because they achieve acceptable type I error rates. However, several traditional baseline models (i.e., AdaBoost and SVM) tend to ignore the minority class, leading to high type I error rates. XGBoost-TPE displays better separation ability than all of the other models, with an AUC−H measure of 0.5321. XGBoost-TPE also achieves the highest BS, exhibiting better performance than the other baseline models.

To summarize, the following conclusions can be drawn from the experimental results:

1. XGBoost-based models are efficient and accurate approaches across multiple evaluation measures and datasets. These models also perform well in imbalanced credit datasets.
2. GBDT and RF are powerful classifiers in credit scoring, outperforming other ensemble and single classifiers.
3. LR remains a competitive classifier in credit scoring, and other single models have not achieved an acceptable performance.

5.3. Statistical significance test

Non-parametric Friedman test and Nemenyi test are employed to test the statistical significance of the differences in performance.

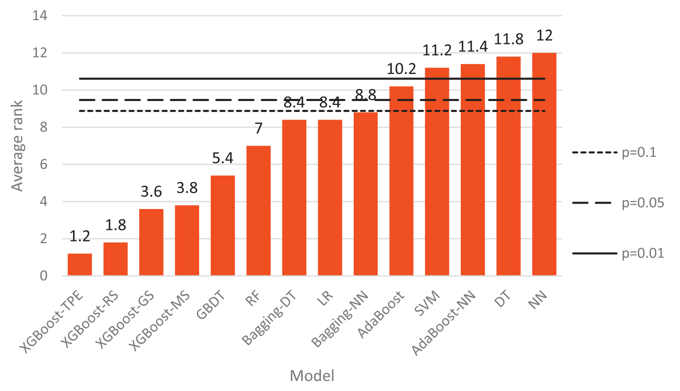

Fig. 7. Average rank of models and the threshold of Nemenyi test with $p = 0.1$, $p = 0.05$ and $p = 0.01$.

First, Friedman test statistic is computed (Eq. (34)) to verify if the models exhibit differing performances. When the null hypothesis of Friedman test is rejected, post-hoc Nemenyi test is used to perform paired comparisons. Critical values are calculated at several significance levels (i.e., 0.01, 0.05, and 0.1). The sum of the corresponding critical value and the lowest rank represent the thresholds at different significance levels. In this paper, the models are compared in terms of ACC. Friedman test statistic is computed as 47.45, rejecting the null hypothesis at a 99% significance level. The critical values of the Nemenyi test are calculated according to Eq. (35); $q_{0.1}$, $q_{0.05}$, and $q_{0.01}$ are calculated as 9.41, 8.26, and 7.68, respectively. Therefore, the CDs at significance levels of 0.01, 0.05, and 0.1 are computed as 10.61, 9.46, and 8.88, respectively. In Fig. 7, the bars represent the average rank of the models over five datasets. The three horizontal lines denote the threshold at different significant levels. As shown in Fig. 7, XGBoost-based models occupy the top four spots among all models. XGBoost-TPE has the lowest average rank. Although GBDT and RF are behind XGBoost-based models, they still achieve considerable performances over the five datasets. LR is at the eighth place and is the best single model. When XGBoost-TPE is used as the benchmark, bagging-NN, AdaBoost, SVM, AdaBoost-NN, DT, and NN have average ranks that are higher than at least one threshold, suggesting that they are significantly worse than the XGBoost-TPE approach at a significance level of 0.1.

5.4. Interpretability

Aside from accuracy, interpretability is another vital component in credit scoring. First, comprehensive credit scoring models clarify the decision process to managers, suggesting that feasible improvements can be targeted toward expert knowledge (Hand, 2006). Second, managers remain reluctant toward ensemble credit

