Neurocomputing 74 (2010) 447–456

Contents lists available at ScienceDirect

Neurocomputing

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

Ordinal extreme learning machine

Wan-Yu Deng [a,b,*], Qing-Hua Zheng [a], Shiguo Lian [c,**], Lin Chen [b], Xin Wang [a]

[a] *MOE KLINNS Lab & SKLMS Lab, Department of Computer Science, Xi'an Jiaotong University, 710049, China*
[b] *Xi'an Institute of Post & Telecommunications, 710121, China*
[c] *France Telecom R&D (Orange Labs), Beijing, China*

ARTICLE INFO

Article history:
Received 22 December 2009
Received in revised form
22 April 2010
Accepted 28 August 2010
Communicated by G.-B. Huang
Available online 16 October 2010

Keywords:
Ordinal regression
Extreme learning machine
Error correcting output codes

ABSTRACT

Recently, a new fast learning algorithm called Extreme Learning Machine (ELM) has been developed for Single-Hidden Layer Feedforward Networks (SLFNs) in G.-B. Huang, Q.-Y. Zhu and C.-K. Siew "[Extreme learning machine: theory and applications," Neurocomputing 70 (2006) 489–501]. And, ELM has been successfully applied to many classification and regression problems. In this paper, the ELM algorithm is further studied for ordinal regression problems (named ORELM). We firstly proposed an encoding-based framework for ordinal regression which includes three encoding schemes: single multi-output classifier, multiple binary-classifications with *one-against-all* (OAA) decomposition method and *one-against-one* (OAO) method. Then, the SLFN was redesigned for ordinal regression problems based on the proposed framework and the algorithms are trained by the extreme learning machine in which input weights are assigned randomly and output weights can be decided analytically. Lastly widely experiments on three kinds of datasets were carried to test the proposed algorithm. The comparative results with such traditional methods as Gaussian Process for Ordinal Regression (ORGP) and Support Vector for Ordinal Regression (ORSVM) show that ORELM can obtain extremely rapid training speed and good generalization ability. Especially when the data set's scalability increases, the advantage of ORELM will become more apparent. Additionally, ORELM has the following advantages, including the capabilities of learning in both online and batch modes and handling non-linear data.

1. Introduction

In machine learning, the classification and metric regression are two important supervised learning problems. Classification is a learning task that assigns labels to instances that belong to a finite set of object classes, while metric regression predicts one value in one continuous distinction for new instances. Ordinal regression, a setting bridging between classification and metric regression, is a learning task of predicting variables of ordinal scale. In contrast to metric regression problems, ordinal regression often requires the grades be discrete and finite. These grades are also different from the class labels in classification problems due to the existence of ranking information. The good ordinal regression methods should not only obtain good accuracy, but also hold the fast training speed and responding speed, the ability to deal with linear and non-linear problems and good online learning ability.

Many machine learning algorithms have been proposed or redesigned for ordinal regression [1,2], including the perceptron [3] and its kernelized generalization [4], the neural network with gradient descent [2,6], the Gaussian process [7,8], the large margin classifier [10,11,29], the k-partite classifier [12], the boosting algorithm [13,14], the constraint classification [15], regression trees [16], Naive Bayes [17], Bayesian hierarchical experts [18], the binary classification approach [19,20], and the optimization of nonsmooth cost functions [21]. Although these ordinal regression methods have some interesting properties, they have also certain disadvantages. For example, Prank [14] is a fast online algorithm whose accuracy suffers when dealing with non-linear data. Large-margin methods [22] are accurate approaches, but they convert the ordinal relations into $O(N^2)$ (N is the number of data points) pairwise ranking constraints, and thus, are impractical for large size datasets. Ordinal support vector machines (ORSVM) [11,10] are another powerful large-margin methods finding $m-1$ real value thresholds (m is the number of ranks), their optimization problem's complexity is $O(N)$, and the prediction speed is slow when the support vector is not sparse. Similarly, the Gaussian process method for ordinal regression (ORGP) [7] has also the difficulty of handling large size datasets. Therefore, some better methods are expected to solve the ordinal regression problem.

Most of these ordinal regression algorithms are based on iterative learning for thresholds, for example, ORGP is the combination of Gaussian process (GP) with threshold-based learning, and

* Corresponding author at: MOE KLINNS Lab & SKLMS Lab, Department of Computer Science, Xi'an Jiaotong University, 710049, China. Tel./fax: +86 8816 6294.
** Corresponding author.
E-mail addresses: wanyu.deng@gmail.com (W.-Y. Deng), shiguo.lian@orange-ftgroup.com (S. Lian).

0925-2312/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2010.08.022

ORSVM is the combination of support vector machine (SVM) with threshold-based learning. These threshold-based approaches are always relatively complex and long consuming for iterative learning steps. If we want to enhance learning speed, we have to give up this iterative learning approach. The encoding approach has been proved to be effective for multi-classification problems by experimental results reported in Allwein's paper [9]. Allwein only discussed the encoding framework for multi-classification problems (called extended correcting output coding, ECOC), ordinal regression problem is different with classification problem as the ordinal relationship of categories exists. Therefore, while designing the encodings for classification, it need not consider this ordinal relationship. But, for ordinal regression, it is necessary to consider this point. In this paper, we proposed three encoding schemes for ordinal regression that can reflect these ordinal relationships. They are different with the coding for multi-classification. For example, considering of the order of the categories, if a data point x belongs to the kth rank, it is automatically classified into lower-order categories $(1, 2, \ldots, k-1)$. So the target output of x is encoded as $t=[1, 1, ..., 1, -1, -1, ..., -1]$ where $t_i (1 \leq i \leq k)$ is set to "1" and other elements "-1". While in classification problem, if a data point x belongs to the kth category, it is will not be able to be classified into another category. So the target output of x will be encoded as $t=[-1, -1, ..., 1, -1, -1, ..., -1]$. This is our first contribution. This proposed framework can be combined with SVM, RVM and so on. In order to get a fast learning algorithm, we select the Extreme Learning Machine [23] and propose three ELM-based ordinal regression methods through designing three different output encoding schemes [9] and the corresponding model structure: learning with a single multi-output ELM (called ORELM-SingleELM), multi-binary ELM with One-against-All scheme (called ORELM-OAA) and multi-binary ELM with One-against-One scheme (called ORELM-OAO). This is our second attribution. Lastly, we analyze and test the proposed schemes by comparing with popular thresholds-based ones, and show that our scheme is very fast and is especially suitable for real-time application, e.g., user mode track and personalized recommendation. These proposed methods origins ELM's advantages, such as learning in both online [28,30,31–33] and batch mode [23], training on very large datasets, handling non-linear data, and extremely rapid training speed. Some comparative experiments will be done to show its performances.

The rest of the paper is arranged as follows. In Section 2, some related work, including ELM and ECOC, are introduced. The proposed ELM schemes for ordinal regression are presented in detail in Section 3, which include the ORELM-SingleELM, ORELM-OAA and ORELM-OAO schemes. In Section 4, the schemes' performances are evaluated and compared with existing ones. Finally, some conclusions are drawn and discussions are given in Section 5.

2. Related work

2.1. Extreme learning machine (ELM)

Recently, a new fast learning algorithm, Extreme Learning Machine (ELM), has been developed for Single-Hidden Layer Feedforward Networks (SLFNs) [23]. In ELM, input weights and biases can be randomly assigned and the output weights can be analytically determined by the simple generalized inverse operation [5]. Compared with traditional learning machines [34,35], ELM not only learns much faster with higher generalization ability but also avoids many difficulties, such as the stopping criteria, learning rate, learning epochs, and local minima [35]. Due to these properties, ELM has been successfully applied to many classification and regression problems [24–27]. In this section, we will give brief introduction about ELM.

