Received: 12 May 2022 | Revised: 14 November 2022 | Accepted: 4 January 2023

DOI: 10.1002/sam.11613

WILEY

RESEARCH ARTICLE

Adaptive boosting for ordinal target variables using neural networks

Insung Um[1,2] | **Geonseok Lee**[1] | **Kichun Lee**[1]

[1]Department of Industrial Engineering, Hanyang University, Seoul, Republic of Korea

[2]Data Science Team, Hyundai Mobis, Seoul, Republic of Korea

Correspondence
Kichun Lee, Department of Industrial Engineering, Hanyang University, Seoul, Republic of Korea.
Email: skylee@hanyang.ac.kr

Funding information
Korea Institute of Energy Technology Evaluation and Planning, Grant/Award Number: 20204010600090; National Research Foundation of Korea, Grant/Award Number: NRF-2018R1A5A7059549; Ministry of Education of the Republic of Korea

Abstract

Boosting has proven its superiority by increasing the diversity of base classifiers, mainly in various classification problems. In reality, target variables in classification often are formed by numerical variables, in possession of ordinal information. However, existing boosting algorithms for classification are unable to reflect such ordinal target variables, resulting in non-optimal solutions. In this paper, we propose a novel algorithm of ordinal encoding adaptive boosting (AdaBoost) using a multi-dimensional encoding scheme for ordinal target variables. Extending an original binary-class AdaBoost, the proposed algorithm is equipped with a multi-class exponential loss function. We show that it achieves the Bayes classifier and establishes forward stagewise additive modeling. We demonstrate the performance of the proposed algorithm with a base learner as a neural network. Our experiments show that it outperforms existing boosting algorithms in various ordinal datasets.

KEYWORDS

adaptive boosting, neural networks, ordinal classification

1 | INTRODUCTION

Adaptive boosting (AdaBoost) is one of the most powerful machine learning algorithms for learning classifiers using ensemble techniques. Basically, this method trains multiple base learners to classify different patterns and yields the final classifier as a linear combination of these base learners. Each base learner trains in an iterative way, focusing on misclassified data from its previous base learners. The most popular boosting method, called AdaBoost.M1, is built for binary classification [5]. This algorithm proved to be equivalent to forward stagewise additive modeling using a binary-class exponential loss function. Since then, numerous AdaBoost variants have been proposed to solve not only the multi-class classification problem, but also regression problems. For example, AdaBoost.MH [11] applied the AdaBoost algorithm to a multi-class problem by converting it into a multiple binary classification problem. Hastie et al. [8] proposed stagewise additive modeling using a multi-class exponential loss function (SAMME) that uses a multi-class classifier as the base learner and solves the classification directly without reducing it to multiple binary classification problems. They adopted an encoding scheme that assigns each class to a k-dimensional vector and proved that SAMME is equivalent to forward stagewise additive modeling. To solve a regression problem using AdaBoost, Freund, and Schapire [5] proposed AdaBoost.R, which applies AdaBoost by converting a regression problem into a binary classification problem. Subsequently, Drucker [3] developed AdaBoost.R2, a modification of AdaBoost.R involving three candidate loss functions.

Most of the AdaBoost studies focus on classification or regression problems, but there is little interest in ordinal classification problems. Ordinal classification is the problem of classifying labels in an ordered relationship. For example, classifying labels of Excellent, Good, Fair, Poor, and Very Poor on a Likert scale is quite frequent, in which the labels, though categorical, are ordered by nature. The use of a multi-class classifier for this problem leads to a non-optimal solution [7]. The first reason is that all misclassified observations are given the same loss. When "Excellent" is the correct answer, a larger penalty should be given when classifying as "Poor" than "Good" but multi-class classification loss function does not reflect this. The second is that when training a classification model, the order information of target variables is not used.

Indeed, AdaBoost variants exist, solving ordinal classification problems. Frank and Hall [4] proposed a method to convert K ordinal label classification problems into $K-1$ binary classification problems. This method was applied to AdaBoost using decision trees as base learners. There have been attempts to address ordinal classification directly without reducing it to multiple binary classification. Lin and Li [9] proposed AdaBoost.OR, which increases performance by applying the boosting technique to the existing cost-sensitive base ordinal ranking algorithm. Riccardi et al. [10] proposed a modified SAMME algorithm using a cost-sensitive method to address ordinal classification problems. They applied the misclassification cost to the errors and weights of the SAMME and used the extreme learning machine as a base learner.

In this paper, we propose an encoding scheme and an exponential loss function for the ordinal multi-class problem and solve this optimization problem with an AdaBoost approach. Like the SAMME algorithm [8], the proposed algorithm achieves the Bayes classifier and proved to be equivalent to forward stagewise additive modeling. We demonstrate the proposed boosting algorithm using 1-hidden-layer neural networks as a base learner. The proposed algorithm directly solves the ordinal problem without reducing it to $K-1$ binary classification problems.

2 | PRELIMINARY

2.1 | AdaBoost for binary classification

Briefly, we describe the AdaBoost algorithm for binary classification called AdaBoost.M1 [5], which is a popular boosting algorithm. The algorithm involves forward additive stagewise modeling using an exponential loss function. Suppose N observations consisting of pairs $(\boldsymbol{x}_i, y_i), i = 1, \dots, N$, of an input vector and a target variable, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. For binary classification, a base learner T produces one of two values $\{-1, 1\}$ based on the input vector $\boldsymbol{x}_i$. The boosting algorithm sequentially trains base learners $\left\{T^{(1)}, \dots, T^{(M)}\right\}$ and produces a final classifier T in the form of a linear combination of base learners for good classification performance. Each observation $(\boldsymbol{x}_i, y_i)$ is given a weight, w_i, initialized to $1/N$. As the iteration, m, progresses, the weights are modified so that the observations misclassified by the current base learner $T^{(m)}$ increase the weights. The updating of w_i uses a coefficient, $\alpha^{(m)}$, representing the base learner's performance, and the construction of a final classifier also uses $\alpha^{(m)}$ by a linear combination, $\sum_m \alpha^{(m)} T^{(m)}(\boldsymbol{x})$. The whole sequence of AdaBoost.M1 is shown in Algorithm 1:

2.2 | Neural networks for ordinal classification

Neural networks, a flexible and popular model of nonlinear mapping from inputs to outputs through nodes and layers, have shown successful performance in numerous prediction tasks. Among the properties the model possesses, noticeable properties for ordinal classification are that the model easily extends to multiple responses and embraces various loss functions. Usually, to set up a multi-class problem with a neural network, class label $c_i \in \{1, 2, \dots, K\}$ is encoded as a K-dimensional vector $\boldsymbol{y}_i = \left(y_{i,1}, \dots, y_{i,K}\right)^{\intercal}$, and the class prediction $\hat{\boldsymbol{y}}$ is given by $\boldsymbol{g} = \sigma(\boldsymbol{W}\boldsymbol{x})$, for a transfer function $\sigma(\cdot)$ and a layer's coefficient matrix, $\boldsymbol{W}$. Backpropagation reduces the difference between the output $\boldsymbol{g}_i = \left(g_{i,1}, \dots, g_{i,K}\right)^{\intercal} = \sigma\left(\boldsymbol{W}\boldsymbol{x}_i\right)$ and the encoding vector $\boldsymbol{y}_i$. In nominal classification, the most widely used encoding scheme is a one-hot encoding in which the c_i-th element of $\boldsymbol{y}_i$ is set to 1 and all the other elements to 0. In contrast, Cheng et al. [2] proposed the following encoding scheme for ordinal classification:

$$y_{i,\ell} = \begin{cases} 1, & \ell \le c_i, \\ 0, & \ell > c_i. \end{cases}$$

According to the encoding vector $\boldsymbol{y}_i = (1, 1, \dots, 0, 0)^{\intercal}$, the neural network model learns the weights so that output $g_{i,\ell}$ $(\ell \le c_i)$ is close to 1 and $g_{i,\ell}$ $(\ell > c_i)$ is close to 0. The two popular loss functions for classification and regression, respectively, in the neural network model are as follows:

Algorithm 1. AdaBoost.M1

Initialize the weights $w_i = \frac{1}{N},\ i = 1, 2, \cdots, N$

for $m = 1$ *to* M **do**

Find a classifier $T^{(m)}$

$$T^{(m)} = \underset{T}{\operatorname{argmin}} \sum_{i=1}^{N} w_i \mathbb{I}(T(\boldsymbol{x}_i) \neq y_i)$$

Compute

$$err^{(m)} = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(T(\boldsymbol{x}_i) \neq y_i)}{\sum_{i=1}^{N} w_i}$$

Compute

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

Set

$$w_i \leftarrow w_i \exp\left(\alpha^{(m)} \mathbb{I}(T^{(m)}(\boldsymbol{x}_i) \neq y_i)\right),\ \ i = 1, 2, \cdots, N$$

Normalization of the weights

$$w_i \leftarrow \frac{w_i}{\sum_{i=1}^{N} w_i},\ \ i = 1, 2, \cdots, N$$

end

Output

$$T(\boldsymbol{x}) = sign\left[\sum_{m=1}^{M} \alpha^{(m)} T^{(m)}(\boldsymbol{x})\right]$$

$$\text{cross entropy loss} = \sum_{\ell=1}^{K} \left(y_{i,\ell} \log g_{i,\ell} + \left(1 - y_{i,\ell}\right) \log\left(1 - g_{i,\ell}\right)\right),$$

$$\text{squared error loss} = \sum_{\ell=1}^{K} \left(y_{i,\ell} - g_{i,\ell}\right)^2.$$

3 | ORDINAL ENCODING FOR ADABOOST

For an ordinal classification problem, we propose a novel AdaBoost method using multi-class exponential loss. Let us suppose we are given training data $(\boldsymbol{x}_i, c_i), i = 1, \ldots, N$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is an input vector and $c_i \in \{1, 2, \ldots, K\}$ is an ordinal target variable with a strict ordinal relationship, $1 < 2 < \cdots < K$. We encode class label c_i as a K-dimensional vector $\boldsymbol{y}_i = \left(y_{i,1}, \ldots, y_{i,K}\right)^{\intercal}$ using the following encoding scheme: for $\ell = 1, \ldots, K$,

$$y_{i,\ell} = \begin{cases} 1, & c_i \leq \ell, \\ -1, & c_i > \ell. \end{cases} \tag{1}$$

The encoded vector contains $c_i - 1$ number of -1 with 1 starting at index c_i. We show an example of the encoding $\boldsymbol{y}_i$ in Table 1 when $K = 5$.

Let us also suppose that we are given a base learner $\boldsymbol{g} = (g_1, g_2, \ldots, g_K)^{\intercal}$ that maps the input vector $\boldsymbol{x}$ onto $\mathcal{Y}$, in which

$$\boldsymbol{g} : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \mathcal{Y},$$

and $\mathcal{Y}$ is a set containing K-dimensional vectors

TABLE 1 Example of ordinal target-variable encoding with $K = 5$

Class	$\mathbf{y}_i$ encoding				
$c_i = 1$	1	1	1	1	1
$c_i = 2$	−1	1	1	1	1
$c_i = 3$	−1	−1	1	1	1
$c_i = 4$	−1	−1	−1	1	1
$c_i = 5$	−1	−1	−1	−1	1

$$\mathcal{Y} = \begin{cases} c(1, 1, \dots, 1, 1)^\intercal, \\ (-1, 1, \dots, 1, 1)^\intercal, \\ \vdots \\ (-1, -1, \dots, -1, 1)^\intercal. \end{cases}$$

With the encoding above, the proposed algorithm, ordinal encoding AdaBoost, proceeds as follows in Algorithm 2, which differs from Algorithm 1 in weight. In Algorithm 1, the weight is a vector over instances. In Algorithm 2, however, the weight is an $N \times K$ matrix over instances and classes. Algorithm 2 is established on the basis of the encoding above and the corresponding multi-class loss.

In Section 3.1, we show that Algorithm 2 inherently induces the Bayes classifier, optimal in minimization of error rate. The induced classifier produces a prediction of one-half the log-odds of $P(C \le \ell \,|\, \boldsymbol{x})$, where class label $C \in \{1, 2, \dots, K\}$ is random. Moreover, in Section 3.2, we show that the proposed algorithm is equivalent to forward SAMME. Finally, in Section 3.3, we define a loss function for neural networks to satisfy the condition of base learner $\boldsymbol{g}$.

Algorithm 2. Ordinal encoding AdaBoost

1 Initialize the weights $w_{i,\ell} = \frac{1}{NK},\ i = 1, 2, \cdots, N,\ \ell = 1, 2, \cdots, K$

2 **for** $m = 1$ *to* M **do**

3 Optimize the parameter $\gamma^{(m)}$ of the base classifier $\boldsymbol{g}$

$$\gamma^{(m)} = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{\ell=1}^{K} w_{i,\ell} \mathbb{I}\left(g_\ell(\boldsymbol{x}_i; \gamma) \neq y_{i,\ell}\right)$$

4 Compute

$$err^{(m)} = \frac{\sum_{i=1}^{N} \sum_{\ell=1}^{K} w_{i,\ell} \mathbb{I}\left(g_\ell(\boldsymbol{x}_i; \gamma^{(m)}) \neq y_{i,\ell}\right)}{\sum_{i=1}^{N} \sum_{\ell=1}^{K} w_{i,\ell}}$$

5 Compute

$$\alpha^{(m)} = \frac{1}{2} \ln \frac{1 - err^{(m)}}{err^{(m)}}$$

6 Set

$$w_{i,\ell} \leftarrow w_{i,\ell} \exp\left(2\alpha^{(m)} \mathbb{I}\left(g_\ell(\boldsymbol{x}_i; \gamma^{(m)}) \neq y_{i,\ell}\right)\right),\ \ i = 1, 2, \cdots, N,\ \ell = 1, 2, \cdots, K$$

7 Normalization of the weights

$$w_{i,\ell} \leftarrow \frac{w_{i,\ell}}{\sum_{i=1}^{N} w_{i,\ell}},\ \ i = 1, 2, \cdots, N,\ \ell = 1, 2, \cdots, K$$

8 **end**

9 Output

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha^{(m)} \boldsymbol{g}(\boldsymbol{x}; \gamma^{(m)})$$

$$\hat{c} = \underset{\ell \in \{1, 2, \cdots, K\}}{\operatorname{argmax}} \frac{e^{2f_\ell(\boldsymbol{x})}}{1 + e^{2f_\ell(\boldsymbol{x})}} - \frac{e^{2f_{\ell-1}(\boldsymbol{x})}}{1 + e^{2f_{\ell-1}(\boldsymbol{x})}},$$

where

$$\frac{e^{2f_0(\boldsymbol{x})}}{1 + e^{2f_0(\boldsymbol{x})}} \approx 0, \text{ meaning } f_0(\boldsymbol{x}) \approx -\infty, \text{ by definition.}$$

3.1 | Ordinal encoding scheme and multi-class exponential loss

The exponential loss function used in the binary classification AdaBoost.M1 is as follows:

$$L(y_i, f_i) = \exp(-y_i f_i).$$

Extending the binary class exponential loss function, we define the following multi-class exponential loss,

$$L(\boldsymbol{y}_i, \boldsymbol{f}_i) = \sum_{\ell=1}^{K} \exp\left(-y_{i,\ell} f_{i,\ell}\right), \tag{2}$$

where $\boldsymbol{f}_i = (f_{i,1}, \ldots, f_{i,K})^{\intercal}$. Since the sum of convex functions is convex, the multi-class exponential loss is convex. Obviously, the loss is positive and symmetric in that $L(\boldsymbol{y}_i, \boldsymbol{f}_i) = L(\boldsymbol{f}_i, \boldsymbol{y}_i) > 0$. We consider $\boldsymbol{f}_i$ to satisfy the ordinal relationship $f_{i,1} \le f_{i,2} \le \cdots \le f_{i,K}$, which we will describe after the derivation of $\boldsymbol{f}_i$.