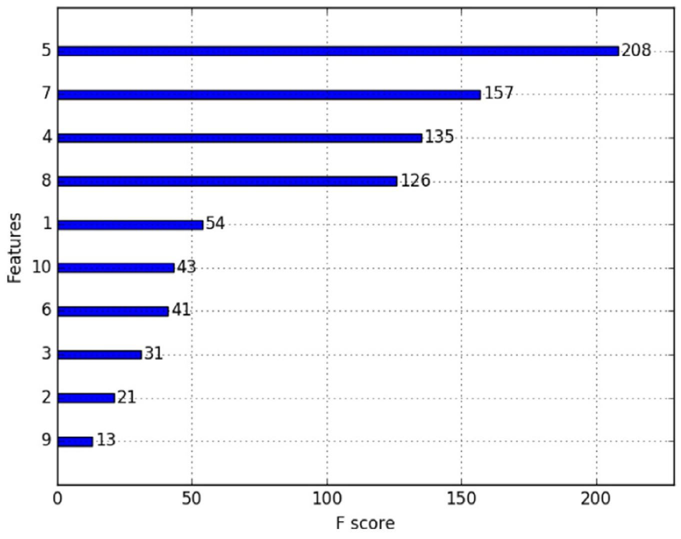

Fig. 8. Diagram of feature importance scores.

models that combine the decisions of various base models. Models with sound interpretability can overcome the reluctance of replacing traditional statistical methods, such as LDA and LR, with complex but powerful ensemble approaches (Finlay, 2011; Lessmann et al., 2015). Finally, many countries require comprehensive models for financial supervision (Finlay, 2011). However, interpretability tends to be ignored in many studies for two reasons. First, popular base models (e.g., SVM) are inherent black-box systems. Second, ensemble methods, such as bagging, boosting, and stacking, also develop problems in establishing comprehensive credit scoring models.

As tree-boosting models, the proposed XGBoost-based credit scoring models are constituted of regression tree as base models. The inherent interpretability of decision trees reduces the complexity of the models, thus enhancing the interpretability of the entire model. The interpretability of the XGBoost-based model mainly lies on two aspects: feature importance score and decision rules.

For example, in the Austrian dataset, the entire dataset is split 90%/10% into the training/ testing sets. The training set is used to train the XGBoost-TPE model. To ensure interpretability, the data are not normalized to the range from 0 to 1. During the feature selection, 10 features remain: $x_9, x_{10}, x_{11}, x_{15}, x_{14}, x_4, x_3, x_8, x_{13}$, and x_7. These features are sorted in descending order of their relative importance scores. These features are relabeled as X_1 to X_{10}. Afterward, the XGBoost model is trained with the optimal hyper-parameter setting derived by TPE. The feature importance scores (F scores) are computed by weight method, as shown in Fig. 8. A higher F score implies that the corresponding feature is more important. Therefore, X_5 is valuable and should be highlighted in the collection of credit data. By contrast, X_9 is hardly a necessary feature in the model and may be removed to accelerate learning.

Sixty base models (i.e., CARTs) are generated, as depicted in Fig. 9. The final model is obtained by adding the 60 CARTs. Specifically, given a new sample, the prediction of the observation is computed by adding the corresponding leaf scores to all trees. Then, the posterior probability is achieved by logit transformation of the sum. In this way, the XGBoost-based model loses the simple interpretability, because the final model is a sum of trees; nevertheless, the use of CART as the base model ensures the interpretability of the model, especially when compared with bagging or MCS structure with a black-box system (i.e., SVM or NN) as the base model. The small trees that grow in each iteration makes our proposal more comprehensive than RF, which uses fully grown DTs for each base model. Thus, the decision chart provides managers with explicit and comprehensive decision rules as to whether to extend credit to an applicant or not. The decision chart (Fig. 9) is a useful tool in real credit scoring situations.

6. Conclusion and future work

Recent reviews of the modeling methods in credit scoring have revealed that ensemble methods have become the popular modeling techniques (Lessmann et al., 2015). Unlike the extensive study of parallel learning methods, sequential approaches have received limited research attention. This paper explores the sequential methods for establishing credit scoring models. XGBoost is introduced as a novel variant of boosting technique, which adds a regularization term in the loss function and makes some engineering modifications based on GBDT. However, XGBoost includes various hyper-parameters and tends to fail without careful tuning. To prevent this situation, a TPE, which is a Bayesian hyper-parameter optimization method, is employed to tune these hyper-parameters.

