Neurocomputing 74 (2011) 2438–2443

Contents lists available at ScienceDirect

Neurocomputing

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

An OS-ELM based distributed ensemble classification framework in P2P networks

Yongjiao Sun [a,b,*], Ye Yuan [a,b], Guoren Wang [a,b]

[a] Key Laboratory of Medical Image Computing (NEU), MOE, China
[b] College of Information Science and Engineering, Northeastern University, Shenyang 110004, China

ARTICLE INFO

Available online 20 May 2011

Keywords:
Peer-to-Peer (P2P)
Extreme learning machine
OS-ELM
Ensemble classification
Parallel ensemble classification
Quad-tree

ABSTRACT

Although classification in centralized environments has been widely studied in recent years, it is still an important research problem for classification in P2P networks due to the popularity of P2P computing environments. The main target of classification in P2P networks is how to efficiently decrease prediction error with small network overhead. In this paper, we propose an OS-ELM based ensemble classification framework for distributed classification in a hierarchical P2P network. In the framework, we apply the incremental learning principle of OS-ELM to the hierarchical P2P network to generate an ensemble classifier. There are two kinds of implementation methods of the ensemble classifier in the P2P network, *one-by-one* ensemble classification and *parallel* ensemble classification. Furthermore, we propose a data space coverage based peer selection approach to reduce high the communication cost and large delay. We also design a two-layer index structure to efficiently support peer selection. A peer creates a local *Quad-tree* to index its local data and a super-peer creates a global *Quad-tree* to summarize its local indexes. Extensive experimental studies verify the efficiency and effectiveness of the proposed algorithms.

1. Introduction

It is a very important problem to classify and predict data on distributed systems due to higher application demands, for example, automatically organizing Web documents in a Peer-to-Peer (P2P) environment [5]. Although data classifications in centralized environments have been studied extensively in past few years, there still exist several challenging issues for data classification in a P2P network, such as *scalability*, *anytimeness*, and *asynchronism* [6]. Recently, a significant amount of distributed classification techniques have been proposed for classification in P2P networks [1–4]. The key idea of these articles is to learn an ensemble of classifiers on the local training data by combining all the classifiers to achieve high classification accuracies on unseen test data examples.

In this paper, we present a novel P2P ensemble classification framework based on OS-ELM to get an ensemble classifier by combining the classifiers learnt from peers in the P2P network. OS-ELM [7] is an online sequential *extreme learning machine* (ELM) that is an easy-to-use and effective learning algorithm for single hidden layer feedforward neural networks (SLFNs) [8–11]. ELM can only be used for batch learning, while OS-ELM can learn data one-by-one or chunk-by-chunk with varying or fixed chunk length, and the input weights and biases of OS-ELM with additive nodes are randomly generated and the output weights are analytically determined. As such, we can generate a distributed classifier by directly applying the incremental learning principle of OS-ELM.

There are two ways to achieve distributed ensemble learning in a P2P network, *one-by-one* ensemble learning and *parallel* ensemble learning. In the *one-by-one* ensemble learning, each classifier is learnt from all the data one peer by one peer. In the *parallel* ensemble learning, all the classifiers are learnt from all the data in parallel.

However, the *one-by-one* approach has a large network delay, while the *parallel* method has a high communication cost. Therefore, we propose a data space coverage based peer selection approach to solve these two problems. We also design a two-layer index structure to efficiently support peer selection. A peer creates a local *Quad-tree* to index its local data and a super-peer creates a global *Quad-tree* to summarize its local indexes.

The rest of this paper is organized as follows: Section 2 presents the distributed classification framework based on OS-ELM; Section 3 describes the implementation of distributed classification framework; Section 4 proposes the operations of peer dynamic; Section 5 gives the experimental results; and Section 6 concludes the paper and gives the future work.

* Corresponding author at: Key Laboratory of Medical Image Computing (NEU), MOE, Shenyang 110004, China.
E-mail address: gence23@gmail.com (Y. Sun).

0925-2312/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2010.12.040

2. A distributed classification framework based on OS-ELM

ELM is a good learning method to classify data due to its many advantages, such as good generalization performance as well as improving the learning speed of neural network [8,16–18], maximizing the separating margin, and minimizing the training errors [12]. In the traditional P2P distributed classification methods [1–4], an ensemble classifier can be obtained by a two-step approach. The local classifiers are first obtained by training local data in each peer, then an ensemble classier can be obtained by combining these local classifiers. However, this kind of ensemble classifier has a lower classification accuracy since each local classifier only learns partial data. In order to solve this problem, we need an incremental learning method that is the main feature of OS-ELM. OS-ELM can learn data arriving or chunk-by-chunk with varying chunk size. Based on this feature, OS-ELM can learn all the data in a P2P network and get a higher classification accuracy.