We solve the ordinal classification problem by finding $\boldsymbol{f}$ that minimizes the loss function of Equation (2) using the encoding of Equation (1). Given the loss form, we minimize the expectation of the loss function:

$$\boldsymbol{f}^* = \underset{\boldsymbol{f}}{\operatorname{argmin}}\, \mathbb{E}_{\boldsymbol{Y}|\boldsymbol{x}} \sum_{\ell=1}^{K} \exp(-Y_\ell f_\ell), \tag{3}$$

where $\boldsymbol{f}^*$ is the population minimizer.

We can rewrite Equation (3) as follows:

$$\begin{aligned}
&\left[\exp(-f_1) + \exp(-f_2) + \cdots + \exp(-f_K)\right] \\
&P(\boldsymbol{Y} = (1, 1, \cdots, 1, 1)^{\intercal} | \boldsymbol{x}) \\
&+ \left[\exp(f_1) + \exp(-f_2) + \cdots + \exp(-f_K)\right] \\
&P(\boldsymbol{Y} = (-1, 1, \cdots, 1, 1)^{\intercal} | \boldsymbol{x}) \\
&+ \cdots + \left[\exp(f_1) + \exp(f_2) + \cdots + \exp(-f_K)\right] \\
&P(\boldsymbol{Y} = (-1, -1, \cdots, -1, 1)^{\intercal} | \boldsymbol{x}).
\end{aligned}$$

K-dimensional vector $\boldsymbol{Y}$ is an encoding vector of class C, so the probability is rewritten using C: for example,

$$P(\boldsymbol{Y} = (1, 1, \ldots, 1, 1)^{\intercal} | \boldsymbol{x}) = P(C \le 1 | \boldsymbol{x}).$$

Using the cumulative distribution function of a random variable $C \in \{1, 2, \ldots, K\}$, we can simplify the equation as follows:

$$\begin{aligned}
&\exp(-f_1) P(C \le 1 | \boldsymbol{x}) + \exp(f_1)(1 - P(C \le 1 | \boldsymbol{x})) \\
&\quad + \cdots + \exp(-f_{K-1}) P(C \le K - 1 | \boldsymbol{x}) + \exp(f_{K-1}) \\
&\quad (1 - P(C \le K - 1 | \boldsymbol{x})) + \exp(-f_K)
\end{aligned}$$

Taking the derivatives with respect to $f_\ell, \ell = 1, \ldots, K-1$, we have

$$-\exp(-f_\ell) P(C \le \ell | \boldsymbol{x}) + \exp(f_\ell)(1 - P(C \le \ell | \boldsymbol{x})) = 0.$$

By solving this equation, we find a population minimizer,

$$\begin{aligned}
f_\ell^* &= \frac{1}{2} \ln \frac{P(C \le \ell | \boldsymbol{x})}{1 - P(C \le \ell | \boldsymbol{x})} \\
&= \frac{1}{2} \ln \frac{P(C \le \ell | \boldsymbol{x})}{P(C > \ell | \boldsymbol{x})}, \quad \ell = 1, 2, \ldots, K-1.
\end{aligned}$$

Equivalently, this can be rewritten as

$$P(C \le \ell | \boldsymbol{x}) = \frac{e^{2 f_\ell^*}}{1 + e^{2 f_\ell^*}}, \quad \ell = 1, \ldots, K-1.$$

Thus, use of the proposed encoding scheme and the multi-class exponential loss lead to a vector estimation where each element f_ℓ^* is one-half the log-odds of $P(C \le \ell | \boldsymbol{x})$. Since $P(C \le \ell | \boldsymbol{x}) \le P(C \le \ell + 1 | \boldsymbol{x})$, it is obvious that $f_\ell^* \le f_{\ell+1}^*$, so we observe $\boldsymbol{f}_i$ that satisfies $f_{i,1} \le f_{i,2} \le \cdots \le f_{i,K}$. Once $P(C \le \ell | \boldsymbol{x})$ is obtained by f_ℓ^*, the Bayes classifier, $\operatorname{argmax}_\ell P(C = \ell | \boldsymbol{x})$, easily derives from its difference:

$$\begin{aligned}
\hat{c} &= \underset{\ell \in \{1, 2, \ldots, K\}}{\operatorname{argmax}} P(C = \ell | \boldsymbol{x}) \\
&= \underset{\ell \in \{1, 2, \ldots, K\}}{\operatorname{argmax}} \{P(C \le \ell | \boldsymbol{x}) - P(C \le \ell - 1 | \boldsymbol{x})\} \\
&= \underset{\ell \in \{1, 2, \ldots, K\}}{\operatorname{argmax}} \left\{ \frac{e^{2 f_\ell^*}}{1 + e^{2 f_\ell^*}} - \frac{e^{2 f_{\ell-1}^*}}{1 + e^{2 f_{\ell-1}^*}} \right\},
\end{aligned}$$

where we define $e^{2 f_0^*} = 0$ and $\frac{e^{2 f_K^*}}{1 + e^{2 f_K^*}} = 1$. When $K = 2$, the above classifier becomes $\hat{c} = 1$ if $\log \frac{e^{2 f_1^*}}{1 + e^{2 f_1^*}} > 1/2$, which reduces to $P(C = 1 | \boldsymbol{x}) > P(C = 2 | \boldsymbol{x})$, and $\hat{c} = 2$ otherwise. This case is the same as a Bayesian classifier for non-ordinal target variables.

In particular, when $\boldsymbol{f}_i = b\boldsymbol{v}$ and, for a constant $b > 0$, we can prove that the loss increases as the difference between true class c_i and predict class $\hat{c}$ increases. It is straightforward to rewrite the loss, $L = \sum_{\ell=1}^{K} \exp\left(-b y_\ell v_\ell\right)$, in (2) as follows: if predicted $\hat{c}$ is less than true target c, $\hat{c} < c$,

$$\begin{aligned}
L = &\exp\left(-b y_1 v_1\right) + \cdots + \exp\left(-b y_{\hat{c}} v_{\hat{c}}\right) + \cdots \\
&+ \exp\left(-b y_{c-1} v_{c-1}\right) + \exp\left(-b y_c v_c\right) + \cdots \\
&+ \exp\left(-b y_K v_K\right).
\end{aligned}$$

Since $y_1, \ldots, y_{c-1} = -1$ and $y_c, \ldots, y_K = 1$ according to our encoding scheme and, similarly, $v_1, \ldots, v_{\hat{c}-1} = -1$

TABLE 2 Exemplary loss values when predicted class $\hat{c}_i$ is 2 with $\boldsymbol{f}_i = (-2, 2, 2, 2, 2)^\intercal$

True class	Loss value
$c_i = 1$	$e^2 + 4e^{-2}$
$c_i = 2$	$5e^{-2}$
$c_i = 3$	$e^2 + 4e^{-2}$
$c_i = 4$	$2e^2 + 3e^{-2}$
$c_i = 5$	$3e^2 + 2e^{-2}$

and $v_{\hat{c}}, \dots, v_K = 1$, we can rewrite the equation as follows:

$$\begin{aligned}&\exp(-b(+1)) + \cdots + \exp(-b(-1)) + \cdots + \exp(-b(-1))\\ &\quad + \exp(-b(+1)) + \cdots + \exp(-b(+1))\\ &= (c - \hat{c})e^b + (K - (c - \hat{c}))e^{-b}\\ &= (c - \hat{c})\left(e^b - e^{-b}\right) + Ke^{-b}.\end{aligned}$$

In the same way, $L = (\hat{c} - c)\left(e^b - e^{-b}\right) + Ke^{-b}$ when $\hat{c} \geq c$. Thus, we prove the following:

$$L = |c - \hat{c}|\left(e^b - e^{-b}\right) + Ke^{-b} \propto |c - \hat{c}|.$$

The minimum is Ke^{-b} when $\boldsymbol{f}_i = b\boldsymbol{y}_i$, that is, $|c - \hat{c}| = 0$. Notice that the loss is in proportion to the difference between the true class label c and estimated label $\hat{c}$, which is equivalent to L_1 loss. To the best of our knowledge, no AdaBoost algorithm equipped with forward stagewise modeling has been proposed for classification tasks with such an L_1 loss form. We provide the associated forward stagewise modeling in the following section. For example, suppose the true class c_i is 2, $\boldsymbol{y}_i = (-1, 1, 1, 1, 1)^\intercal$. If the current prediction $\boldsymbol{f}_i$ is $(-2, 2, 2, 2, 2)^\intercal$, then the predicted class $\hat{c}_i$ is 2. The loss is $5e^{-2}$, which is the smallest. If $\boldsymbol{f}_i$ is $(-2, -2, -2, 2, 2)^\intercal$, equivalently $\hat{c} = 4$, then the loss becomes $2e^2 + 3e^{-2}$. For $\boldsymbol{f}_i = (-2, 2, 2, 2, 2)^\intercal$, the loss values according to the true class are shown in Table 2.