The proposed XGBoost-based credit scoring model comprises three steps: data preprocessing, data scaling, and marking the missing values. Then, a model-based feature selection technique is applied to remove the redundant variables, thereby improving the performance and decreasing computational costs. Finally, the hyper-parameters are optimized by TPE approach, and the final model is trained with the obtained setting.

The proposed XGBoost-based method is validated over five real-world credit scoring datasets using five evaluation measures. TPE outperforms RS, GS, and MS, even though the latter two are commonly used measures in parameter-tuning. The comparisons with baseline models show the superiority of the XGBoost-based model in terms of predictive performance. The performance of the proposed model is also comparable to that of other state-of-the-art parallel learning credit scoring techniques (Ala'raj & Abbod, 2016a; Florez-Lopez & Ramon-Jeronimo, 2015). The XGBoost-based model is interpretative and comprehensive. The F score and the decision chart are efficient tools in real credit scoring situations.

Suggestions for future work are listed as follows:

1. Combine the XGBoost-based model with MCS to generate a more powerful classifier.
2. Consider exponential loss, hinge loss, squared loss, and other custom loss functions to examine how these loss functions perform.
3. Replace the original loss function with a cost-sensitive one to realize a cost-sensitive XGBoost, which is close to the real-world cost distribution in credit scoring.
4. Apply other novel hyper-parameter optimization approaches to credit scoring and make a comprehensive comparison.

Acknowledgement(s)

We thank the anonymous reviewers who have helped to improve the paper.

Appendix A. Classification and regression tree (CART)

Classification and regression tree (CART) is a top-down, tree-structured decision tree learning technique that produces either a classification or regression tree, depending on the type of response variable (Breiman, Friedman, Stone, & Olshen, 1984). Unlike conventional AI classification techniques that simultaneously place all available features into the model, CART has a flow-chart-like structure, wherein the recursive partitioning procedure begins at the topmost node (i.e., root node). The root node contains the entire feature, and the space is sequentially split into small and

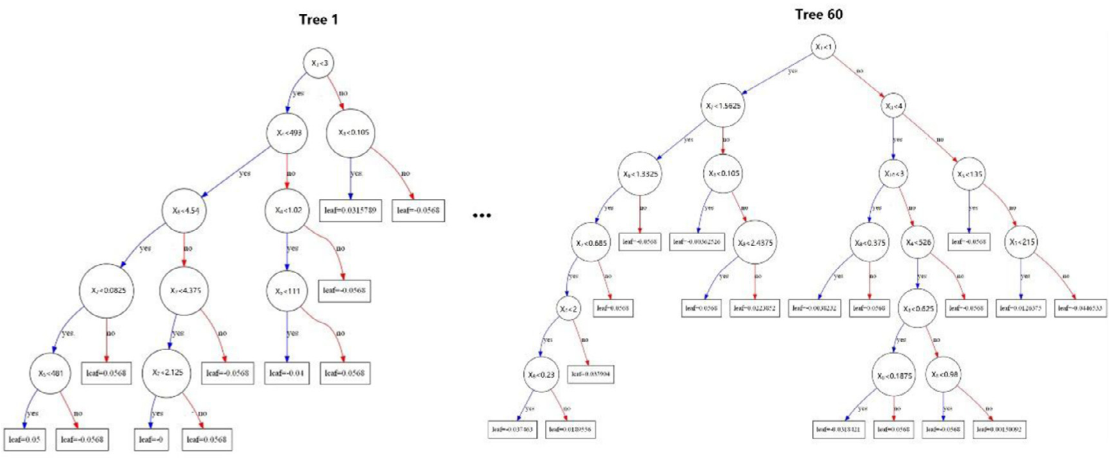

Fig. 9. Decision chart of XGBoost-based credit scoring model.

homogeneous subsets (i.e., nodes), which can be categorized into non-terminal nodes (nodes with child nodes) and terminal nodes (nodes without child nodes). Ideally, every node is composed of observations that belong to the same class. Given a dataset D = {**x**, y}, where ***x*** denote the features *y* denote the response variables, the issues of fitting a CART include the following:

(1) Selecting a certain split (i.e., which feature and the precise rule);
(2) Determining a terminal node (pruning); and
(3) Predicting each node.