For $N (N>0)$ samples $\{(\mathbf{x}_k, \mathbf{t}_k)\}_{k=1}^{N}$ where $\mathbf{x}_k=[x_{k1}, x_{k2}, \ldots, x_{kn}]^{\mathrm{T}}$ and $\mathbf{t}_k=[t_{k1}, t_{k2}, ..., t_{km}]^{\mathrm{T}}$, a standard SLFN [23] with $\tilde{N}$ hidden neurons and activation function $g(\cdot)$ is mathematically modeled as

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i, b_i, \mathbf{x}_k) = \mathbf{o}_k, \quad k=1,\ldots,N \tag{1}$$

where $\mathbf{w}_i=[w_{i1}, w_{i2}, ..., w_{in}]^{\mathrm{T}}$ is the weight vector connecting the ith hidden neuron and the input neurons, $\boldsymbol{\beta}_i=[\beta_{i1}, \beta_{i2}, ..., \beta_{im}]^{\mathrm{T}}$ is the weight vector connecting the ith hidden neuron and the output neurons, $\mathbf{o}_k=[o_{k1}, o_{k2}, ..., o_{km}]^{\mathrm{T}}$ is the kth output vector of the SLFN, and b_i is the threshold of the ith hidden neuron. Set $\mathbf{w}_i \cdot \mathbf{x}_k$ be the inner product of w_i and x_k. These N equations can be written compactly as

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{O}$$

where

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}, \quad \boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\beta}_{\tilde{N}}^{\mathrm{T}} \end{bmatrix}_{\tilde{N} \times m}$$

and $\mathbf{O} = \begin{bmatrix} \mathbf{o}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{o}_N^{\mathrm{T}} \end{bmatrix}_{N \times m}$

Here, $\mathbf{H}$ is called the hidden layer output matrix.

In order to train an SLFN, one may wish to find the specific $\mathbf{w}_i$, b_i, $\boldsymbol{\beta}_i (i=1,2,\ldots,\tilde{N})$ such that

$$\begin{aligned} &\min ||\boldsymbol{\varepsilon}||^2 \\ &\text{s.t.} \sum_{i=1\tilde{N}} \boldsymbol{\beta}_i g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) - \mathbf{t}_k = \boldsymbol{\varepsilon}_k \quad (k=1,\ldots,N) \end{aligned} \tag{2}$$

where $\boldsymbol{\varepsilon}_k=[\varepsilon_{k1}, \varepsilon_{k2}, ..., \varepsilon_{km}]$ is the resident between the target value and real output of kth sample, and $\boldsymbol{\varepsilon}=[\boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2, ..., \boldsymbol{\varepsilon}_N]$. When $\mathbf{H}$ is unknown, the gradient-based learning algorithm is often adopted to train the SLFN, which is susceptible to the local minima and long training times. To overcome these difficulties, Huang et al. proposed a neural network based training algorithm called the Extreme Learning Machine (ELM) [4]. As rigorously proved by Rajaram et al. [1], ELM can work as a universal approximator: it is not difficult to find that the SLFNs with at most N hidden nodes and with almost any non-linear activation function can exactly learn N distinct observations. So, input weights of an SLFN can be randomly chosen (according to any continuous sampling distribution), and the output weights of an SLFN can be analytically determined by the minimum norm least-squares solutions. Since there are not iterative steps, the training speed of ELM can be hundreds of times faster than traditional gradient-based learning algorithms like BP [35] algorithm while obtaining better generalization ability.

2.2. The Extended Error Correcting Output Codes (ECOC)

The commonly used methods for multi-classification can be classified into two types. The first one uses a single multi-output classifier. Another one decomposes the multi-classification problem into multiple binary-classification problems which include two major decomposition methods: *one-against-all* (OAA) and *one-against-one* (OAO). The three methods are unified under the extended Error Correcting Output Codes (ECOC) framework by Allwein [9].

In the extended ECOC framework [9], each category r is associated with a row $\mathbf{M}(r,:)$ of a coding matrix $\mathbf{M} \in \{-1,0,1\}^{m \times \ell}$ where m is the number of categories and ℓ the length of coding.

The "-1" entries indicate that the hypothesis categorizes the examples as a negative class, "1" entries indicate that the hypothesis categorizes the examples as a positive class, and "0" entries indicate that we do not care how the hypothesis categorizes the examples. For the two decomposition schemes, *one-against-all* (OAA) and *one-against-one* (OAO), multiple binary learning algorithms are then run once for each column of the matrix on the induced binary problem. Differently, for the multi-output scheme, only one multi-output classifier is then run for the total output coding matrix $\mathbf{M} \in \{-1,0,1\}^{m\times\ell}$. These methods all yield ℓ hypotheses f_s. Given an example x, we then predict the category r for which the row r of matrix M is "closest" to $(f_1(\boldsymbol{x}),f_2(\boldsymbol{x}),\ldots,f_\ell(\boldsymbol{x}))$. The metrics of distance include loss-based decoding approach and harming decoding method [9]. Some of the entries $M(r,s)$ may be zero, which indicates that we do not care how the hypothesis f_s categorizes the examples with the category r.

3. ELM for ordinal regression

Motivated by the ECOC-based unified framework for multi-classification [9], we propose an encoding-based framework for ordinal regression. The key difference is that ordinal regression has to consider the ordinal relationship among categories and we will give a detailed information in the following section. Based on the proposed framework, many supervised learning algorithms such as ELM, Relevance Vector Machine (RVM) [36] and Support Vector Machine (SVM)[34] can be easily applied to ordinal regression. In this paper, ELM is selected because of its fast training speed and good generalization ability. In this section, three ELM-based approaches will be proposed, including ORELM-SingleELM, ORELM-OAA and ORELM-OAO. In ORELM-SingleELM, ordinal regression is implemented by a single ELM classifier. Differently, ORELM-OAA and ORELM-OAO both decompose the ordinal regression task into multiple binary classifiers. The differences between ORELM-OAA and ORELM-OAO will be discussed in the following section. The proposed algorithms, as shown in Fig. 1, are composed of three steps, i.e., encoding, learning and prediction.

Step 1: **Encoding**. Design the output coding matrix $\mathbf{M} \in \{-1,0,1\}^{m\times\ell}$ for the ordinal regression problem.
Step 2: **Learning**. Re-split or re-label the training set and learn with one multi-output ELM or multiple binary ELMs.
Step 3: **Prediction**. Given a new example x, predict the rank r for which the row r of matrix M is "closest" to $(f_1(\boldsymbol{x}),f_2(\boldsymbol{x}),\ldots,f_\ell(\boldsymbol{x}))$ by loss-based distance or harming distance.

In the following content, we will describe the various encoding methods and the corresponding learning methods, respectively.

3.1. *Ordinal extreme learning machine-single multi-output ELM (ORELM-SingleELM)*

3.1.1. *Encoding*

Similarly to connectionist models in which each class is represented by an output neuron [25], ordinal regression can also be implemented by a single ELM classifier with m output nodes where m is the number of ranks. We call this method ORELM-SingleELM. For each training example x, the target output $\boldsymbol{t}$ is encoded into m bits, $[t_1, t_2, \ldots, t_m]$. Considering of the order of the categories, if a data point x belongs to the kth rank, it is automatically classified into lower-order categories $(1, 2, \ldots, k-1)$. So the target output of x is encoded as $\boldsymbol{t}=[1, 1, \ldots, 1, -1, -1, \ldots, -1]$ where t_i $(1 \le i \le k)$ is set to 1 and other elements -1. The coding matrix M for ORELM is an $m \times m$ matrix. In Table 1, an example taking 5 ranks is described, where the length of coding is $\ell = m$.

The number of coding bits ℓ is equal to the number of ranks m. The ith row represents the output coding vector of ith rank, and every column represents one output coding bit corresponding to an output node.

