Parallel Ensemble of Online Sequential Extreme Learning Machine Based on MapReduce

Shan Huang, Botao Wang, Junhao Qiu, Jitao Yao, Guoren Wang, and Ge Yu

College of Information Science and Engineering,
Northeastern University, Liaoning, Shenyang, China 110004
huangshan.neu@gmail.com,{wangbotao,wanggr,yuge}@ise.neu.edu.cn,
lqqiujunhao@163.com,tao00800@126.com,

Abstract. In this era of big data, analyzing large scale data efficiently and accurately has become a challenge problem. Online sequential extreme learning machine is one of ELM variants, which provides a method to analyze data. Ensemble method provides a way to learn data more accurately. MapReduce provides a simple, scalable and fault-tolerant framework, which can be utilized for large scale learning. In this paper, we propose an ensemble OS-ELM framework which supports ensemble methods including Bagging, subspace partitioning and cross validating. Further we design a parallel ensemble of online sequential extreme learning machine (PEOS-ELM) algorithm based on MapReduce for large scale learning. PEOS-ELM algorithm is evaluated with real and synthetic data with the maximum number of training data 5120K and the maximum number of attributes 512. The speedup of this algorithm can reach as high as 40 on a cluster with maximum 80 cores. The accuracy of PEOS-ELM algorithm is at the same level as that of ensemble OS-ELM running on a single machine, which is higher than that of the original OS-ELM.

Keywords: Parallel learning, Ensemble, Extreme Learning Machine, MapReduce, Sequential Learning.

1 Introduction

In this era of big data, analyzing large scale data efficiently and accurately has become a challenge problem. There are often hidden noises behind large scale data. Ensemble methods are proposed to eliminate the influence of the noises. Generally, ensemble methods can reach higher accuracy dealing with the same data set [9]. Ensemble methods usually train several ensemble members and combine the output of these ensemble members to generate the final result. However, this approach would lead in more calculations, and it is hard to analyze large scale data efficiently. Extreme learning machine (ELM) was proposed based on single-hidden layer feed-forward neural networks (SLFNs) [5], and has been verified to have high learning speed as well as high accuracy [3]. It has also been proved that ELM has have universal approximation capability and classification capability [4]. Online sequential extreme learning machine (OS-ELM) [7] is one

J. Cao et al. (eds.), *Proceedings of ELM-2014 Volume 1*,
Proceedings in Adaptation, Learning and Optimization 3, DOI: 10.1007/978-3-319-14063-6_3

of ELM variants that supports online sequential learning. OS-ELM can learn data chunk by chunk with fixed or varying sizes instead of batch learning. There are works researching on combining ensemble methods and ELM [6], [8], [12]. However, these algorithms mainly focus on the accuracy, but they are inefficient to learn large scale data.

MapReduce framework is a well-known framework for large scale data processing and analyzing on a large cluster of commodity machines. There are works research on parallelizing ELM and OS-ELM to improve learning speed [2], [11], [10]. However, ensemble methods are not taken into consideration in these works, so these works are not suitable for large scale data learning due to the accuracy limitation.

In this paper, we present a parallel ensemble of online sequential extreme learning machine (PEOS-ELM) algorithm based on MapReduce for large scale data processing and analyzing. This algorithm supports the most common ensemble methods such as Bagging, subspace partitioning and cross validating. This algorithm splits data according to user customization and calculates hidden layer output matrix of OS-ELM in Map phase. In Reduce phase, the ensemble members finish the remaining training work in parallel. We also test PEOS-ELM algorithm real and synthetic data, and the results show that PEOS-ELM has good scalability and the accuracy of this algorithm is at the same level with that of ensemble OS-ELM running on a single machine.

The remainder of this paper is organized as follows. Section 2 introduces an ensemble OS-ELM framework. Section 3 proposes parallel ensemble of online sequential learning machine algorithm. Section 4 evaluates the PEOS-ELM algorithm with real data and synthetic data, and section 5 concludes the paper.

2 Ensemble OS-ELM Framework

Figure 1 shows the ensemble OS-ELM framework. This framework considers ensemble methods (Bagging, subspace partitioning and cross validation) as well as training phase and testing phase of OS-ELM. $(X_{m,k}, T_{m,k})$ represents data chunks for ensemble member m, where $X_{m,k}$ represents the attributes set and $T_{m,k}$ represents the set of tags in which class the instances belong to corresponding to $X_{m,k}$. There are two ways of using the training data, one is used for training OS-ELM and the other is used for validating OS-ELM. The superscript of $(X_{m,k}, T_{m,k})$ in the figure, marks these two ways of use.