The answers differ in classification and regression trees. The Gini impurity is commonly employed as the splitting criterion in classification trees to find the optimal splitting feature and the corresponding threshold (Breiman et al., 1984). Gini impurity is a measure of homogeneity of the target variable within the subsets under different predictors and critical values. For a binary classification problem, a certain split on node N_d generates a set of observations with K classes. Let f_i denote the fraction of observations with class i in this subset, where i ∈ {0, 1}. The Gini impurity can be computed as follows:

$$\mathrm{G}(N_d) = \sum_{i=0}^{1} f_i \ (1 - f_i \ ) = \sum_{i=0}^{1} f_i - \sum_{i=0}^{1} f_i^2. \tag{36}$$

All possible splits can be considered, and the best split in node N_d that maximizes the node impurity is chosen. The gain function then measures the change in the Gini impurity after a splitting compared to the Gini impurity without split for node N_d:

$$\mathrm{g}(N_d) = \mathrm{G}(N_d) - (\mathrm{G}_L(N_d) + \mathrm{G}_R(N_d)), \tag{37}$$

where $\mathrm{G}_L(*)$ and $\mathrm{G}_R(*)$ represent the Gini impurity of the left and right child nodes, respectively. If substantial improvement in $\mathrm{g}(N_d)$ is not achieved, the classification tree stops partitioning on node N_d. Thus, the depth of the classification tree is controlled to avoid over-fitting. For each terminal node, the prediction follows a plurality rule: the class having the largest number of samples in the node is assigned as the output.

In the case of regression trees, the least squares criterion is used for splitting. For a given splitting feature j, the desired threshold minimizes

$$\sum_{i:x_j \le s} (y_i - \bar{c}_1)^2 + \sum_{i:x_j > s} (y_i - \bar{c}_2)^2, \tag{38}$$

where $\bar{c}_1$ and $\bar{c}_2$ are the averages of the response variables for the samples in the left and right child nodes, respectively. *s* is the splitting point. The regression tree enumerates all features and selects an optimal threshold. The sum of squared difference before and after splitting can be used to prune the trees. A cost-complexity measure is also employed as follows (Sutton, 2005):

$$\mathrm{C}_\alpha(T) = C(T) + \alpha|T|, \tag{39}$$

where *C*(*T*) is the sum of the squared difference of the response variable and the predicted values for every node in the fitted tree. Please note that *T* herein represents the whole tree and |*T*| is defined as the number of terminal nodes. α functions as a regularization parameter. Given any α, a tree is created to minimize the cost-complexity measure. Independent test samples or cross-validation is usually used to select a robust α. Lastly, the prediction of each node is the average of the response variable of the corresponding samples.

The feature importance of a certain feature *I*(*x*) is defined as the sum across all splits when x is used as a splitting feature:

$$I(x) = \sum_{t \in T} \Delta(S_x, t), \tag{40}$$

where $\Delta(\cdot)$ indicates the difference operator and *t* denotes a certain node in tree T, and S is the improvement (e.g., error reduction).

Moreover, tree-based hyper-parameters, i.e., maximum tree depth and minimum child weight, are discussed.

The maximum tree depth in a CART is the number of edges along the longest path from the root node to the farthest terminal node. As shown in Fig. 9, the maximum tree depths for trees 1 and 60 are 5 and 6, respectively.

Tree size is defined as the number of terminal nodes in a tree. As shown in Fig. 9, the tree size for trees 1 and 60 are 11 and 18, respectively.

The hyper-parameter minimum child weight indicates the minimum sum of instance weight required in a child node. In XGBoost, a base learner is represented by $\omega_{q(x)}$, q∈$\{1, 2, \ldots, \mathrm{T}\}$, where q indicates the decision rules (how nodes are split), and ω denotes the sample weights on the node. This hyper-parameter is similar to that for defining the minimum number of samples in each node. If the tree partition step results in a leaf node with the sum of weight less than the minimum child weight, then the splitting process will terminate further partitioning (Chen & Guestrin, 2016).

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.eswa.2017.02.017.