3.1.2. *Learning*

For the samples in $\{(\mathbf{x}_i,r_i)\}_{i=1}^N$, where $\boldsymbol{x}_i = [x_{i1},x_{i2},\ldots,x_{in}]^T \in \mathbf{R}^n$ and $r_i \in \mathbf{N}$, the ranks r_i are firstly transformed into output coding format: $\mathbf{t}_i = [t_{i1},t_{i2},\ldots,t_{i\ell}]^T$ which is the ith row of output coding matrix. Then a single ELM with $\tilde{N}$ hidden nodes and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g_i(\boldsymbol{x}_j) = \sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g_i(\boldsymbol{a}_i \cdot \boldsymbol{x}_j + \boldsymbol{b}_i) = \boldsymbol{t}_j, \quad j = 1,\ldots,N$$

The above N equations can be written compactly as

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$$

The key point of ELM is that the input weights $(\boldsymbol{a}_i, \boldsymbol{b}_i)$ are assigned randomly instead of tuning. After $(\boldsymbol{a}_i, \boldsymbol{b}_i)$ was assigned, the determination of the output weights $\boldsymbol{\beta}$ is as simple as finding the

Table 1
Output coding matrix $\mathbf{M} \in \{-1,0,1\}^{m\times\ell}$ of ORELM-SingleELM algorithm.

Rank	Coding bits				
	1	2	3	4	5
1	1	−1	−1	−1	−1
2	1	1	−1	−1	−1
3	1	1	1	−1	−1
4	1	1	1	1	−1
5	1	1	1	1	1

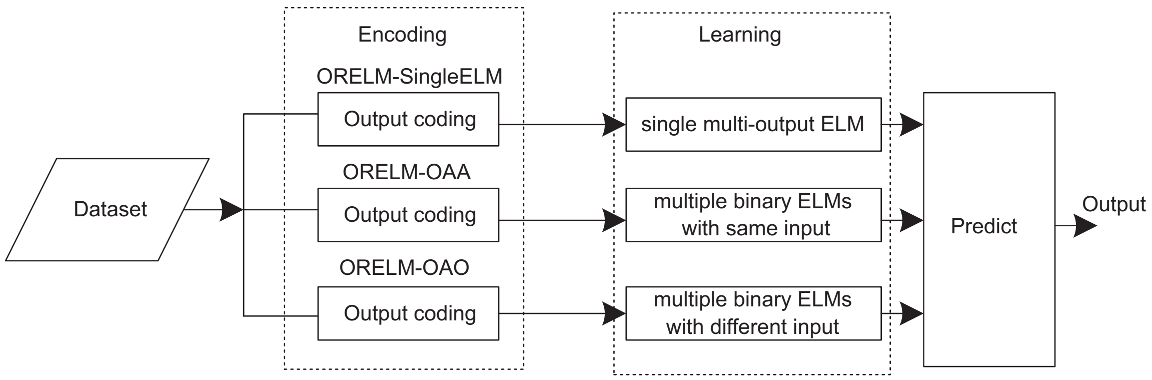


Fig. 1. The framework of ORELM algorithms (SingleELM, OAA and OAO).

least-square solutions to the given linear system,

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T}$$

where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse [5] of the hidden layer output matrix $\mathbf{H}$. The learning algorithm can be summarized as follows:

ORELM-SingleELM Learning Procedure:
Input: training set $\aleph = \{(\boldsymbol{x}_i, \boldsymbol{r}_i) | \boldsymbol{x}_i \in \mathbf{R}^n, r_i \in \mathbf{N}\}_{i=1}^{N}$
Output: the parameters $(\boldsymbol{a}, \boldsymbol{b})$ and $\boldsymbol{\beta}$.

(1) Set $g(\boldsymbol{x})$ (e.g. Sigmoid, Sin, etc.) and the number of hidden neurons $\tilde{N}$(integer).
(2) Randomly assign the input weights $\boldsymbol{a}_i$ and biases $\boldsymbol{b}_i$ according to some continuous probability density function.
(3) Calculate the hidden layer output matrix $\mathbf{H}$.
(4) Calculate the output weight $\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T}$.
(5) Return $(\boldsymbol{a}, \boldsymbol{b})$ and $\hat{\boldsymbol{\beta}}$.

3.2. Ordinal Extreme Learning Machine-One-against-All (ORELM-OAA)

3.2.1. Encoding

Considering that the multi-classification problem can be decomposed into multiple binary classifiers using the OAA method, the ordinal regression can also be decomposed into multiple binary classifiers. Differently, in the encoding scheme, there is an ordered relationship between categories. By considering the ordered relationship, if a data point x belongs to the kth rank, it is automatically classified into lower-order categories $(1, 2, ..., k-1)$. Thus, the m-ranks ordinal regression can be implemented by $m-1$ binary ELM classifiers. For the ith binary ELM classifier, ELM_i, the data set is decomposed into two subsets: one labeled as "1" for all the examples that satisfy $1 \le rank \le i$, and the other labeled as "-1". Taking 5 ranks for example, the coding matrix $\mathbf{M}$ for OAA decomposition is shown in Table 2. As a comparison, the $\mathbf{M}$ for ORELM-OAA is an $m \times (m\text{-}1)$ matrix and is described in Table 3, where the length of coding is $\ell = m-1$. As is known, the input data for each binary ELM classifier are the same, but the target output data are different.

Every column represents the ith $(i=1, 2, ..., m)$ binary classifier and its corresponding output bit. The title of column, i.e. [1|2, 3, 4, 5], represents the split of the training set. Here, "|" splits the training set into two parts. The category on the left side of "|" is re-labeled as "1" and that on the right side of "|" is re-labeled as "-1".

Obviously, it is different from multi-classifications: in multi-classifications problem, the number of category on the left side of "|" is only one, while the number of category in ORELM-OAA is changeable and incremental in ordinal regression. This is because, if a data point x belongs to the kth rank, it is automatically classified into lower-order categories $(1, 2, ..., k-1)$ as well.

Table 2
The output-coding matrix of multi-classification problem based on the OAA decomposition method.

Class	Coding bits				
	[1|2,3,4,5]	[2|1,3,4,5]	[3|1,2,4,5]	[4|1,2,3,5]	[5|1,2,3,4]
1	1	−1	−1	−1	−1
2	−1	1	−1	−1	−1
3	−1	−1	1	−1	−1
4	−1	−1	−1	1	−1
5	−1	−1	−1	−1	1

Table 3
The output-coding matrix of ORELM-OAA.

Rank	Coding bits			
	[1|2,3,4,5]	[1,2|3,4,5]	[1,2,3|4,5]	[1,2,3,4|5]
1	1	1	1	1
2	−1	1	1	1
3	−1	−1	1	1
4	−1	−1	−1	1
5	−1	−1	−1	−1

3.2.2. Learning

ORELM-OAA decomposes an ordinal regression problem into multiple binary-class problems, each of which is implemented by a binary classifier. Since in ELM, the parameters of hidden nodes are assigned randomly and independent of training data, all the binary classifiers in ORELM-OAA can share a common hidden node set. Thus, ORELM-OAA can be considered as a combination of $\ell = m-1$ binary ELM classifiers. ℓ groups of independent training data are generated from the original training data. The ℓ output neurons which are independently learned represent the outputs of the ℓ binary classifiers. When the lth $(l=1,2,...,\ell)$ group of training data enters the model, the output weight $\boldsymbol{\beta}_l$ connecting to the lth binary classifier's output neuron is learned without affecting other weights.