3.2 | Forward stagewise additive modeling

In this section, we show that Algorithm 2 provides the solution (3) using forward stagewise additive modeling, a method of sequentially searching and adding base classifiers to find a solution that minimizes loss. This is quite important in boosting since it serves as a theoretical basis for adding base classifiers to build a strong model. We show that the proposed loss for the ordinal classification task leads to a multi-dimensional extension of forward stagewise additive modeling. First, given the training input data $\boldsymbol{x}_i$ and encoded target vector $\boldsymbol{y}_i$, $i = 1, \dots, N$ we replace the expectation in Equation (3) with an average empirical loss,

$$\underset{\boldsymbol{f}}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{\ell=1}^{K} \exp\left(-y_{i,\ell} f_\ell(\boldsymbol{x}_i)\right).$$

In the m-th iteration of Algorithm 2, we optimize the parameter $\gamma^{(m)}$ of the base classifier $\boldsymbol{g}$ to reduce the loss and calculate a coefficient $\beta^{(m)}$ corresponding to this classifier. Then, we update the approximation $\boldsymbol{f}$ by adding it to the current fit $\boldsymbol{f}^{(m-1)}(\boldsymbol{x})$. Algorithm 3 below expounds the steps of the resulting forward stagewise additive modeling.

Therefore, to solve the multi-class exponential loss minimization problem using forward stagewise additive modeling, we attempt to find an addition parameter, β, in $f_\ell^{(m-1)}(\boldsymbol{x}_i) + \beta g_\ell(\boldsymbol{x}_i; \gamma)$ that minimizes the empirical multi-class exponential loss:

$$\begin{aligned}\beta^{(m)}, \gamma^{(m)} &= \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{\ell=1}^{K} \exp\left(-y_{i,\ell}\left(f_\ell^{(m-1)}(\boldsymbol{x}_i) + \beta g_\ell(\boldsymbol{x}_i; \gamma)\right)\right)\\ &= \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{\ell=1}^{K} w_{i,\ell} \exp\left(-\beta y_{i,\ell} g_\ell(\boldsymbol{x}_i; \gamma)\right), \quad (4)\end{aligned}$$

where $w_{i,\ell} = \exp\left(-y_{i,\ell} f_\ell^{(m-1)}(\boldsymbol{x}_i)\right)$.

The solution of (4) is

$$\begin{aligned}\gamma^{(m)} &= \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{\ell=1}^{K} w_{i,\ell}\mathbb{I}\left(g_\ell(\boldsymbol{x}_i; \gamma) \neq y_{i,\ell}\right),\\ \beta^{(m)} &= \frac{1}{2}\ln\frac{1 - \mathrm{err}^{(m)}}{\mathrm{err}^{(m)}}, \quad (5)\end{aligned}$$

where

$$\mathrm{err}^{(m)} = \frac{\sum_{i=1}^{N}\sum_{\ell=1}^{K} w_{i,\ell}\mathbb{I}\left(g_\ell\left(\boldsymbol{x}_i; \gamma^{(m)}\right) \neq y_{i,\ell}\right)}{\sum_{i=1}^{N}\sum_{\ell=1}^{K} w_{i,\ell}}.$$

Notice that the derivation of coefficients $\gamma^{(m)}$ depends on the choice of base model $g_\ell(\boldsymbol{x}_i; \gamma)$ to minimize the classification error incurred by the pre-existing weights $w_{i,\ell}$. Once $\gamma^{(m)}$ is obtained, its error, $\mathrm{err}^{(m)}$, is computed, and the associated $\beta^{(m)}$ is calculated accordingly. The only difference between the classical forward stagewise additive

Algorithm 3. Forward stagewise additive modeling

Initialize $\boldsymbol{f}^{(0)}(\boldsymbol{x}) = (0, 0, \cdots, 0)^{\intercal}$.
for $m = 1$ *to* M **do**
  Compute
$$\beta^{(m)}, \gamma^{(m)} = \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(\boldsymbol{y}_i, \boldsymbol{f}^{(m-1)}(\boldsymbol{x}_i) + \beta \boldsymbol{g}(\boldsymbol{x}_i; \gamma)\right)$$
  Set
$$\boldsymbol{f}^{(m)}(\boldsymbol{x}) = \boldsymbol{f}^{(m-1)}(\boldsymbol{x}) + \beta^{(m)} \boldsymbol{g}(\boldsymbol{x}; \gamma^{(m)})$$
end

modeling and the resulting one is that it should obtain a K?dimensional vector-form prediction, $\boldsymbol{g}\left(\boldsymbol{x}; \gamma^{(m)}\right)$ that estimates $\boldsymbol{f}^{*}(\boldsymbol{x})$ in Equation (3) and use the associated weights $w_{i,\ell}$. In the current research, we propose the use of neural networks as $\boldsymbol{g}$, described in the following section. Since the model is updated by $\boldsymbol{f}^{(m)}(\boldsymbol{x}) = \boldsymbol{f}^{(m-1)}(\boldsymbol{x}) + \beta^{(m)} \boldsymbol{g}\left(\boldsymbol{x}; \gamma^{(m)}\right)$, the weights for the next iteration are as follows:

$$w_{i,\ell} \leftarrow w_{i,\ell} \exp\left(-\beta^{(m)} y_{i,\ell} g_{\ell}\left(\boldsymbol{x}_i; \gamma^{(m)}\right)\right).$$

Overall, the proposed algorithm can be regarded as forward SAMME loss function.

3.3 | Neural network base classifier

In this section, we explain an implementation of a base classifier $\boldsymbol{g}$ using neural networks among many choices. Recall that base classifier $\boldsymbol{g}$ is obtained by minimizing overall classification error weighted by $w_{i,\ell}$ as given in Equation (5). The choice of $\boldsymbol{g}$ can be regularized logistic regression, support vector machine, and so forth. We present our choice based on neural networks because they are flexible to model multi-dimensional output, $\mathcal{Y}$. In addition, it brings an entrance of possible deep-layered neural networks for big data inputs. However, in this research, we varied the number of hidden nodes in one hidden layer. Now, we define a neural network model for mapping the input vector $\boldsymbol{x}$ onto set $\mathcal{Y}$ of size K as follows: for output layer $\boldsymbol{h} = (h_1, h_2, \ldots, h_K)^{\intercal} = \sigma\left(\boldsymbol{W}_2 \sigma\left(\boldsymbol{W}_1 \boldsymbol{x}\right)\right)$, we minimize

$$\gamma^{(m)} = \underset{\gamma}{\operatorname{argmin}} \left[\sum_{i=1}^{N} \sum_{\ell=1}^{K} w_{i,\ell} \left(h_{\ell}\left(\boldsymbol{x}_i; \gamma\right) - y_{i,\ell}\right)^2 + \sum_{i=1}^{N} \sum_{\ell=1}^{K-1} \max\left(0, h_{\ell}\left(\boldsymbol{x}_i \gamma\right) - h_{\ell+1}(\boldsymbol{x}_i; \gamma)\right) \right], \quad (6)$$

where a parameter, γ, is the weights and bias in the neural network model, $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. For one hidden layer, the size of parameter γ is $(a + 1)H + (H + 1)K$ times when the size of input $\boldsymbol{x}_i$ and the hidden layer are a and H, respectively. We vary dimension H from 8 to 64 for the hidden layer in the experiments. Notice that the second term is introduced to satisfy $y_1 \leq y_2 \leq \cdots \leq y_K$ since $\mathcal{Y}$ consists of encoding vectors with ordinal relations. In each iteration of the forward stagewise additive modeling of Algorithm 3, we numerically find $\gamma^{(m)}$ to predict the encoded $\boldsymbol{y}$ vectors using given $w_{i,\ell}$.