References

Ala'raj, M., & Abbod, M. F. (2016a). Classifiers consensus system approach for credit scoring. *Knowledge-Based Systems, 104*, 89–105.
Ala'raj, M., & Abbod, M. F. (2016b). A new hybrid ensemble credit scoring model based on classifiers consensus system approach. *Expert Systems with Applications, 64*, 36–55.
Altman, E. I. (1968). Financial ratios, discriminant analysis and the prediction of corporate bankruptcy. *The Journal of Finance, 23*, 589–609.
Baesens, B., Van Gestel, T., Viaene, S., Stepanova, M., Suykens, J., & Vanthienen, J. (2003). Benchmarking state-of-the-art classification algorithms for credit scoring. *Journal of the Operational Research Society, 54*, 627–635.
Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*, 105–139.
Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research, 13*, 281–305.
Bergstra, J., Yamins, D., & Cox, D. D. (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference* (pp. 13–20).
Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of advances in neural information processing systems* (pp. 2546–2554).
Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.
Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*, 123–140.
Breiman, L. (2001). Random forests. *Machine Learning, 45*, 5–32.
Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.
Brillante, L., Gaiotti, F., Lovat, L., Vincenzi, S., Giacosa, S., Torchio, F., et al. (2015). Investigating the use of gradient boosting machine, random forest and their ensemble to predict skin flavonoid content from berry physical–mechanical characteristics in wine grapes. *Computers and Electronics in Agriculture, 117*, 186–193.
Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications, 39*, 3446–3453.
Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST), 2*, 27.
Chen, S., Montgomery, J., & Bolufé-Röhler, A. (2015). Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence, 42*, 514–526.
Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *arXiv preprint* arXiv:1603.02754.
Chen, T., & He, T. (2015). xgboost: EXtreme Gradient Boosting. *R package version 0.4-2*.
Chen, X., Zhou, L., & Wan, D. (2016). Group social capital and lending outcomes in the financial credit market: An empirical study of online peer-to-peer lending. *Electronic Commerce Research and Applications, 15*, 1–13.
Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*, 273–297.
Demšar, J. (2006a). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.
Demšar, J. (2006b). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research, 7*, 1–30.
Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning, 40*, 139–157.
Duin, R. P., & Tax, D. M. (2000). Experiments with classifier combining rules. In *International workshop on multiple classifier systems* (pp. 16–29).
Elith, J., Leathwick, J. R., & Hastie, T. (2008). A working guide to boosted regression trees. *Journal of Animal Ecology, 77*, 802–813.
Finlay, S. (2011). Multiple classifier architectures and their application to credit risk assessment. *European Journal of Operational Research, 210*, 368–378.
Florez-Lopez, R., & Ramon-Jeronimo, J. M. (2015). Enhancing accuracy and interpretability of ensemble strategies in credit risk assessment. A correlated-adjusted decision forest proposal. *Expert Systems with Applications, 42*, 5737–5753.
Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (pp. 23–37).
Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of statistics, 29*, 1189–1232.
Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis, 38*, 367–378.
Guelman, L. (2012). Gradient boosting trees for auto insurance loss cost modeling and prediction. *Expert Systems with Applications, 39*, 3659–3667.
Guo, Y., Zhou, W., Luo, C., Liu, C., & Xiong, H. (2016). Instance-based credit risk assessment for investment decisions in P2P lending. *European Journal of Operational Research, 249*, 417–426.
Hand, D. J. (2006). Classifier technology and the illusion of progress. *Statistical science, 21*, 1–14.
Hand, D. J. (2009). Measuring classifier performance: A coherent alternative to the area under the ROC curve. *Machine learning, 77*, 103–123.
Hand, D. J., & Henley, W. E. (1997). Statistical classification methods in consumer credit scoring: A review. *Journal of the Royal Statistical Society: Series A (Statistics in Society), 160*, 523–541.
Harris, T. (2015). Credit scoring using the clustered support vector machine. *Expert Systems with Applications, 42*, 741–750.