Define the weight vector $\boldsymbol{\beta}_l$ as $[\beta_{l1}, \beta_{l2}, ..., \beta_{l\tilde{N}_l}]$ where $\beta_{lj}(j=1,2,...,\tilde{N}_l)$ denotes the weight connecting the jth shared hidden node to the lth binary classifier's output neuron. $\boldsymbol{\beta}_l$ can be determined by

$$\boldsymbol{\beta}_l = (\mathbf{H}_l)^{\dagger}\boldsymbol{t}_l$$

where $\boldsymbol{t}_l(l=1,2,...,\ell)$ is defined as

$$\boldsymbol{t}_l = [t_{l1}, t_{l2}, ..., t_{lN_l}]^{\mathrm{T}}_{l\times N_l}$$

and the matrix $\mathbf{H}_l(l=1,2,...,\ell)$ is defined as

$$\mathbf{H}_l = \begin{bmatrix} g(\boldsymbol{a}_1\cdot\boldsymbol{x}_1+b_1) & \cdots & g(\boldsymbol{a}_{\tilde{N}_l}\cdot\boldsymbol{x}_1+b_{\tilde{N}_l}) \\ \vdots & \cdots & \vdots \\ g(\boldsymbol{a}_1\cdot\boldsymbol{x}_{N_l}+b_1) & \cdots & g(\boldsymbol{a}_{\tilde{N}_l}\cdot\boldsymbol{x}_{N_l}+b_{\tilde{N}_l}) \end{bmatrix}_{N_l\times\tilde{N}_l}$$

where N_l is the training data's size for the ith binary classifier and $\tilde{N}_l$ $(l=1,2,...,\ell)$ is the number of hidden neurons of the lth ELM. $\tilde{N}_l$ $(l=1,2,...,\ell)$ may be different for different binary ELM. For simplicity, we set $\tilde{N}_l$ be the same number.

3.3. Ordinal Extreme Learning Machine-One-against-One (ORELM-OAO)

3.3.1. Encoding

Similar to the multi-classifications problem, the m-ranks ordinal regression can be decomposed into $m(m-1)/2$ binary classifiers, $ELM_i(r_1, r_2)$ $(i=1, 2, ..., m(m-1)/2;\ r_1=1, 2, ..., m-1;\ r_2=1, 2, ..., m)$, that discriminate the rank r_1 and r_2. The training examples for each $ELM_i(r_1, r_2)$ are only the examples from rank r_1 and rank r_2, while ignoring the ones from other ranks. The target output for the r_1th rank examples is labeled as "1" and the one for the r_2th rank examples is labeled as "-1". Thus, the coding matrix M for ORELM-OAO is an $m \times m(m-1)/2$ matrix. Table 4 shows an example with 5 ranks, where the length of coding is $\ell = m(m-1)/2$. Here, the input data for each binary ELM classifier are different.

Obviously, it is accurately same to the coding matrix of multi-classification. Thus, ordinal regression can be directly transformed into multi-classification problem.

Table 4
Output-coding matrix of ORELM-OAO.

Rank	Coding bits									
	[1|2]	[1|3]	[1|4]	[1|5]	[2|3]	[2|4]	[2|5]	[3|4]	[3|5]	**[4|5]**
1	1	1	1	1	0	0	0	0	0	**0**
2	−1	0	0	0	1	1	1	0	0	**0**
3	0	−1	0	0	−1	0	0	1	1	**0**
4	0	0	−1	0	0	−1	0	−1	0	**1**
5	**0**	**0**	**0**	**−1**	**0**	**0**	**−1**	**0**	**−1**	**−1**

3.3.2. *Learning*

The learning procedure of ORELM-OAO is nearly the same to the one of ORELM-OAA except two differences. Firstly, there are $\ell = m-1$ binary SLFNs in ORELM-OAA while $\ell = m(m-1)/2$ in ORELM-OAO. Secondly, the input data for each ELM classifier in ORELM-OAA are same while the ones for each classifier in ORELM-OAO are different. This is because the input data for each $ELM_i(r_1, r_2)$ in ORELM-OAO are only examples from rank r_1 and rank r_2 which ignore the examples from other ranks.

3.4. *Prediction*

These three approaches have the same prediction process and therefore it will be presented in this section in summary. Here, the loss-based decoding approach [9] is utilized to decode the collective outputs $\boldsymbol{y} = [y_1, y_2, \ldots, y_\ell]$ which is more effective than the harming decoding method [9]. In this approach, the chosen rank is the one that is most consistent with the output y in the sense that the total loss for x is minimal. The total loss on an example x is given by

$$d_L(\mathbf{M}(r), \boldsymbol{y}(\boldsymbol{x})) = \sum_{i=1}^{\ell} L(\mathbf{M}(r,i)\cdot \boldsymbol{y}_i(\boldsymbol{x})), \quad r = 1,2,\ldots,m$$

In ORELM, $\mathbf{M}$ is the encoding matrix as shown in Table 1. $\mathbf{M}(r,i)$ denotes the elements located in the rth row and the ith column, and $y_i(x)$ $(i = 1,2,\ldots,\ell)$ is the output value for the ith output nodes on an example $\boldsymbol{x}$. In ORELM-SingleELM, ORELM-OAA and OREL-OAO, $\mathbf{M}$ is the encoding matrix as shown in Tables 1, 3 and 4 respectively, $\boldsymbol{y}_i(\boldsymbol{x})$ $(i = 1,2,\ldots,\ell)$ is the output value for the ith binary classifier on an example x. $L(\cdot)$ is the exponential loss function that satisfies $L(\mathbf{M}(r,q)\cdot \boldsymbol{y}_i(\boldsymbol{x})) = e^{-\mathbf{M}(r,i)\cdot \boldsymbol{y}_i(\boldsymbol{x})}$. Based on the above loss function, the final rank for the training example $\boldsymbol{x}$ is given by

$$\hat{r} = \underset{i \in \{1,2,\ldots,m\}}{\operatorname{argmin}}\; d_L(\mathbf{M}(i)\cdot \boldsymbol{y}(\boldsymbol{x}))$$

4. Performance evaluation

In this section, we test the proposed schemes, i.e., ORELM-SingleELM, ORELM-OAO and ORELM-OAA, on quite a few benchmark datasets for ordinal regression, and compare them with existing works, ORGP [7] and ORSVM [11]. The simulations for ORELM-SingleELM, ORELM-OAO and ORELM-OAA are carried out in MATLAB 8.1 environment running on a Pentium 4, 2.7 GHz CPU. The simulations for ORGP and ORSVM are carried out by using compiled C-coded ORGP[1] and ORSVM[2] packages, running on the same PC. The activation function used in our proposed schemes is a sigmoid function $g(x) = 1/(1+e^{-\lambda x})$. If the data have been normalized, λ is set as 1, otherwise as 0.001 or 0.01. These methods are compared concerning the network complexity, mean absolute error, mean zero-one error, and the training and testing time:

- Mean zero-one error (MZOE) is the percentage of wrong assignments of ordinal categories.
- Mean absolute error (MAE) is the average deviation of the prediction from the true target, i.e., $(1/N)\sum_{i=1}^{N} |\hat{r}_i - r_i|$, in which we treat the ordinal scales as consecutive integers.

4.1. *Artificial data*

In this example, a metric regression data set $\{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in \mathbf{R}^2, y_i \in \mathbf{R}\}_{i=1}^{1000}$ is firstly generated by the following function:

$$y = (0.5 - x_1)^2 + (2 - x_2)^2$$

where x_i's are uniformly randomly distributed on the interval (0,1).

Firstly, the data set is re-labeled and transformed into an ordinal regression data set $\{(\boldsymbol{x}_i, t_i) | \boldsymbol{x}_i \in \mathbf{R}^2, \quad t_i \in \{1,2,3\}\}_{i=1}^{1000}$ by the following rule:

$$t = \begin{cases} 1 & y \le \theta_1 \\ 2 & \theta_1 < y \le \theta_2 \\ 3 & y > \theta_2 \end{cases}$$

where the thresholds satisfy $\theta_1 = 2.08$ and $\theta_2 = 3.17$.