4 | EXPERIMENTS

In this section, we present the results of experiments performed to validate our ordinal encoding AdaBoost algorithm. In Section 4.1, we describe the ordinal classification datasets used in the validation. Section 4.2 explains measures to evaluate ordinal data classification algorithms. We provide details of selected ensemble-based algorithms for comparison with the proposed algorithms in Section 4.3. Moreover, we present a performance comparison and analysis of the proposed algorithm and other selected ensemble-based methods in Section 4.4. Finally, in Section 4.5, we compare the performance as the iteration increases to check the effectiveness of the proposed AdaBoost algorithm. Notice that our method is implemented based on PyTorch and is publicly available.[1]

4.1 | Ordinal classification datasets

Seven datasets selected from the UCI repository and one simulation dataset were used for the experiment, and the characteristics of the datasets are shown in Table 3. We

[1] https://github.com/in-sung/OEAB

TABLE 3 Details of benchmark datasets

Dataset	# of obs	Input dim (a)	# of class (K)	# of instances per class
Toy	500	2	5	57, 146, 120, 122, 55
Wine-quality	1599	11	6	10, 53, 681, 638, 199, 18
Car	1728	21	4	1210, 384, 69, 65
Balance-scale	625	4	3	288, 49, 288
Tae	151	56	3	49, 50, 52
Boston-house	506	13	5	77, 239, 123, 36, 31
Machine	199	6	10	107, 35, 21, 6, 8, 5, 3, 4, 4, 6
Stock	950	9	5	158, 227, 272, 207, 86

created the simulation dataset following the procedure described in Reference [1]: we generated 500 observations $\boldsymbol{x} = (x_1, x_2)^{\mathsf{T}}$, where x_1 and x_2 are independent and each follows a uniform distribution. A class label for each observation is assigned as follows:

$$c = \min_{\ell \in \{1,2,3,4,5\}} \{\ell | r_{\ell-1} < 10\,(x_1 - 0.5)\,(x_2 - 0.5) + \varepsilon < r_\ell\},$$
$$(r_0, r_1, r_2, r_3, r_4, r_5) = (-\infty, -1, -0.1, 0.25, 1, \infty),$$

where $\varepsilon \sim N\left(0, 0.125^2\right)$. Notice that the class labels c are ordered since the values of $r_i, i = 0, \ldots, 5$, are ordered.

4.2 | Performance measure for ordinal classification

We evaluated the ordinal classification model using accuracy and mean absolute error. From the viewpoint of classification, the accuracy measure evaluates whether the model predicts true class. In terms of regression, the mean absolute error is a measure to check the difference between prediction and truth. In addition, we considered the performance measure of F_1-score for multi-class classification to compare the overall ordinal class classification performance of the algorithms:

$$\text{Acc} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left(\hat{c}_i = c_i\right),$$
$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} |\hat{c}_i - c_i|,$$
$$F_1 = \frac{1}{K}\sum_{\ell=1}^{K} F_1 \text{ for class } \ell,$$
$$F_1 \text{ for class } \ell = \frac{2}{(\text{precision for class } \ell)^{-1} + (\text{recall for class } \ell)^{-1}}.$$

4.3 | Comparison methods

In the experiment, we compare the proposed method with the cost sensitive SAMME (cs-SAMME) [10] method for ordinal classification, specifically, the SAMME using the exponential loss in the form of a cost-weighted sum that increases the loss as the distance from the correct answer increases. In other words, it is calculated by additionally multiplying the update rule of the error estimation and of the pattern weights by the cost of misclassification. Table 4 shows an example of the cost matrix used in the cs-SAMME algorithm, and we use the absolute cost matrix for the experiment. Extreme learning machine was used as the base classifier in the original work [10]. In contrast, we adopted a one-hidden-layer neural network as the base classifier of cs-SAMME for comparison with the boosting algorithm derived by its flexibility and universal approximation theorem. In order to compare with the case where the characteristics of the ordinal target variable are not considered, the standard multi-class AdaBoost algorithm, SAMME [8], is compared. For the same reason, a one-hidden-layer neural network, a multi-class classifier with squared loss, was used as the base classifier of SAMME. In addition, we varied the number of hidden nodes to see the effect of the degree of parameterization in the follow-up experiments. We implement our method (OEAB), cs-SAMME [10], and SAMME [8]

TABLE 4 Cost matrices of cost sensitive stagewise additive modeling using a multi-class exponential loss function

Zero–one	Absolute cost	Quadratic cost
$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 2 & 3 \\ 2 & 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 0 & 1 \\ 4 & 3 & 2 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 4 & 9 & 16 \\ 1 & 0 & 1 & 4 & 9 \\ 4 & 1 & 0 & 1 & 4 \\ 9 & 4 & 1 & 0 & 1 \\ 16 & 9 & 4 & 1 & 0 \end{pmatrix}$

using PyTorch. We continue to compare the proposed method with neural networks, the base learner model. It is meaningful to observe the performance of the proposed ensemble model itself in comparison with the base learner model. If excelling, the OEAB method has the ability to effectively increase beneficial diversity in the construction of final classifiers. In other words, OEAB has the advantage of ensemble models that cannot be achieved by merely increasing the number of nodes in a neural network itself. For that purpose, we changed the number of hidden nodes in a single neural network to be more varied than the base learner of OEAB. For comparison, we selected the standard neural network classifier and the NNrank [2] mentioned in Section 2.2, which is a neural network that uses an ordinal encoding scheme for ordinal classification. Finally, we also experimented with the random forest (RF), SVM, and GLM [6] algorithms that are known to perform well in general multi-class classification. We implement NNRank based on PyTorch, we use scikit-learn package to implement neural network classifier, RF and SVM, and glmnet package to implement GLM. The experiments are run on a computer with the following features: Intel(R) Xeon(R) Gold 6150 CPU @ 2.70GHz processor, CentOS Linux 7 operating system.

4.4 | Performance comparison on ordinal datasets

For fair performance evaluation, we apply nested cross-validation, including hyperparameter tuning with four-fold cross-validation, and grid-search on the training set and evaluate on the test set. Table 5 shows the hyperparameter search space of each algorithm. For outer cross-validation, we repeat three, four, five, and six-fold cross-validation three times each in order to vary the train-test ratio. The number of base learners of the neural network-based ensemble algorithms was selected considering the following values $M \in \{10, 30\}$ whereas the number of base learners of the RF were selected considering the larger value as $M \in \{50, 100\}$. The number of hidden nodes of the neural network used as the base learner of the ensemble model was considered $H \in \{4, 16, 32, 64\}$, whereas the number of hidden nodes of a single neural network was considered

TABLE 5 Hyper-parameters setting for the experiments

Algorithm	Hyperparameters
OEAB	$M \in \{10, 30\}$; $H \in \{4, 16, 32, 64\}$; $lr \in \{0.001, 0.005\}$;
cs-SAMME	$M \in \{10, 30\}$; $H \in \{4, 16, 32, 64\}$; $lr \in \{0.001, 0.005\}$;
SAMME	$M \in \{10, 30\}$; $H \in \{4, 16, 32, 64\}$; $lr \in \{0.001, 0.005\}$;
NNRank	$H \in \{4, 16, 32, 64, 80, 100, 120\}$; $lr \in \{0.001, 0.005\}$;
ANN	$H \in \{4, 16, 32, 64, 80, 100, 120\}$; $lr \in \{0.001, 0.005\}$;
RF	$M \in \{50, 100\}$; $D \in \{4, 6, 8\}$; Criterion $\in \{\text{gini}, \text{entropy}\}$;
SVM	$C \in \{0.1, 1, 10\}$; $\gamma \in \left\{0.1, 1, 10, \frac{1}{p}, \frac{1}{p \times \text{var}}\right\}$;
GLM	$\alpha \in \{0, 0.1, \cdots, 0.9, 1\}$; $\lambda \in \{0.0001, \ldots, 1\}$;

TABLE 6 Test accuracy results for the eight ordinal datasets

Dataset	OEAB	cs-SAMME	SAMME	NNRank	ANN	RF	SVM	GLM
Toy	0.8135	0.8059	0.7669	**0.8277**	0.8127	0.7801	0.8234	0.3880
Wine-quality	**0.6612**	0.6567	0.6506	0.6204	0.6236	0.6571	0.6434	0.5938
Car	**0.9997**	0.9974	0.9975	0.9994	0.9968	0.9401	0.9979	0.9328
Balance-scale	0.9713	0.9673	0.9666	**0.9749**	0.9697	0.8590	0.9495	0.8890
Tae	**0.6435**	0.6368	0.6288	0.6203	0.6282	0.5629	0.5779	0.5370
Boston-house	**0.7656**	0.7513	0.7484	0.7441	0.7370	0.7541	0.7459	0.7142
Machine	0.6176	0.6211	0.5562	**0.6315**	0.6205	0.6210	0.6157	0.6310
Stock	0.8953	0.9009	0.8922	0.8928	0.8885	0.8943	**0.9042**	0.8354
Average	0.7960	0.7922	0.7759	0.7889	0.7846	0.7586	0.7822	0.6902

Note: The bold represents the best performance for the dataset.