In the framework, data are processed in the following steps.

1. The $(X_{m,k}, T_{m,k})$ used for training and validating are generated by taking with replacement from training data. This procedure is needed by Bagging.
2. The subspace sets of $(X_{m,k}, T_{m,k})$ are generated. This procedure is needed by subspace partitioning.
3. All OS-ELMs are sequentially trained using the subspace sets. This phase follows the way of OS-ELM.
4. Data generated for validating are used to valid the trained OS-ELMs. This procedure is needed by cross validating.

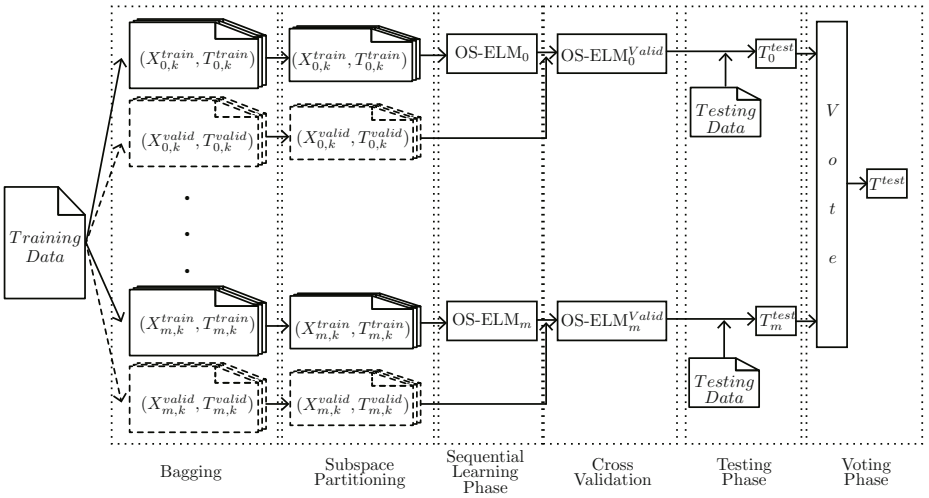

Fig. 1. Ensemble OS-ELM Framework

5. T_m^{test} are generated by OS-ELMs with testing data. This phase follows the way of testing phase of OS-ELM.
6. The T_m^{test} are processed by vote procedure to generated the final T^{test}. This procedure is needed by Bagging, subspace partitioning and cross validating.

The ensemble OS-ELM algorithm can be divided into three mainly phases, initialization phase, sequential learning phase and testing phase. In initialization phase, the initial parameters of OS-ELM and subspace for each ensemble member are generated. In sequential learning phase, several ensemble members are trained in the same way with OS-ELM [7]. In testing phase, the final result is created according to all the results from the ensemble members.

3 Parallel Ensemble of OS-ELM

3.1 Basic Idea

The goal of the parallel ensemble of online sequential extreme learning machine (PEOS-ELM) is to improve the performance of sequential learning phase in EOS-ELM using parallel techniques for large scale learning.

Figure 2 shows the matrix calculation dependency relationships among the matrices in sequential learning phase of EOS-ELM. One dependency is denoted as "$\rightarrow$", which means the matrix at arrow side depends on the matrix at the other side. That is to say, the calculation of matrix at the arrow side cannot start until the calculation of matrix at the other side finished. The matrices which do not depend on any other matrices can be calculated in parallel. Based on the above observations, our basic ideas can be summarized as follows:

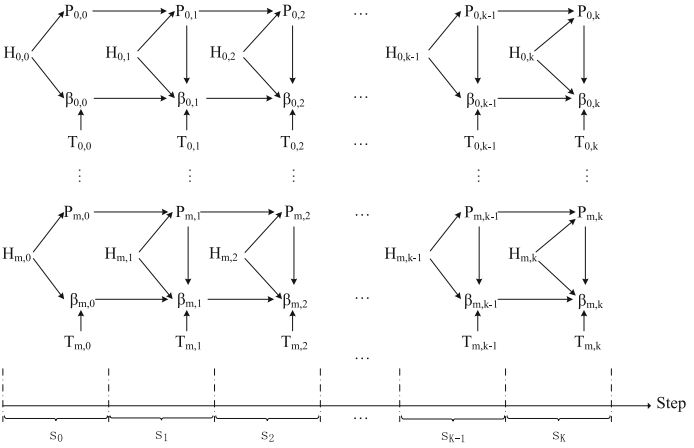

Fig. 2. Dependency relationships of matrix calculations in EOS-ELM

1. The calculation of $\mathbf{H}_{m,k}$ $((0 \leq m \leq M), (0 \leq k \leq K))$ depends on none of $\mathbf{T}_{m,k+1}$, $\mathbf{P}_{m,k+1}$ and $\beta_{m,k}$, besides this matrix can be calculated immediately when its related training data are available. This means that $\mathbf{H}_{m,k}$ can be calculated in parallel when K train data chunks are available for M ensemble members. This can be done in Map phase of MapReduce framework.
2. The calculation of $\beta_{m,k+1}$ is dependent on the calculation of $\mathbf{H}_{m,k+1}$, $\mathbf{T}_{m,k+1}$, $\mathbf{P}_{m,k+1}$ and $\beta_{m,k}$, but not dependent on matrix calculations for another ensemble member $j(0 \leq j \neq m \leq M)$. So the calculation of $\beta_{m,k+1}$ for different ensemble members can also be executed in parallel. This can be done by Reduce phase of MapReduce framework.

3.2 Parallel EOS-ELM

It is preferred to execute sequential learning phase of EOS-ELM in parallel on MapReduce framework as it is the most time consuming phase. The parallel ensemble of online sequential extreme learning machine algorithm uses one MapReduce job to train ensemble members.

Procedure 1 shows the *map*() procedure of PEOS-ELM. For each ensemble member, the input sample is possibly used for normal calculation (line 2-11) and validating (line 12-13).

When the sample is chosen for normal calculation, a buffer is used to store samples and a counter is used to count the number of samples in buffer (line 3-5). There are several data processing steps (line 6-11) when buffer is full to extract subspace, calculate $\mathbf{H}_{m,k}$, calculate $\mathbf{T}_{m,k}$, generate key-value pair, clear the counter and increase k. In the key-value pair, the *key* is composed with ensemble member ID m, blockID k and Tag while the value is made up of $\mathbf{H}_{m,k}$ and $\mathbf{T}_{m,k}$.

Procedure 1. PEOS-ELM map()

Input: (Key, Value): Key is the offset in bytes, Value is a sample pair $(x_i, t_i) \in (X_k^{train}, T_k^{train})$ where $0 \leq i \leq |\ (X_k^{train}, T_k^{train})\ |$;
Result: m: Ensemble member ID;
k: blockID;
tag: marks whether output is used for normal calculation or validating;
$\mathbf{H}_{m,k}$: Output weight;
$\mathbf{T}_{m,k}$: Observation value vector;

```
1  for m=0 to M do
2      if chooseForThisMember() then
3          add to block_m;
4          count_m++;
5          if count_m ≥ BLOCK then
6              block_m=GetSpace(block_m);
7              H_{m,k}=calcH(block_m);
8              T_{m,k}=calcT(block_m);
9              output((m,k_m,NormalTag), (H_{m,k}, T_{m,k}));
10             count_m = 0;
11             k_m++;
12     if chooseForValid() then
13         calucateForValid();
```

When the sample is chosen for normal calculation, similar operations are applied with those for normal calculation except that the output key is marked with $ValidTag$ instead of $NormalTag$ to facilitate distinguishing them later in Reduce phase. We briefly express it as $calucateForValid()$.

Procedure 2 shows the $reduce()$ procedure of PEOS-ELM. The output results of Map which belong to the same ensemble member are partitioned to the same Reducer and then sorted by tag and k. When the set of key-value pairs reaches to $reduce()$ procedure, the parameters composed in key are firstly resolved (line 1-2) Then the key-value pair is processed differently according to the tag. If the tag is $NormalTag$, the parameters for ensemble member m are initialized if it has not been initialized (line 4-6) and then $\mathbf{H}_{m,k+1}$ and $\mathbf{T}_{m,k+1}$ composed in $value$ are resolved (line 7-8). After that, the $\mathbf{P}_{m,k+1}$ and $\beta_{m,k+1}$ are updated according to the equations (line 9-10). If the tag is $ValidTag$, it means that this key-value pair is used for validating. The $\mathbf{H}_{m,k}^{valid}$ and $\mathbf{T}_{m,k}^{valid}$ are firstly resolved from $value$ (line13-14) and then used for cross validating (line 15).