Then, a training set $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^{700}$ with 700 data and testing set $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^{300}$ with 300 data are obtained randomly. In order to make the regression problem 'real', the large uniform noise distributed on $[-0.45, 0.45]$ has been added to all the training samples while the testing data remain noise-free.

There are 25 hidden nodes assigned for our ORELM-SingleELM, ORELM-OAA and ORELM-OAO algorithms, respectively. 20 trials have been conducted for all the algorithms and the average results are shown in Table 5. It can be seen that ORELM-SingleELM spent **0.0381** s CPU time obtaining the testing mean zero-one error **0.0333**, however, it takes **60** s CPU time for ORSVM, and **123** s CPU time for ORGP to reach a little higher testing error **0.0533**. Thus, the ORELM-SingleELM runs **1574 times** faster than the ORSVM algorithm and **3228 times** faster than the ORGP algorithm even though the fact that C executable program may be faster than MATLAB program has not been taken into account. Since the number of support vectors obtained by ORSVM is much larger than the hidden nodes required by ORELM-SingleELM (493 vs. 25), the testing time spent for the obtained ORSVM is **300** times longer than the testing time for ORELM-SingleELM, meaning that after training, the ORELM-SingleELM may respond to new external unknown stimuli much faster than ORSVM in real deployment. Fig. 2(a) shows the true data distribution, while Fig. 2(b)–(d) show the approximated results of the ORELM-SingleELM, ORELM-OAA and ORELM-OAO, respectively.

4.2. *Small sample benchmark data*

The nine datasets shown in Table 6 are originally used for the metric of regression problems. Chu and Ghahramani [7] discretized the real-value targets into ordinal quantities by using equal-length binning. For each data set, two versions are generated by discretizing the target values into five and ten intervals, respectively. Here, we randomly partitioned each data set into training/test splits as specified in Table 6. The partitioning was repeated for 20 times independently.

Tables 7 and 8 show the comparison of network complexity of ORELM-OAA, ORELM-OAO and ORELM-SingleELM on 5-bin and 10-bin cases, respectively. It is apparent that ORELM-OAO requires a smaller number of hidden nodes than ORELM-OAA and ORELM-SingleELM since each binary classifier in ORELM-OAO utilizes

[1] www.gatsby.ucl.ac.uk/~chuwei/code/gpor.tar
[2] www.gatsby.ucl.ac.uk/~chuwei/code/svorex.tar

Table 5
Performance comparison in artificial ordinal regression data set.

	ORELM-SingleELM[a]	ORELM-OAA[a]	ORELM-OAO[a]	ORSVM[b]	ORGP[b]
Mean zero one error	**0.0333**	0.0433	0.0333	0.0533	0.0533
Mean absolute error	**0.0333**	0.0433	0.0333	0.0533	0.0533
Nodes/SVs	25($\lambda=1$)	25($\lambda=1$)	25($\lambda=1$)	493 ($C=100$ and kappa=0.006)[c]	–
Training time	**0.0381**	0.0625	0.0625	60	123
Testing time	$<\mathbf{10^{-4}}$	$<\mathbf{10^{-4}}$	$<\mathbf{10^{-4}}$	0.031	0.0001

[a] $C=100$ and kappa=0.006 are the parameters of the trained ORSVM.
[b] Run in MATLAB environment.
[c] Run in *C* executable environment (*C* executable is faster than MATLAB environment).

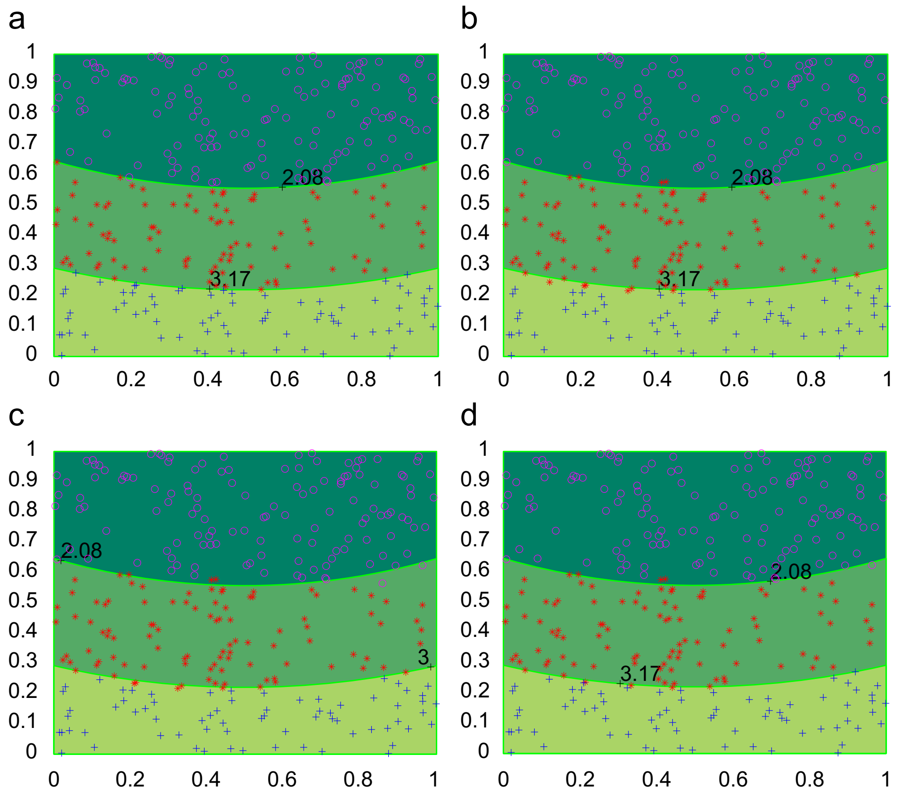


Fig. 2. Approximated results of the (b) ORELM-SingleELM, (c) ORELM-OAA and (d) ORELM-OAO. (a) True data.

Table 6
Specification of real world small sample benchmark problems.

Data sets	**Attributes (Num Nom)**	**Training set**	**Testing set**
Diabetes	2(2,0)	30	13
Pyrimidines	27(27,0)	50	24
Triazines	60(60,0)	100	86
Wisconsin	32(32,0)	130	64
Machine CPU	7(4,3)	200	192
Auto MPG	13(12,1)	300	206
Boston housing	9(9,0)	600	350
Stocks domain	8(7,1)	1000	3177
Abalone	2(2,0)	30	13

Table 7
Comparison of network complexity (5-bin).

Data set	ORELM-OAA	ORELM-OAO	ORELM-SingleELM
Diabetes	5($\lambda=0.001$)	**3**($\lambda=0.001$)	**3**($\lambda=0.001$)
Pyrimidines	**5**($\lambda=1$)	**5**($\lambda=1$)	15($\lambda=1$)
Triazines	15($\lambda=0.001$)	**3**($\lambda=0.001$)	10($\lambda=0.001$)
Wisconsin	15($\lambda=0.001$)	19($\lambda=0.001$)	**10**($\lambda=0.001$)
Machine CPU	30($\lambda=0.001$)	**20**($\lambda=0.001$)	25($\lambda=0.001$)
Auto MPG	20($\lambda=0.001$)	**15**($\lambda=0.001$)	25($\lambda=0.001$)
Boston housing	50($\lambda=0.001$)	**30**($\lambda=0.001$)	40($\lambda=0.001$)
Stocks domain	150($\lambda=0.001$)	**40**($\lambda=0.001$)	160($\lambda=0.001$)
Abalone	31($\lambda=4$)	**20**($\lambda=1$)	40($\lambda=1$)

a smaller number of training data than that in ORELM-OAA or ORELM-SingleELM does.