TABLE 7 Test F_1-score results for eight ordinal datasets

Dataset	OEAB	cs-SAMME	SAMME	NNRank	ANN	RF	SVM	GLM
Toy	0.8359	0.829	0.7807	0.8141	0.8306	0.8013	**0.8455**	0.2037
Wine-quality	**0.3713**	0.3568	0.3666	0.3404	0.3416	0.3136	0.3264	0.2931
Car	**0.9998**	0.9907	0.9925	0.9995	0.9893	0.8100	0.9924	0.8843
Balance-scale	0.9431	0.9458	0.9431	**0.9507**	0.9314	0.6012	0.8984	0.7752
Tae	**0.6379**	0.6330	0.6226	0.6363	0.6223	0.5459	0.5742	0.5290
Boston-house	**0.7756**	0.7145	0.7148	0.7460	0.6922	0.7392	0.7389	0.6502
Machine	0.3115	0.2715	0.2442	0.3206	0.3037	0.2766	0.2865	**0.3226**
Stock	0.9014	**0.9075**	0.9000	0.8983	0.8942	0.9000	0.9071	0.8489
Average	0.7221	0.7061	0.6956	0.7132	0.7007	0.6235	0.6962	0.5634

Note: The bold represents the best performance for the dataset.

TABLE 8 Test MAE results for eight ordinal datasets

Dataset	OEAB	cs-SAMME	SAMME	NNRank	ANN	RF	SVM	GLM
Toy	0.1870	0.1947	0.2304	**0.1749**	0.1941	0.2210	0.1768	0.9128
Wine-quality	**0.3764**	0.3914	0.3931	0.4161	0.4374	0.3778	0.4118	0.4409
Car	**0.0003**	0.0028	0.0026	0.0007	0.0038	0.0687	0.0029	0.0717
Balance-scale	0.0314	0.0446	0.0465	**0.0264**	0.0355	0.1878	0.0584	0.1309
Tae	**0.4476**	0.4503	0.4718	0.4505	0.4776	0.5612	0.5442	0.5963
Boston-house	**0.2456**	0.2684	0.2697	0.2671	0.2919	0.2592	0.2667	0.3144
Machine	0.5114	0.5325	0.6311	**0.4856**	0.5264	0.5169	0.5500	0.5598
Stock	0.1048	0.0990	0.1070	0.1087	0.1115	0.1055	**0.0959**	0.1706
Average	0.2381	0.2480	0.2690	0.2412	0.2598	0.2873	0.2633	0.3997

Note: The bold represents the best performance for the dataset.

TABLE 9 Confusion matrix for wine dataset

	Predicted Class											
	OEAB						ANN					
Actual class	3	4	5	6	7	8	3	4	5	6	7	8
3	7	16	72	25	0	0	6	33	54	24	3	0
4	12	58	368	181	17	0	33	55	310	207	30	1
5	10	128	6050	1851	133	0	22	187	5831	1927	193	12
6	0	41	1868	5087	624	36	3	114	1977	4677	829	56
7	0	1	153	908	1281	45	1	14	190	876	1252	55
8	0	0	3	92	102	19	0	0	5	85	114	12

$H \in \{4, 16, 32, 64, 80, 100, 120\}$. And the training was set to 1000 epochs, and the learning rate of the Adam optimizer was considered between 0.001 and 0.005. In addition, hyperparameter tuning was performed for the maximum depth of the trees of RF, C and gamma of SVM, and alpha and lambda of GLM [6].

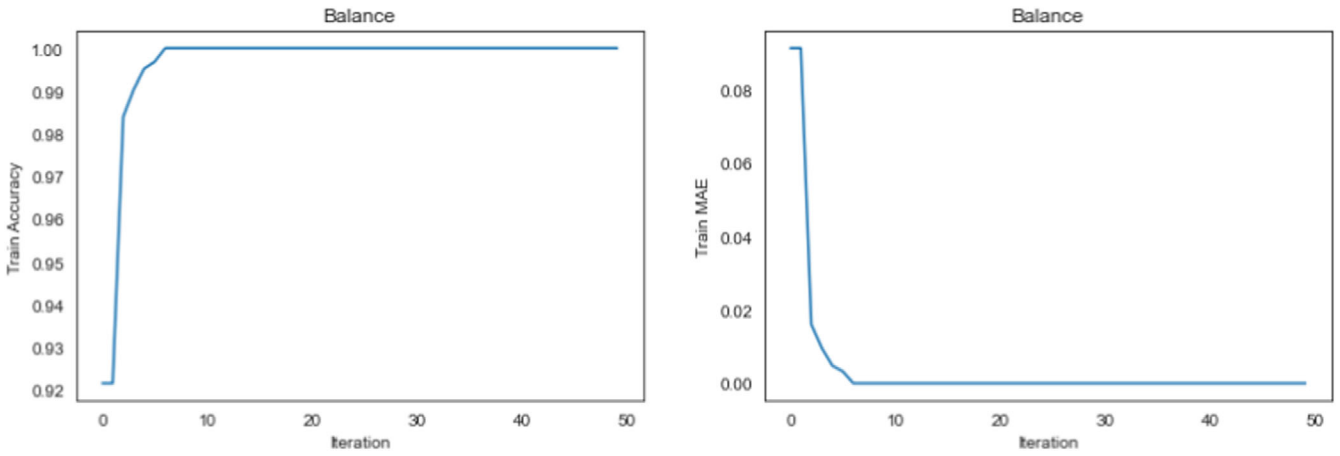
Tables 6–8 show the results obtained from all algorithms for eight ordinal datasets The results represent the mean values of Accuracy, F_1 score, and MAE for 54 experiments, respectively. As the Table shows, the ordinal classification algorithms (OEAB, cs-SAMME, and NNRank) overall surpassed the standard classification algorithms

TABLE 10 Confusion matrix for stock dataset

	Predicted Class									
	OEAB					ANN				
Actual class	1	2	3	4	5	1	2	3	4	5
1	1775	121	0	0	0	1802	94	0	0	0
2	148	2316	259	1	0	169	2251	304	0	0
3	0	237	2864	163	0	0	236	2876	152	0
4	0	0	131	2272	81	0	0	161	2239	84
5	0	0	0	75	957	0	0	0	95	937

TABLE 11 Comparison of training and testing time for the wine dataset: average of the 20 holdouts

Algorithm	Training(s)	Testing(s)
OEAB	8.7243	0.0047
cs-SAMME	6.3636	0.0047
SAMME	6.3581	0.0047
NNRank	3.3450	0.0003
ANN	3.3147	0.0015
RF	0.2127	0.0093
SVM	0.1062	0.0363
GLM	0.7813	0.0009