3.3 Cost Model

The cost of PEOS-ELM algorithm mainly has four parts, (1) cost of starting a MapReduce job, (2) cost of Map procedure, (3) cost of Reduce procedure and (4) cost of data transmitting between Map and Reduce.

As the number of cores in cluster increases while the other parameters keep the same, the cost of Map procedure and cost of Reduce procedure would be

Procedure 2. PEOS-ELM reduce()

Input:
Set of (*key*, *value*): *key* is a combination of m, k and tag . *value* is a vector pair (H_{kb}, T_{kb});
Result: β_m: output weight vector (corresponding to $\beta_{m,k}$).

```
1  m = getm(key);
2  tag = gettag(key);
3  if tag=NormalTag then
4      if firstRun=true then
5          initMember(m);
6          firstRun=false;
7      H_{m,k+1} = getH(value);
8      T_{m,k+1} = getT(value);
9      P_{m,k+1} = P_{m,k} − P_{m,k}H^T_{m,k+1}(I + H_{m,k+1}P_{m,k}H^T_{m,k+1})^{-1}H_{m,k+1}P_{m,k};
10     β_{m,k+1} = β_{m,k} + P_{m,k+1}H^T_{m,k+1}(T_{m,k+1} − H_{m,k+1}β_{m,k});
11 if tag=ValidTag then
12     for k = 0 to K do
13         H^{valid}_{m,k} = getH(value);
14         T^{valid}_{m,k} = getT(value);
15         CrossValid(H^{valid}_{m,k}, T^{valid}_{m,k});
```

significantly reduced, the cost of starting a MapReduce job and cost of data transmitting between Map and Reduce are all fixed for MapReduce applications. The reason is that all the calculations are equally distributed to the cores and be executed in parallel. So the PEOS-ELM has good scalability.

4 Experimental Evaluation

4.1 Experimental Setup

In this section POS-ELM indicates parallel online sequential learning machine algorithm in our previous work [10] that train each ensemble member one by one. PEOS-ELM-B, PEOS-ELM-S and PEOS-ELM-C represent PEOS-ELM algorithm for Bagging, subspace partitioning and cross validating, respectively.

PEOS-ELM algorithm is evaluated with real data and synthetic data. The real data sets (gisette[1], mnist[1]) are mainly used to test training accuracy and testing accuracy. The specification of real data is shown in Table 1.

The synthetic data are only used for scalability test, which are generated by extending based on Flower[2]. The volume and attributes of training data are extended by duplicating the original data in a round-robin way. The parameters used in scalability test are summarized in Table 2. In the experiments, all the parameters use default values unless otherwise specified.

[1] Downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`
[2] Downloaded from `http://www.datatang.com/data/13152`

Table 1. Specifications of real data

Data Set	#attributes	#class	#training data	#testing data	Size of test data (KB)
gisette	5000	2	6000	1000	128137.240
mnist	780	10	60000	10000	176001.138

Table 2. Specifications of synthetic data and running parameters for scalability test

Parameter	Value range	Default value
#training data	640k, 1280k, 2560k, 5120k	640k
#attributes	64, 128, 256, 512	64
#cores	10, 20, 40, 80	80
#ensemble member	10, 20, 40, 80	80

In scalability test, all of the data are pushed to each of the ensemble member. That is to say the largest data set in Tabel 2 is as large as 640 instances * (80*512) attributes or (80*5120)instances *64 attributes for normal use. For PEOS-ELM-C, we randomly choose 80% of data for training and another 20% are used for validating.

PEOS-ELM algorithm is implemented in Java 1.6. The universal java matrix package (UJMP) [1] versioned 0.2.5 is used for matrix storage and processing, and Hadoop versioned 0.20.2 is chosen as our MapReduce platform. A Hadoop cluster deployed on 5 servers is used in our experiment. Each server has two Xeon E5-2620 CPUs (6 cores *2 threads), 32G memory, 4*2T hard disk. The servers are connected with Gigabit network. The servers are all running Centos6.4 64 bits Linux operating system. The number of hidden layer node is 25 and the activation function is $g(x) = \frac{1}{1+e^{-x}}$.