The comparison of testing accuracy of ORELM-OAA, ORELM-OAO and ORELM-SingleELM is given in Tables 9 and 10. Seen from the tables, ORELM-OAA achieves the best testing accuracy on 3 datasets, ORELM-SingleELM achieves the best testing accuracy on 4 datasets, and ORELM-OAO achieves the best testing accuracy on 2 datasets. Thus, ORELM-SingleELM has better advantages on the testing accuracy.

Tables 11 and 12 give the comparison of training and testing time of ORELM-OAA, ORELM-SingleELM and ORELM-OAO in 5 ranks and 10 ranks, respectively. As can be seen, the ORELM-OAO and ORELM-OAA require more training time than ORELM-SingleELM. Furthermore, ORELM-SingleELM usually requires less testing time than ORELM-OAA and ORELM-OAO. The reason is that in ORELM-OAA and ORELM-OAO, multiple binary classifiers need to be used, which spend more time.

Tables 13 and 14 give the comparison of testing accuracy and training/testing time of ORELM, ORSVM and ORGP. Here, we choose the best result to report. It can be seen that, in small sample cases, the ORGP achieves better testing accuracy than ORELM in most datasets. But, ORELM runs about **1625 times**(ORELM **0.048**s vs ORGP **78.166**s) faster than the ORGP. Thus, for small sample datasets and without time constraint, ORGP is the best choice. Comparing with ORSVM, ORELM runs about **716 times** (ORELM **0.048**s vs ORSVM **34.463**s) faster than ORSVM. Furthermore, one can find that ORELM often requires less testing time than SVM. The reason is that in ORSVM, more support vectors need to be used.

4.3. Large sample benchmark data set

In this experiment, we selected 5 very large regression data sets (in Table 15). The input vectors were normalized to zero mean. The target values of these datasets were discretized into 5 ordinal quantities using equal-frequency binning. For each data set, a large subset was randomly selected for training (about 70%) and then tested on the remaining samples (about 30%), as specified in Table 15. The partitioning was repeated for 20 times independently. Seen from the experimental results in Tables 16 and 17, the ORELM (here we select ORELM-SingleELM) often yields better results (of mean zero-one error) than the ORSVM and ORGP with thousands of times faster training speed. It shows that ORELM is competent for large scale data set.

Table 8
Comparison of network complexity (10 bin).

Data set	ORELM-OAA	ORELM-OAO	ORELM-SingleELM
Diabetes	3($\lambda=0.001$)	**2**($\lambda=0.001$)	3($\lambda=0.001$)
Pyrimidines	17($\lambda=1$)	**5**($\lambda=0.01$)	15($\lambda=0.001$)
Triazines	**7**($\lambda=1$)	14($\lambda=1$)	10($\lambda=1$)
Wisconsin	25($\lambda=0.001$)	15($\lambda=1$)	**3**($\lambda=0.01$)
Machine CPU	20($\lambda=0.001$)	**13**($\lambda=0.001$)	25($\lambda=0.001$)
Auto MPG	23($\lambda=0.001$)	**13**($\lambda=0.001$)	25($\lambda=0.001$)
Boston housing	80($\lambda=0.001$)	**30**($\lambda=0.001$)	60($\lambda=0.001$)
Stocks domain	170($\lambda=0.001$)	**110**($\lambda=0.001$)	150($\lambda=0.001$)
Abalone	55($\lambda=0.001$)	**20**($\lambda=0.001$)	55($\lambda=0.001$)

Table 9
Comparison of testing accuracy of ORELM-OAA,ORELM-OAO and ORELM-SingleELM (5 bin).

Datasets	Mean zero one error			Mean absolute error		
	ORELM-OAA	ORELM-OAO	ORELM-SingleELM	ORELM-OAA	ORELM-OAO	ORELM-SingleELM
Diabetes	0.5385	**0.4885**	0.5192	0.6538	**0.6077**	0.6192
Pyrimidines	**0.4729**	0.4979	0.5146	**0.5667**	0.6687	0.6375
Triazines	0.5523	0.5552	**0.5366**	0.7273	0.7384	**0.7064**
Wisconsin	0.7172	0.7188	**0.7156**	1.0586	1.2711	**1.0414**
Machine	0.1788	0.2314	**0.1737**	0.2119	0.3220	**0.2017**
Auto MPG	**0.2495**	0.2682	0.2539	**0.2562**	0.2799	0.2628
Boston	0.2951	**0.2932**	0.2968	**0.3221**	0.3226	0.3243
Stocks	0.1121	0.1286	**0.1111**	0.1123	0.1287	**0.1113**
Abalone	**0.2172**	0.2205	0.2174	0.2338	0.2399	**0.2334**

Table 10
Comparison of testing accuracy of ORELM-OAA,ORELM-OAO and ORELM-SingleELM(10 bin).

Datasets	Mean zero one error			Mean absolute error		
	ORELM-OAA	ORELM-OAO	ORELM-SingleELM	ORELM-OAA	ORELM-OAO	ORELM-SingleELM
Diabetes	**0.6577**	0.6923	**0.6577**	1.2577	1.0962	**1.0962**
Pyrimidines	**0.6208**	0.7542	0.6542	**1.0833**	1.6833	1.1333
Triazines	0.6919	0.6971	**0.6953**	1.2866	1.4895	**1.2756**
Wisconsin	0.8523	**0.7594**	0.8672	2.4375	2.8984	**2.1250**
Machine	**0.3737**	0.4127	**0.3737**	0.5500	0.6653	**0.5186**
Auto MPG	**0.4372**	0.4685	**0.4372**	0.5333	0.5940	**0.5255**
Boston	0.6578	0.6653	**0.4556**	1.1303	1.2614	**0.5937**
Stocks	0.3200	0.3021	**0.1780**	0.3631	0.3624	**0.1797**
Abalone	0.4230	0.4279	**0.4222**	0.5342	0.5131	**0.5115**

Table 11
Comparison of training and testing time (5 bin).

Data set	ORELM-OAA		ORELM-OAO		ORELM-SingleELM	
	Training(s)	Testing(s)	Training(s)	Testing(s)	Testing(s)	Training(s)
Diabetes	**$<10^{-4}$**	**$<10^{-4}$**	0.0156	0.0016	**$<10^{-4}$**	0.0016
Pyrimidines	0.0094	**$<10^{-4}$**	0.0219	0.0016	**0.0016**	**$<10^{-4}$**
Triazines	0.0078	**0.0016**	0.0117	**0.0016**	**0.0031**	0.0031
Wisconsin	0.0078	0.0031	0.0313	**$<10^{-4}$**	**0.0016**	0.0016
Machine CPU	0.0250	0.0016	0.0313	0.0016	**0.0047**	0.0016
Auto MPG	0.0141	0.0031	0.0234	**0.0016**	**0.0047**	0.0031
Boston housing	0.0672	0.0031	0.0578	0.0031	**0.0109**	0.0031
Stocks domain	0.8633	0.0109	**0.0984**	0.0078	0.2461	**0.0063**
Abalone	0.0836	0.0250	0.0609	**0.0148**	**0.0336**	0.0234

Table 12
Comparison of training and testing time (10 bin).

Datasets	ORELM-OAA		ORELM-OAO		ORELM-SingleELM	
	Training(s)	Testing(s)	Training(s)	Testing(s)	Training(s)	Testing(s)
Diabetes	$<\mathbf{10^{-4}}$	$<10^{-4}$	0.0070	$<10^{-4}$	$<\mathbf{10^{-4}}$	$<10^{-4}$
Pyrimidines	0.0094	$<\mathbf{10^{-4}}$	0.0328	0.0031	$<\mathbf{10^{-4}}$	$<\mathbf{10^{-4}}$
Triazines	0.0047	$<\mathbf{10^{-4}}$	0.0672	0.0016	**0.0031**	0.0031
Wisconsin	0.0375	$<\mathbf{10^{-4}}$	0.0656	0.0016	**0.0016**	$<\mathbf{10^{-4}}$
Machine CPU	0.0156	$<\mathbf{10^{-4}}$	0.0852	0.0016	**0.0031**	0.0070
Auto MPG	0.0281	$<\mathbf{10^{-4}}$	0.0719	$<\mathbf{10^{-4}}$	**0.0031**	0.0031
Boston housing	0.4086	0.0094	0.1437	0.0063	**0.0398**	$<\mathbf{10^{-4}}$
Stocks domain	1.5172	**0.0125**	1.4766	0.0164	**0.2469**	0.0141
Abalone	0.3656	0.0164	0.1531	**0.0094**	**0.0383**	0.0258

Table 13
Comparison of testing accuracy of ORELM, ORSVM, and ORGP (5 bin).