Hastie, T., Tibshirani, R., & Friedman, J. (2009). Boosting and additive trees. In *The elements of statistical learning* (pp. 337–387). Springer.
Huang, C.-L., Chen, M.-C., & Wang, C.-J. (2007). Credit scoring with a data mining approach based on support vector machines. *Expert Systems with Applications, 33*, 847–856.
Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization* (pp. 507–523).
Johnson, R., & Zhang, T. (2014). Learning nonlinear functions using regularized greedy forest. *IEEE transactions on pattern analysis and machine intelligence, 36*, 942–954.
Lee, T.-S., Chiu, C.-C., Chou, Y.-C., & Lu, C.-J. (2006). Mining the customer credit using classification and regression tree and multivariate adaptive regression splines. *Computational Statistics & Data Analysis, 50*, 1113–1130.
Lessmann, S., Baesens, B., Seow, H.-V., & Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research, 247*, 124–136.
Lin, S.-W., Ying, K.-C., Chen, S.-C., & Lee, Z.-J. (2008). Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert Systems with Applications, 35*, 1817–1824.
Min, J. H., & Lee, Y.-C. (2005). Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters. *Expert Systems with Applications, 28*, 603–614.
Nanni, L., & Lumini, A. (2009). An experimental comparison of ensemble of classifiers for bankruptcy prediction and credit scoring. *Expert Systems with Applications, 36*, 3028–3033.
Nascimento, D. S., Coelho, A. L., & Canuto, A. M. (2014). Integrating complementary techniques for promoting diversity in classifier ensembles: A systematic study. *Neurocomputing, 138*, 347–357.
Nie, G., Rowe, W., Zhang, L., Tian, Y., & Shi, Y. (2011). Credit card churn forecasting by logistic regression and decision tree. *Expert Systems with Applications, 38*, 15273–15285.
Pötzsch, S., & Böhme, R. (2010). The role of soft information in trust building: Evidence from online social lending. In *International conference on trust and trustworthy computing* (pp. 381–395).
Paleologo, G., Elisseeff, A., & Antonini, G. (2010). Subagging for credit scoring models. *European Journal of Operational Research, 201*, 490–499.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine learning research, 12*, 2825–2830.
Serrano-Cinca, C., & Gutiérrez-Nieto, B. (2016). The use of profit scoring as an alternative to credit scoring systems in peer-to-peer (P2P) lending. *Decision Support Systems, 89*, 113–122.
Simon, H. (2008). *Neural networks : A comprehensive foundation*. New Delhi: Prentice-Hall of India.
Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951–2959). Curran Associates.
Sutton, C. D. (2005). 11-classification and regression trees, bagging, and boosting. *Handbook of Statistics, 24*, 303–329.
Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 847–855).
Tsai, C.-F., Hsu, Y.-F., & Yen, D. C. (2014). A comparative study of classifier ensembles for bankruptcy prediction. *Applied Soft Computing, 24*, 977–984.
Twala, B. (2010). Multiple classifier application to credit risk assessment. *Expert Systems with Applications, 37*, 3326–3336.
Wang, G., Hao, J., Ma, J., & Jiang, H. (2011). A comparative assessment of ensemble learning for credit scoring. *Expert Systems with Applications, 38*, 223–230.
Wang, G., Ma, J., Huang, L., & Xu, K. (2012). Two credit scoring models based on dual strategy ensemble trees. *Knowledge-Based Systems, 26*, 61–68.
West, D. (2000). Neural network credit scoring models. *Computers & operations research, 27*, 1131–1152.
Wiginton, J. C. (1980). A note on the comparison of logit and discriminant models of consumer credit behavior. *Journal of Financial and Quantitative Analysis, 15*, 757–770.
Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*, 67–82.
Wu, T.-C., & Hsu, M.-F. (2012). Credit risk assessment and decision making by a fusion approach. *Knowledge-Based Systems, 35*, 102–110.
Yeh, C.-C., Lin, F., & Hsu, C.-Y. (2012). A hybrid KMV model, random forests and rough set theory approach for credit rating. *Knowledge-Based Systems, 33*, 166–172.
Zhang, Y., & Haghani, A. (2015). A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies, 58*, 308–324.
Zhang, Y., Jia, H., Diao, Y., Hai, M., & Li, H. (2016). Research on Credit Scoring by Fusing Social Media Information in Online Peer-to-Peer Lending. *Procedia Computer Science, 91*, 168–174.
Zięba, M., Tomczak, S. K., & Tomczak, J. M. (2016). Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction. *Expert Systems with Applications, 58*, 93–101.