4.2 Evaluation Results

Accuracy Test

Table 3 shows the results of accuracy and performance tests with real data. It can be found that the training accuracy and testing accuracy of EOS-ELM are higher than those of OS-ELM. This verifies that ensemble method is useful to increase the learning accuracy. Compared with EOS-ELM, training time of PEOS-ELM reduces while keeps the accuracy at the same level. This result demonstrates that PEOS-ELM can learn large scale data accurately and efficiently.

Scalability Test

Figure 3 shows the scalability (speedup) of PEOS-ELM and POS-ELM. The speedup of PEOS-ELM can reach to as high as 40 whereas the speedup of POS-ELM can only reach to 1.3. The reason for this is that the there are several reduce

Table 3. Evaluation results with real data

Data Set	Algorithm	Training time (s)	Training accuracy	Testing accuracy
gisette	OS-ELM	175.796	0.682	0.643
	EOS-ELM	311.6	0.87	0.869
	PEOS-ELM-B	66.745	**0.882**	0.87
	PEOS-ELM-S	**36.793**	0.876	**0.881**
	PEOS-ELM-C	64	0.767	0.773
mnist	OS-ELM	188.765	0.621	0.634
	EOS-ELM	178.3	0.764	0.78
	PEOS-ELM-B	56.988	**0.789**	**0.798**
	PEOS-ELM-S	**34.633**	0.761	0.778
	PEOS-ELM-C	70.753	0.784	0.796

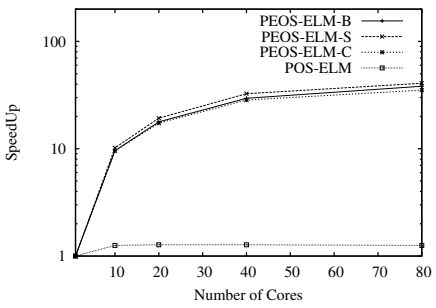

Fig. 3. Speedup of PEOS-ELM with regard to different number of cores

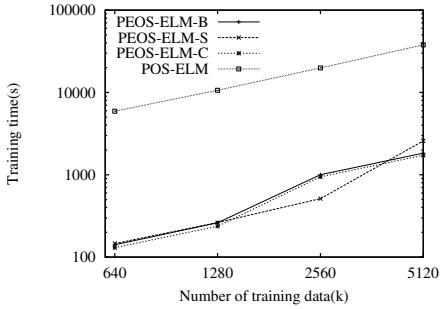

Fig. 4. Scalability of PEOS-ELM with regard to the number of training data

tasks running in parallel to sequentially calculate $\beta_{m,k}$ for different ensemble members whereas there is only one reduce task to calculate β in POS-ELM algorithm. The high speedup is consistent with the cost model.

The speedup decreases as the number of cores increases. This is due to the task scheduling cost and the bottleneck of memory and I/Os.

Figure 4 shows the scalability of PEOS-ELM and POS-ELM algorithm with regard to the number of training data. It can be found that PEOS-ELM has good scalability with regard to the number of training data. This is because the calculations are equally distributed to different map tasks and reduce tasks.

It also can be found from Figure 4 that the performance of PEOS-ELM for different ensemble methods outperforms that of POS-ELM. There are several reasons for this. First, for PEOS-ELM, the calculation of $\beta_{m,k}$ are running in parallel in Reduce phase, while in POS-ELM this calculation is running on one reduce task. Second, the sequential learning data sets are read once and push to each ensemble member in memory, while the POS-ELM read data many times. Third, as there is trade off to run a MapReduce job, the cost of running several MapReduce jobs for POS-ELM is higher than that of running one MapReduce job for PEOS-ELM.

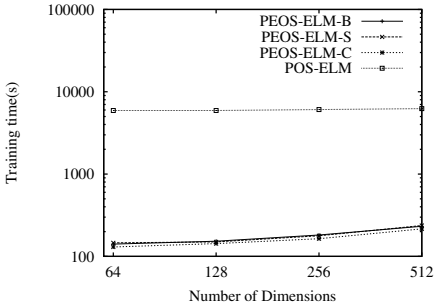

Fig. 5. Scalability of PEOS-ELM with regard to the number of attributes

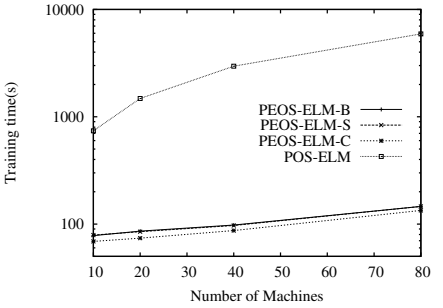

Fig. 6. Scalability of PEOS-ELM with regard to the number of ensemble members

Figure 5 shows the scalability of PEOS-ELM with regard to the number of attributes. It can be found that the training time of PEOS-ELM increased slowly as the number of attributes increases. On reason for this is the equally distributed calculations among map tasks and reduce tasks, while another reason is that the cost for transmitting data between Map phase and Reduce phased does not increases with the number of attributes.