2.1. Review of OS-ELM

Based on batch ELM, OS-ELM develops SLFNs with both additive and radial basis function (RBF) hidden nodes. OS-ELM can learn data one-by-one or chunk-by-chunk with varying or fixed chunk length, and the input weights and biases of additive nodes or centers and impact factors of RBF nodes are randomly generated and the output weights are analytically determined. For N arbitrary distinct samples $(\mathbf{x}_i,\mathbf{t}_i)$, where $\mathbf{x}_i=[x_{i1},x_{i2},\ldots,x_{in}]^T\in\mathbf{R}^n$ and $\mathbf{t}_i=[t_{i1},t_{i2},\ldots,t_{im}]^T\in\mathbf{R}^m$, since standard SLFN with $\tilde{N}$ hidden nodes can approximate these N samples with zero error [9], the output of a SLFN with hidden nodes can be represented by

$$f_{\tilde{N}}(\mathbf{x})=\sum_{i=1}^{\tilde{N}}\beta_i G(\mathbf{a}_i,b_i,\mathbf{x}),\quad \mathbf{x}\in\mathbf{R}^n,\ \mathbf{a}_i\in\mathbf{R}^n \tag{1}$$

where $\mathbf{a}_i$ and b_i are the learning parameters of hidden nodes, β_i is the output weight, and $G(\mathbf{a}_i,b_i,\mathbf{x})$ is the output of the ith hidden node with respect to the input $\mathbf{x}$. For additive hidden node, $G(\mathbf{a}_i,b_i,\mathbf{x})=g(\mathbf{a}_i\cdot\mathbf{x}+b_i)$, $b_i\in R$, where $\mathbf{a}_i$ is the input weight, b_i is the bias of the ith hidden node and $\mathbf{a}_i\cdot\mathbf{x}$ is the inner product. For RBF hidden node $G(\mathbf{a}_i,b_i,\mathbf{x})=g(b_i\|\mathbf{x}-\mathbf{a}_i\|)$, $b_i\in R^+$, where $\mathbf{a}_i$ and b_i are the center and impact factor of ith RBF node. R^+ is the set of all positive real values.

Through selecting the type of node, the activation function, and the number $\tilde{N}$ of hidden nodes, the OS-ELM can be implemented with data $\aleph=\{(\mathbf{x}_i,\mathbf{t}_i)|\mathbf{x}_i\in\mathbf{R}^n,\mathbf{t}_i\in\mathbf{R}^m,i=1,\ldots,N\}$ arrived one-by-one or chunk-by-chunk. The two-step OS-ELM algorithm is presented as follows:

Initialization phase: From the given training set $\aleph=\{(\mathbf{x}_i,\mathbf{t}_i)|\mathbf{x}_i\in\mathbf{R}^n,\mathbf{t}_i\in\mathbf{R}^m,\ i=1,\ldots,N\}$ a small chunk of training data $\aleph_0=\{(\mathbf{x}_i,\mathbf{t}_i)\}_{i=1}^{N_0}$ is used to initialize the learning, where $N_0\geq\tilde{N}$.

(1) For additive hidden nodes, assign random input weights $\mathbf{a}_i$ and bias b_i; for RBF hidden nodes, input weights is center $\mathbf{a}_i$ and impact factor is b_i, where $i=1,\ldots,\tilde{N}$.

(2) Calculate the initial hidden layer output matrix $\mathbf{H}_0$.

$$\mathbf{H}_0=\begin{bmatrix} G(\mathbf{a}_1,b_1,\mathbf{x}_1) & \cdots & G(\mathbf{a}_{\tilde{N}},b_{\tilde{N}},\mathbf{x}_1)\\ \vdots & \cdots & \vdots\\ G(\mathbf{a}_1,b_1,\mathbf{x}_{N_0}) & \cdots & G(\mathbf{a}_{\tilde{N}},b_{\tilde{N}},\mathbf{x}_{N_0})\end{bmatrix}_{N_0\times\tilde{N}} \tag{2}$$

(3) Estimate the initial output weight $\beta^{(0)}$:
Considering using the batch ELM algorithm, the solution to minimizing $\|\mathbf{H}_0\beta-\mathbf{T}_0\|$ is given by $\beta^{(0)}=\mathbf{P}_0\mathbf{H}_0^T\mathbf{T}_0$, where $\mathbf{P}_0=(\mathbf{H}_0^T\mathbf{H}_0)^{-1}=\mathbf{K}_0^{-1}$ and $\mathbf{T}_0=[\mathbf{t}_1,\ldots,\mathbf{t}_{N_0}]^T$.

(4) Set $k=0$, where k is the number of chunks that is trained currently.

Sequential learning phase: Present the $(k+1)$th chunk of new observations $\aleph_{k+1}=\{(\mathbf{x}_i,\mathbf{t}_i)\}_{i=(\sum_{j=0}^{k}N_j)+1}^{\sum_{j=0}^{k+1}N_j}$, where N_{k+1} denotes the number of observations in the kth chunk.