Datasets	Mean zero one error			Mean absolute error		
	ORELM	ORSVM	ORGP	ORELM	ORSVM	ORGP
Diabetes	**0.4885**	0.5731	0.5423	**0.6077**	0.7462	0.6615
Pyrimidines	0.4729	0.4146	**0.3646**	0.5667	0.4500	**0.3917**
Triazines	0.5366	0.5419	**0.5262**	0.7064	0.6977	**0.6872**
Wisconsin	0.7156	0.7078	**0.6571**	1.0414	1.0031	**1.0102**
Machine CPU	0.1737	0.1737	**0.1653**	0.2017	0.1915	**0.1847**
Auto MPG	0.2495	0.2573	**0.2375**	0.2562	0.2596	**0.2411**
Boston housing	0.2932	0.2556	**0.2449**	0.3221	0.2672	**0.2585**
Stocks domain	0.1111	**0.1081**	0.1199	0.1113	**0.1081**	**0.1081**
Abalone	0.2172	0.2158	0.215	0.2334	**0.2293**	**0.2293**

Table 14
Comparison of training and testing time of ORELM, ORSVM, and ORGP (5 bin).

Datasets	Training time (s)			Testing time (s)		
	ORELM	ORSVM	ORGP	ORELM	ORSVM	ORGP
Diabetes	**0.0156**	0.356	0.5	0.0016	$<\mathbf{10^{-4}}$	0.002
Pyrimidines	**0.0094**	0.156	3	$<\mathbf{10^{-4}}$	0.0016	0.0016
Triazines	**0.0016**	0.656	20	$<\mathbf{10^{-4}}$	0.0015	0.0012
Wisconsin	**0.0031**	4.5	5	0.0031	0.016	**0.0021**
Machine CPU	**0.0016**	2.5	8	**0.0016**	0.016	**0.0016**
Auto MPG	**0.0141**	30	7	0.0031	0.203	**0.0026**
Boston housing	**0.0578**	170	20	**0.0031**	0.203	0.0036
Stocks domain	**0.2461**	37	240	**0.0063**	0.156	0.057
Abalone	**0.0836**	65	400	**0.0250**	0.241	0.06
Mean time (s)	0.048	34.463	78.166	0.00486	0.093	0.0146

Table 15
Specification of real world large sample benchmark problems.

Data Sets	Attributes(Num Nom)	Training set	Testing set
Bank domains(1)	8(8,0)	5735	2457
Bank domains(2)	32(32,0)	5735	2817
Computer activity(1)	12(12,0)	5734	2458
Computer activity(2)	21(21,0)	5734	2458
California housing	8(8,0)	14 448	6192

Table 16
Comparison of training and testing time (s) of ORELM, ORGP, and ORSVM.

Datasets	Training time (nodes or SVs)			Testing time		
	ORELM	ORSVM	ORGP	ORELM	ORSVM	ORGP
Bank domains(1)	**2.8641(300)**	2112(2693)	–	**0.1000**	35.48	–
Bank domains(2)	**3.0078(90)**	54161(3364)	–	**0.1953**	475	–
Computer activity(1)	**1.3258(200)**	1638(2943)	–	**0.0758**	25	–
Computer activity(2)	**1.6336(200)**	2682(3161)	–	**0.0695**	21	–
California housing	**7.1211(400)**	15119(4292)	–	**0.3391**	125	–

Table 17
Comparison of testing accuracy of ORELM, ORGP and ORSVM.

Datasets	Mean zero one error			Mean absolute error		
	ORELM	ORSVM	ORGP	ORELM	ORSVM	ORGP
Bank domains(1)	0.2467	**0.2222**	–	0.2497	**0.2239**	–
Bank domains(2)	0.5927	**0.5289**	–	0.7472	**0.6858**	–
Computer activity(1)	**0.3240**	0.3250	–	**0.3498**	0.3520	–
Computer activity(2)	**0.2936**	0.3011	–	**0.3117**	0.3225	–
California housing	**0.4015**	0.4056	–	**0.4617**	0.4689	–

"–" denotes the algorithm can not get the result until the machine overlap after more than 10 hours, and thus we have to give it up.

5. Conclusions

Different from threshold-based ordinal regression framework that find a model and the corresponding thresholds of ranks, e.g., ORSVM and ORGP, we proposed an encoding-based ordinal regression framework and three ELM-based ordinal regression algorithms. In our algorithms, three output coding matrixes are firstly designed for ORELM-SingleELM, ORELM-OAA and ORELM-OAO, respectively. Then, one or more ELMs are used to construct the prediction model. Compared with threshold-based algorithms, the proposed algorithms have the following advantages. Firstly, they obtain simple implementation and extremely rapid training speed. Especially, when the data set's scalability increases, the advantage of ORELM will become more apparent. Secondly, they have the capabilities of learning in both online and batch modes. Thirdly, they can be trained on very large datasets and even non-linear data. Additionally, they are suitable for real-time tasks which need fast real-time learning and on-line updating speed, e.g., user interest tracking and real-time personalized recommendation. The disadvantage of the ORELM is that its accuracy is apparently worse than ORGP on small datasets. In future work, we will improve the ORELM's accuracy by regularization methods and statistic learning theory. Additionally, the optimization of encoding schemes will be also studied to improve the accuracy of ORELM.

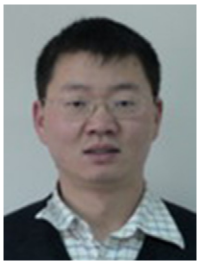
Acknowledgments

The work was partially supported by the National Science Foundation of China under Grant nos. 60825202, 60803079, 61070072 and 60633020, the National High-Tech R&D Program of China under Grant no. 2008AA01Z131, the National Key Technologies R&D Program of China under Grant nos. 2006BAK11B02, 2009BAH51B02 and 2006BAJ07B06, the Science Research Plan of

Shaanxi Provincial Department of Education under Grant no. 09JK717, and the Key Projects in the National Science under Grant no. 2009BAH51B00, China CNGI project under Grant no. CNGI-09-01-13, doctoral fund of ministry of education of china under Grant no. 20090201110060, the Fundamental Research Funds for the Central Universities under Grant no. xjj20100057.