It can also be found from Figure 5 that the training time of PEOS-ELM algorithms for different ensemble methods are nearly the same. This shows that the PEOS-ELM algorithm is suitable for different ensemble methods to analyze large scale data.

Figure 6 shows the scalability of PEOS-ELM with regard to the number of ensemble members. It can be found that the training time scales linearly as the number of ensemble members increases. The reason for this is that all of the matrix calculations are evenly distributed to tasks that running in parallel. This also shows that PEOS-ELM is efficient to train many ensemble members.

5 Conclusions

In this paper, a parallel ensemble of online sequential extreme learning machine (PEOS-ELM) algorithm has been proposed for large scale learning. The basic idea of this algorithm is to parallelize the calculation of $\mathbf{H}_{m,k}$ in Map phase and $\beta_{m,k}$ in reduce phase for different ensemble members.

The algorithm is implemented on MapReduce framework. PEOS-ELM algorithm for Bagging, subspace partitioning and cross validating are evaluated with real and synthetic data with the maximum number of training data 5120k and the maximum number of attributes 512 for each ensemble member. The experimental results show that the accuracy of PEOS-ELM for different ensemble methods are at the same level as that of EOS-ELM, and it has a good scalability with the number of training data and number of attributes. The speedup of PEOS-ELM can reaches as high as 40 on a cluster with maximum 80 cores.

Compared with EOS-ELM and POS-ELM, PEOS-ELM can be used to learn large scale data efficiently and accurately.

Acknowledgments. This research was partially supported by the National Natural Science Foundation of China under Grant No. 61173030, 61272181, 61272182; and the Public Science and Technology Research Funds Projects of Ocean Grant No. 201105033; and the National Basic Research Program of China under Grant No. 2011CB302200-G; and the 863 Program under Grant No.2012AA011004.

References

1. Arndt, H., Bundschus, M., Naegele, A.: Towards a next-generation matrix library for java. In: 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2009, vol. 1, pp. 460–467. IEEE (2009)
2. He, Q., Shang, T., Zhuang, F., Shi, Z.: Parallel extreme learning machine for regression based on mapreduce. Neurocomput. 102, 52–58 (2013)
3. Huang, G.-B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing 70, 3056–3062 (2007)
4. Huang, G.-B., Zhou, H., Ding, X., Zhang, R.: Extreme learning machine for regression and multiclass classification. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 42(2), 513–529 (2012)
5. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine. In: Technical Report ICIS/03/2004. School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore (January 2004)
6. Lan, Y., Soh, Y.C., Huang, G.-B.: Ensemble of online sequential extreme learning machine. Neurocomputing 72(13), 3391–3395 (2009)
7. Liang, P.N.-Y., Huang, G.-B., Saratchandran, Sundararajan, N.: A fast and accurate online sequential learning algorithm for feedforward networks. IEEE Transactions on Neural Networks 17(6), 1411–1423 (2006)
8. Liu, N., Wang, H.: Ensemble based extreme learning machine. IEEE Signal Processing Letters 17(8), 754–757 (2010)
9. Rokach, L.: Ensemble-based classifiers. Artificial Intelligence Review 33(1-2), 1–39 (2010)
10. Wang, B., Huang, S., Qiu, J., Liu, Y., Wang, G.: Parallel online sequential extreme learning machine based on mapreduce. In: The International Conference on Extreme Learning Machines (ELM 2013), Beijing, China, October 15-17 (2013)
11. Xin, J., Wang, Z., Chen, C., Ding, L., Wang, G., Zhao, Y.: Elm*: distributed extreme learning machine with mapreduce. In: World Wide Web, pp. 1–16 (2013)
12. Zhai, J.-h., Xu, H.-y., Wang, X.-z.: Dynamic ensemble extreme learning machine based on sample entropy. Soft Comput. 16(9), 1493–1502 (2012)