1. Calculate the partial hidden layer output matrix $\mathbf{H}_{k+1}$,

$$\mathbf{H}_{k+1}=\begin{bmatrix} G(\mathbf{a}_1,b_1,\mathbf{x}_{(\sum_{j=0}^{k}N_j)+1}) & \cdots & G(\mathbf{a}_{\tilde{N}},b_{\tilde{N}},\mathbf{x}_{(\sum_{j=0}^{k}N_j)+1})\\ \vdots & \cdots & \vdots\\ G(\mathbf{a}_1,b_1,\mathbf{x}_{\sum_{j=0}^{k+1}N_j}) & \cdots & G(\mathbf{a}_{\tilde{N}},b_{\tilde{N}},\mathbf{x}_{\sum_{j=0}^{k+1}N_j})\end{bmatrix}_{N_{k+1}\times\tilde{N}} \tag{3}$$

where $\mathbf{H}_{k+1}$ is calculated by the $(k+1)$th chunk of data $\aleph_{k+1}$.

2. Set $\mathbf{T}_{k+1}=\left[\mathbf{t}_{(\sum_{j=0}^{k}N_j)+1},\ldots,\mathbf{t}_{(\sum_{j=0}^{k+1}N_j)}\right]^T_{N_{k+1}\times m}$.

3. Calculate the output weight $\beta^{(k+1)}$. We have

$$\mathbf{K}_{k+1}=\mathbf{K}_k+\mathbf{H}_{k+1}^T\mathbf{H}_{k+1} \tag{4}$$

$$\beta^{(k+1)}=\beta^{(k)}+\mathbf{K}_{k+1}^{-1}\mathbf{H}_{k+1}^T(\mathbf{T}_{k+1}-\mathbf{H}_{k+1}\beta^{(k)}) \tag{5}$$

From Eq. (5), we find that $\mathbf{K}_{k+1}^{-1}$ rather than $\mathbf{K}_{k+1}$ is used to compute $\beta^{(k+1)}$ from $\beta^{(k)}$. The update formula for $\mathbf{K}_{k+1}^{-1}$ is derived using the Woodbury formula [12].

$$\begin{aligned}\mathbf{K}_{k+1}^{-1}&=(\mathbf{K}_k+\mathbf{H}_{k+1}^T\mathbf{H}_{k+1})^{-1}\\ &=\mathbf{K}_k^{-1}-\mathbf{K}_k^{-1}\mathbf{H}_{k+1}^T(\mathbf{I}+\mathbf{H}_{k+1}\mathbf{K}_k^{-1}\mathbf{H}_{k+1}^T)^{-1}\times\mathbf{H}_{k+1}\mathbf{K}_k^{-1}\end{aligned} \tag{6}$$

$\beta^{(k+1)}$ can be written by $\mathbf{P}_{k+1}$,

$$\mathbf{P}_{k+1}=\mathbf{P}_k-\mathbf{P}_k\mathbf{H}_{k+1}^T(\mathbf{I}+\mathbf{H}_{k+1}\mathbf{P}_k\mathbf{H}_{k+1}^T)^{-1}\mathbf{H}_{k+1}\mathbf{P}_k \tag{7}$$

$$\beta^{(k+1)}=\beta^{(k)}+\mathbf{P}_{k+1}\mathbf{H}_{k+1}^T(\mathbf{T}_{k+1}-\mathbf{H}_{k+1}\beta^{(k)}) \tag{8}$$

where $\mathbf{P}_{k+1}=\mathbf{K}_{k+1}^{-1}$.

4. Set $k=k+1$. Go to Sequential Learning Phase (1).

2.2. OS-ELM for distributed classification

The OS-ELM algorithm can handle data arriving chunk-by-chunk with varying chunk size [7]. By viewing the data of a peer as a chunk of the training dataset, we can easily design a naïve approach to generate a distributed classifier based on OS-ELM. Similar to OS-ELM, a classifier is initialized in a peer by learning the local data, and is propagated to other peers to sequentially learn the remaining data in the network. It is obvious that the naïve approach is not feasible due to a very large network delay. Therefore, we present an effective data space coverage based peer selection approach to avoid the problem.

In a P2P network, a peer manages a data space in which all the local data are located, and there are overlaps among the data spaces. In order to reduce the network delay of the naïve approach, we only need to find minimal peers that cover the whole data space to generate an ensemble classifier. We design a two-layer index structure to efficiently support peer selection. A common peer creates a local *Quad-tree* to index its local data and a super-peer creates a global *Quad-tree* to summarize its local indexes. Next, we introduce the peer selection method based on the two-layer index.

The peer selection can be done in two steps. (1) A super-peer selects minimal common peers based on its *Quad-tree* index in a