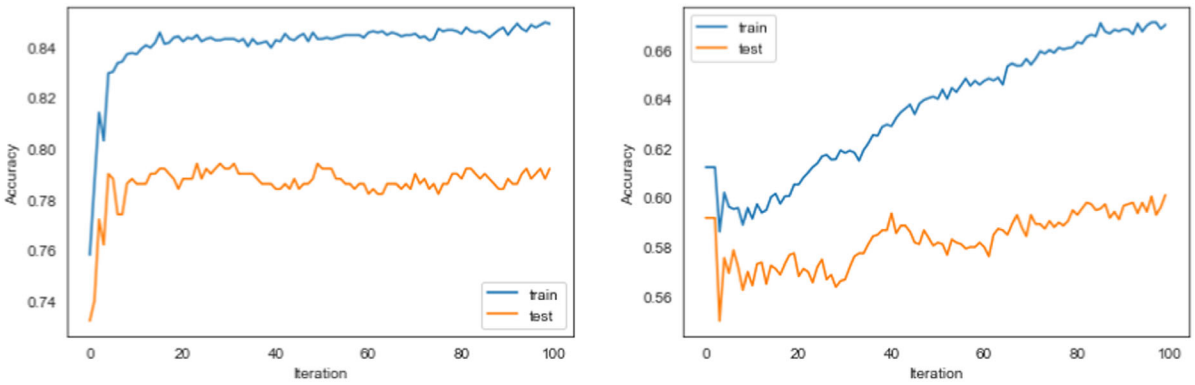


(A) Training accuracy in `balance-scale` (B) Training MAE in `balance-scale`

FIGURE 1 Training accuracy and MAE in balance-scale

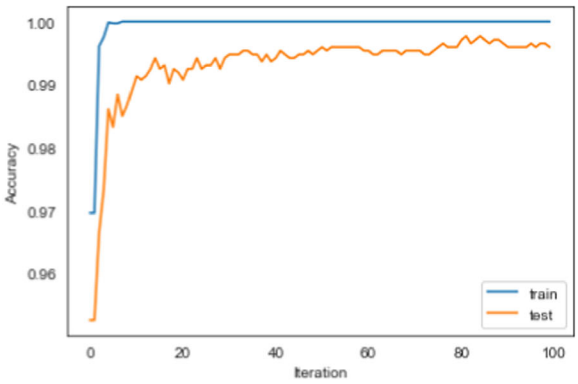
in the adopted performance measures. Therefore, it was shown that the method in which a larger loss is assigned as the distance from the correct answer is more effective than the method in which the same loss is assigned to all misclassified observation points. In addition, the proposed algorithm outperforms cs-SAMME and NNRank overall for all performance measures. Based on the experimental results, we conclude that the proposed method using the encoding scheme classifies ordinal classes closer to the truth than the cost-sensitive algorithm, cs-SAMME. We also conclude that applying ensemble learning to ordinal neural networks has another advantage that cannot be achieved by increasing the number of hidden nodes.

Tables 9 and 10 show the results of adding up all the confusion matrices for the test set of 54 experiments.