References

[1] S. Rajaram, A. Garg, Z.S. Zhou, T.S. Huang, Classification approach towards ranking and sorting problems, Lecture Notes in Artificial Intelligence 2837 (2003) 301–312.
[2] J.L. Cheng, Z. Wang, G. Pollastri, A neural network approach to ordinal regression, in: Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'2008), 2008, pp. 1279–1284.
[3] K. Crammer, Y. Singer, Pranking with ranking, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'2001), vol. 14, 2001, pp. 641–647.
[4] J. Basilico, T. Hofmann, Unifying collaborative and content-based filtering, in: Proceedings of the 21st International Conference on Machine Learning (ICML'2004), 2004, pp. 65–72.
[5] D. Serre, in: Matrices: Theory and Applications, Springer, New York, 2002.
[6] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, G. Hullender, Learning to rank using gradient descent, in: Proceedings of International Conference on Machine Learning (ICML'2005), 2005, pp. 89–97.
[7] W. Chu, Z. Ghahramani, Gaussian processes for ordinal regression, Journal of Machine Learning Research 6 (2005) 1019–1041.
[8] A. Schwaighofer, V. Tresp, K. Yu, Hiearachical bayesian modelling with gaussian processes, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'2005), vol. 17, 2005, pp. 1209–1216.
[9] E.L. Allwein, R.E. Schapire, Y. Singer, Reducing multiclass to binary: a unifying approach for margin classifiers, Journal of Machine Learning Research 1 (2001) 113–141.
[10] A. Shashua, A. Levin, Ranking with large margin principle: two approaches, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'2003), vol. 15, 2003, pp. 937–944.
[11] W. Chu, S.S. Keerthi, Support vector ordinal regression, Neural Computation 19 (3) (2007) 792–815.
[12] S. Agarwal, D. Roth, Learnability of bipartite ranking functions, in: Proceedings of the 18th Annual Conference on Learning Theory (COLT'2005), 2005, pp. 16–31.
[13] Y. Freund, R. Iyer, R.E. Schapire, Y. Singer, An efficient boosting algorithm for combining preferences, Journal of Machine Learning Research 4 (2003) 933–969.
[14] O. Dekel, J. Keshet, Y. Singer, Log-linear models for label ranking, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'2003), vol. 16, 2003, pp. 209–216.
[15] S. Har-Peled, D. Roth, D. Zimak, Constraint classification: a new approach to multiclass classification and ranking, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'2002), vol. 15, 2002, pp. 267–280.
[16] S. Kramer, G. Widmer, B. Pfahringer, M. DeGroeve, Prediction of ordinal classes using regression trees, Fundamenta Informaticae 47 (2001) 1–13.
[17] H. Zhang, L. Jiang, J. Su, Augmenting naive bayes for ranking, in: Proceedings of the International Conference on Machine Learning (ICML'2005), 2005, pp. 1020–1027.
[18] U. Paquet, S. Holden, A. Naish-Guzman, Bayesian hierarchical ordinal regression, in: Proceedings of the International Conference on Artifical Neural Networks (ICANN'2005), 2005, pp. 267–272.
[19] E. Frank, M. Hall, A simple approach to ordinal classification, in: Proceedings of the European Conference on Machine Learning (ECML'2001), 2001, pp. 145–156.
[20] L. Li, H. Lin, Ordinal regression by extended binary classification, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'2006), vol. 20, 2006, pp. 865–872.
[21] C.J.C. Burges, R. Ragno, Q.V. Le, Learning to rank with nonsmooth cost functions, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS'2006), vol. 20, 2006, pp. 193–200.
[22] T. Joachims, Optimizing search engines using clickthrough data, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'2002), 2002, pp.133–142.
[23] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (2006) 489–501.
[24] N.-Y. Liang, P. Saratchandran, G.-B. Huang, N. Sundararajan, Classification of mental tasks from EEG signals using extreme learning machine, International Journal of Neural Systems 16 (1) (2006) 29–38.
[25] H.-J. Rong, G.-B. Huang, Y.-S. Ong, Extreme learning machine for multi-categories Classification applications, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN'2008), 2008, pp. 1709–1713.
[26] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Transactions on Neural Networks 17 (4) (2006) 879–892.
[27] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Real-time learning capability of neural networks, IEEE Transactions on Neural Networks 17 (4) (2006) 863–878.
[28] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, IEEE Transactions on Neural Network 17 (6) (2006) 1411–1423.
[29] X.J. Zhu, A.B. Goldberg, Kernel regression with order preferences, in: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI'2007), 2007, pp. 681–687.
[30] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16–18) (2007) 3056–3062.
[31] Y. Lan, Y.C. Soh, G.-B. Huang, Ensemble of online sequential extreme learning machine, Neurocomputing 72 (13-15) (2009) 3391–3395.
[32] G.-B. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16–18) (2008) 3460–3468.
[33] G.-B. Huang, M.-B. Li, L. Chen, C.-K. Siew, Incremental extreme learning machine with fully complex hidden nodes, Neurocomputing 71 (4-6) (2008) 576–583.
[34] V.N. Vapnik, in: The Nature of Statistical Learning Theory, Springer, New York, 1995.
[35] S. Haykin, in: Neural Networks: A Comprehensive Foundation,, Prentice-Hall, New Jersey, 1999.
[36] Michael E. Tipping, Sparse Bayesian learning and the relevance vector machine, Journal of Machine Learning Research 1 (2001) 211–244.


Wanyu Deng received his B.S. degree in Computer Science and Technology in 2001, his M.S. degree in Software Technology and Theory in 2004 from Northwest Polytechnical University, China. He is now a Ph.D. candidate in the Department of Computer Science and Technology in Xi'an Jiaotong University, China. His research interests include machine learning, collaborative filtering and personalized service. He has participated in several national research projects and got FUJIXEROX scholarships from the Xi'an Jiaotong University. He is the author or co-author of more than 10 refereed international journal and conference papers covering topics of collaborative filtering, personalized service and artificial intelligence.

Qinghua Zheng received his B.S. degree in computer software in 1990, his M.S. degree in computer organization and architecture in 1993, and his Ph.D. degree in system engineering in 1997, all from the Xi'an Jiaotong University, China. He is a professor with the Department of Computer Science and Technology in Xi'an Jiaotong University. He serves as the dean of the Department of Computer Science and Technology, and vice dean of E-Learning School of Xi'an Jiaotong University. His research areas include multimedia distance education, intelligent e-learning theory and algorithm, and computer network security. He was a Postdoctoral Researcher in Harvard University from February 2002 to October 2002 and a Visiting Professor Research in Hong Kong University from November 2004 to January 2005. He has published more than 90 papers, and held 13 patents. He won the National Science Fund for Distinguished Young Scholars in 2008. He got the First Prize for National Teaching Achievement, State Education Ministry in 2005 and the First Prize for Scientific and Technological Development of Shanghai City and Shaanxi Province in 2004 and 2003, respectively. Till now, he has held more than 10 national fund projects, including National Natural Science Foundation of China, National 863 Major Programs, and Key (Key grant) Projects of Chinese Ministry of Education, etc. He is the member of the Education Subcommittee of National Standard Committee of Information Technology and a member of the IEEE.

Shiguo Lian got his B.S. degree and Ph.D. from the Nanjing University of Science and Technology, China. He was a research assistant in the City University of Hong Kong in 2004. Since July 2005, he has been a Research Scientist with France Telecom R&D (Orange Labs) Beijing. He is the author or co-author of more than 80 refereed international journal and conference papers covering topics of secure multimedia communication, intelligent multimedia analysis, and ubiquitous communication. He is the author/editor of 5 published books, has contributed 15 chapters to books and held 14 filed patents. He got the Nomination Prize of "Innovation Prize in France Telecom" and "Top 100 Doctorate Dissertation in Jiangsu Province" in 2006. He is a member of IEEE ComSoc Communications & Information Security Technical Committee (CIS TC), IEEE Multimedia Communications Technical Committee (MMTC), and IEEE Technical

Committee on Nonlinear Circuits and Systems (TC NCAS). He is on the editor board of several international journals, and is the guest editor of Informatica, Soft Computing, Neural Network World, Applied Soft Computing, Intelligent Automation and Soft Computing (AutoSoft), Telecommunication Systems, and Computer Communications, etc. He is in the organization committee of some refereed conferences, and the reviewer of many refereed international journals and magazines.

Lin Chen received his B.S. degree in Computer Science and Technology in 1999 from the Shaanxi Normal University, his M.S. degree in Software Technology and Theory in 2005 from the Northwest Polytechnical University, China. She is a lecturer in the Department of Computer Science and Technology in Xi'an Institute of Posts and Telecommunications, China. His research interests include collaborative filtering and personalized service.