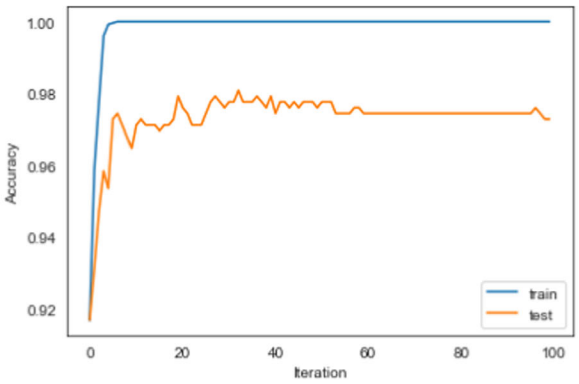


(A) Accuracy in Toy

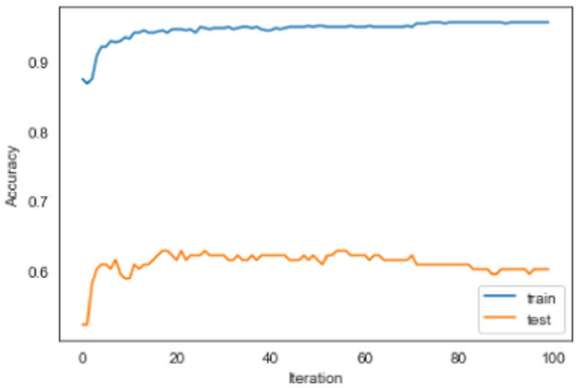
(B) Accuracy in Wine



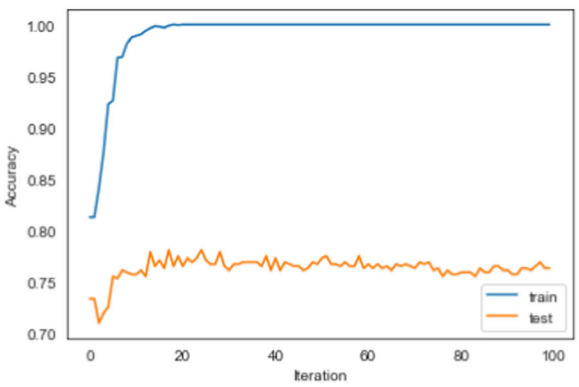
(C) Accuracy in Car



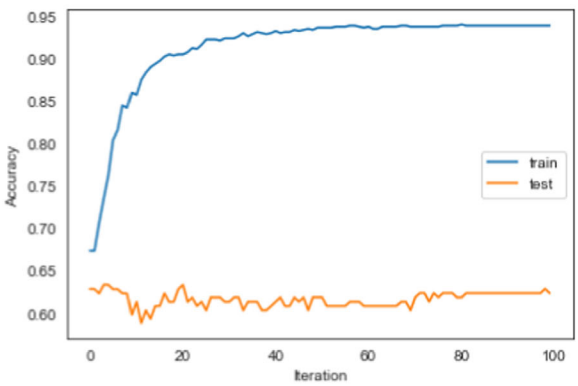
(D) Accuracy in Balance-scale



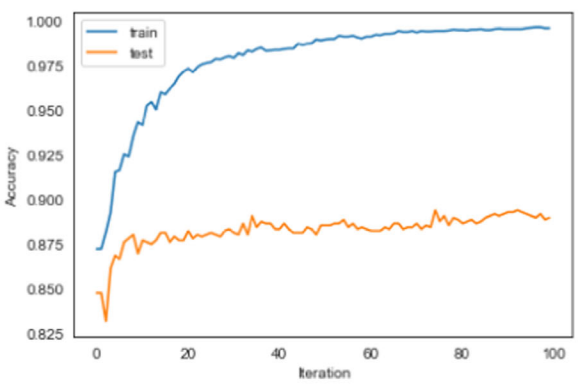
(E) Accuracy in Tae



(F) Accuracy in Boston-house

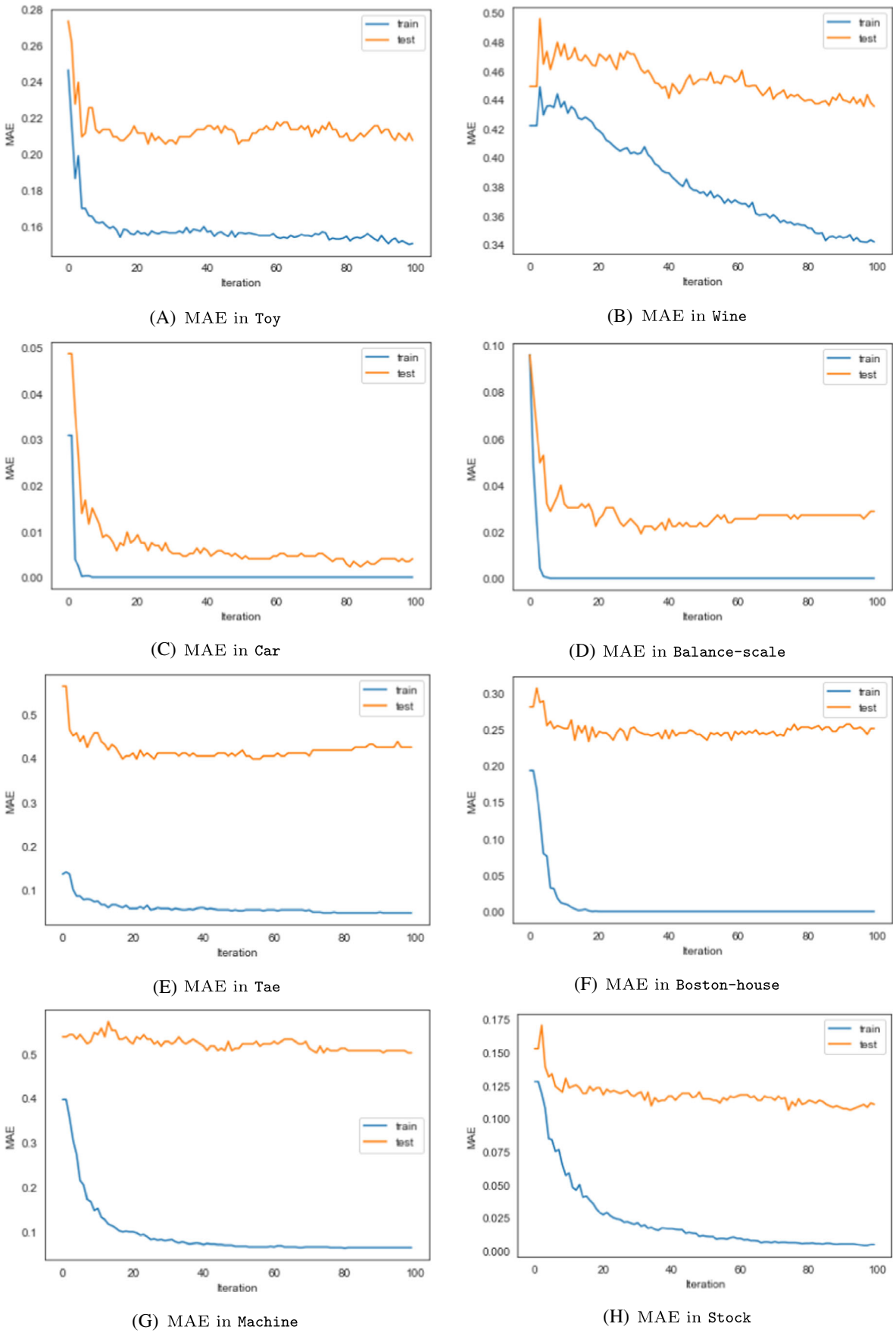


(G) Accuracy in Machine



(H) Accuracy in Stock

FIGURE 2 Five-fold accuracy of OEAB on eight benchmark datasets



(A) MAE in Toy

(B) MAE in Wine

(C) MAE in Car

(D) MAE in Balance-scale

(E) MAE in Tae

(F) MAE in Boston-house

(G) MAE in Machine

(H) MAE in Stock

FIGURE 3 Five-fold MAE of OEAB on eight benchmark datasets

The number of each class in the wine dataset is (10, 53, 681, 638, 199, 18), and the number of classes in the stock dataset is (158, 227, 272, 207, 86). Considering that the data is imbalanced, the optimistic and pessimistic predictions are almost symmetrical. It also shows that the confusion matrices predicted by the OEAB algorithm for the both datasets are more closely clustered diagonally.

Table 11 gives the comparison of training and testing times for all eight algorithms used in the experiment. The wine dataset was used for the experiment, and the training and testing times represent the time for one experiment. The computational complexity of the ensemble algorithm depends on the number of base learners and the choice of the base learner model. Referring to the hyperparameter tuning of the previous experiment, the SAMME-based algorithms set the number of base learners to 10, and the RF set the number of base learners to 50. In particular, the training time of the neural network model is conditioned by the number of hidden nodes. Similarly, referring to previous experiments, we set the number of hidden nodes in the neural network of the ensemble model to 16 and the number of hidden nodes in the neural network of the single model to 32. In training time, the proposed model takes the most time, but in testing time, it shows faster performance than RF and SVM.

4.5 | Effectiveness of AdaBoost

The purpose of the boosting algorithm is to improve performance by adding base classifiers that focus on observations that were misclassified by previous classifiers. Thus, we compared the performance of the ordinal encoding AdaBoost algorithm by adding base classifiers one by one. Figure 1 shows the performance of the training set of balance-scale as the iteration of the boosting algorithm increases. It is clear that the performance improves as the number of base classifiers increases, and the performance for the training set is better than that for the test set. The results in Figures 2 and 3 represent test accuracy and test MAE using 30% of the data as a test set, respectively. Even in the test sets, the model performance tends to improve as the number of base classifiers increases. As in Figures 2 and 3, our algorithm converges for all datasets.

5 | CONCLUSIONS

Boosting is an ensemble method to integrate weak learners and construct a strong one. In reality, there exist ample examples of ordinal target variables. This study tackles effective boosting with ordinal target variables. In this study, we propose a new encoding scheme that properly reflects ordinal target variables and naturally induces multi-class exponential loss understandable in terms of L1 loss. We proposed an AdaBoost algorithm based on the proposed encoding scheme for ordinal classification. We showed that the optimal solution for minimizing multi-class exponential loss is a prediction vector of which the ℓ-th-th element is one-half the log-odds of probability that class c is less than or equal to ℓ. In addition, our ordinal encoding AdaBoost algorithm minimizes the multi-class exponential loss through forward additive stagewise modeling and becomes the Bayes classifier that minimizes error rate. For base learners, we also proposed the loss function of the neural network to satisfy the condition of ordinal relations. The proposed algorithm returns one ordinal classifier instead of $K-1$ classifiers for each iteration. We verified the effectiveness of the proposed algorithm by conducting experiments with various ordinal benchmark datasets. In the future, we envision the inclusion of other machine learning methods such as SVMs or tree-based algorithms as base classifiers instead of neural networks. For future research, we plan to investigate its further theoretical and empirical aspects. Especially for sufficiently large K, one may observe the difference between the solutions of the proposed approaches and those from the use of a continuous target. In addition, one may observe the effect of base learners other than neural networks.

ACKNOWLEDGMENTS

This research was supported by the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2018R1A5A7059549). This work was also supported by "Human Resources Program in Energy Technology" of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resources from the Ministry of Trade, Industry & Energy, Republic of Korea. (No. 20204010600090).

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

ORCID

Kichun Lee https://orcid.org/0000-0002-5184-7151

REFERENCES

1. J. Cardoso, J. P. da Costa, Learning to classify ordinal data: the data replication method. (2007).
2. J. Cheng, Z. Wang, G. Pollastri, A neural network approach to ordinal regression, in: 2008 IEEE international joint conference

on neural networks (IEEE world congress on computational intelligence), IEEE, 2008, pp. 1279–1284.
3. H. Drucker, "*Improving regressors using boosting techniques*," *ICML*, Vol 97, Citeseer, Princeton, New Jersey, 1997, pp. 107–115.
4. E. Frank, M. Hall, A simple approach to ordinal classification, in: European conference on machine learning, Springer, 2001, pp. 145–156.
5. Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, J. Comput. Syst. Sci. 55 (1997), no. 1, 119–139.
6. J. Friedman, T. Hastie, and R. Tibshirani, *Regularization paths for generalized linear models via coordinate descent*, J. Stat. Softw. 33 (2010), no. 1, 1–22.
7. P. A. Gutiérrez, M. Perez-Ortiz, J. Sanchez-Monedero, F. Fernandez-Navarro, and C. Hervas-Martinez, *Ordinal regression methods: Survey and experimental study*, IEEE Trans. Knowl. Data Eng. 28 (2015), no. 1, 127–146.
8. T. Hastie, S. Rosset, J. Zhu, and H. Zou, *Multi-class adaboost, statistics and its*, Interface 2 (2009), no. 3, 349–360.
9. H.-T. Lin, L. Li, Combining ordinal preferences by boosting, in: Proceedings ECML/PKDD 2009 Workshop on Preference Learning, 2009, pp. 69–83.
10. A. Riccardi, F. Fernández-Navarro, and S. Carloni, *Cost-sensitive adaboost algorithm for ordinal regression based on extreme learning machine*, IEEE Trans. Cybernet. 44 (2014), no. 10, 1898–1909.
11. R. E. Schapire and Y. Singer, *Improved boosting algorithms using confidence-rated predictions*, Mach. Learn. 37 (1999), no. 3, 297–336.

How to cite this article: I. Um, G. Lee, and K. Lee, *Adaptive boosting for ordinal target variables using neural networks*, Stat. Anal. Data Min.: ASA Data Sci. J. (2023), 1–15. https://doi.org/10.1002/sam.11613