Extreme Learning Machine Ensemble Classifier for Large-Scale Data

Haocheng Wang[1,2], Qing He[1], Tianfeng Shang[3], Fuzhen Zhuang[1], and Zhongzhi Shi[1]

[1] Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China
[2] University of Chinese Academy of Sciences, Beijing 100049, China
{wanghc,heq,zhuangfz,shizz}@ics.ict.ac.cn
[3] School of Information Systems, Singapore Management University, Singapore
tfshang@smu.edu.sg

Abstract. For classification problem, extreme learning machine (ELM) can get better generalization performance at a much faster learning speed. Nevertheless, a single ELM is unstable in data classification. The Bagging-based ensemble classifier, i.e., Bagging-ELM has been studied popularly and proved to improve the performance of ELM significantly in terms of accuracy, however, it is inappropriate to deal with large-scale datasets due to the highly intensive computation. In this study, we propose a novel ELM ensemble classifier, namely b-ELM, which leverages the Bag of Little Bootstraps technique to obtain a scalable, efficient means of classification for large-scale data. Efficiency of classification is achieved as it only requires repeated training under consideration on quantities of data that can be much smaller than the original training data. Furthermore, b-ELM is suited to implementation on modern parallel and distributed computing platforms. The experimental results demonstrate that b-ELM can efficiently handle large-scale data with a good performance on prediction accuracy.

Keywords: extreme learning machine, bag of little bootstraps, large-scale data, classification.

1 Introduction

Extreme learning machine (ELM) was proposed as a powerful machine learning technique for single-hidden-layer feedforward neural networks (SLFNs) [8,9,6], and has been studied popularly for its fast learning speed and good generalization performance [3,5,11,4,7]. The essence of ELM is that the hidden layer of SLFNs need not be tuned. Concretely, the hidden neuron parameters are randomly generated which may be independent of the training data, and the output weights are analytically resolved by using Moore–Penrose generalized inverse. Therefore, ELM can overcome the difficulties that the traditional classic gradient-based learning algorithms have to face, such as local minima, learning rate, stopping criteria, and overfitting, etc. Additionally, a wide range of activation functions

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_14

including all piecewise continuous functions can be used as activation functions in ELM.

However, a single ELM is unstable in data classification. To improve the stability and boost the accuracy, more and more researchers consider using ensemble of ELMs. One popular ensemble learning method is Bagging [1], which takes different bootstrap resamples from the original training set and trains a classifier or predictor on each resample to build its constituent members. Several studies have demonstrated that Bagging-ELM is generally more accurate than the individual members [12,14,13].

Bagging-ELM, despite its favorable accuracy of predictability, has high or even prohibitive computational costs. Therefore, its usefulness is severely blunted by the large-scale datasets increasingly encountered in practice. In Bagging-ELM, training in question is repeatedly applied to the bootstrap resamples of the original training set. Because these resamples have size on the order of that of the original training data, with approximately 63.2% of data points appearing at least once in each resample [2], classification on large-scale data can be prohibitively costly. To reduce the computational complexity, one might spontaneously attempt to employ the modern trend toward parallel and distributed computing, i.e., different processors or compute nodes are used to process different bootstrap resamples independently in parallel. However, the large size of bootstrap resamples in the large-scale data setting renders this approach problematic.

For the sake of alleviating the aforementioned problem, we present b-ELM, a novel ELM ensemble classifier which utilizes the Bag of Little Bootstraps (BLB) [10] technique to obtain a scalable, efficient means of classification for large-scale data. The b-ELM algorithm constructs an ensemble of the predictors of bootstrapping multiple small subsets of a larger original training set, and then makes decisions for testing samples through majority voting with the ensemble. It is worth noting that b-ELM has a more favorable computational profile than Bagging-ELM, as it only requires repeated training under consideration on quantities of data that can be much smaller than the original training dataset. Moreover, b-ELM is suited to implementation on modern parallel and distributed computing architectures which are often used to process large datasets. As we show empirically, our procedure possesses superior ability to scale computationally to large-scale datasets, typically incurring less total computation to reach comparably high accuracy.

The remainder of this paper is arranged as follows. In Section 2, preliminary knowledge is described. Subsequently, we introduce b-ELM in full detail in Section 3. Our experimental results to demonstrate the efficiency and effectiveness of b-ELM are given in Section 4. Finally, we conclude in Section 5.

2 Preliminaries

This section will briefly introduce the techniques related to b-ELM.

2.1 Extreme Learning Machine

ELM was originally developed for the single-hidden-layer feedforward neural networks (SLFNs) [8,9,6] and then extended to the "generalized" SLFNs where the hidden layer need not be neuron alike. ELM typically applies random computational nodes in the hidden layer, which may be independent of the training data. It increases learning speed by means of randomly generating weights and biases for hidden nodes rather than iteratively adjusting network parameters which is commonly adopted by gradient-based methods. Different from traditional learning algorithms, ELM tends to reach not only the smallest training error but also the smallest norm of output weights.

The output function of ELM with L hidden nodes for generalized SLFNs is

$$f_L(\boldsymbol{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i g_i(\boldsymbol{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\boldsymbol{a}_i, b_i, \boldsymbol{x}),\ \boldsymbol{x} \in \mathbb{R}^d,\ \boldsymbol{\beta}_i \in \mathbb{R}^m \tag{1}$$

where $\boldsymbol{a}_i$ is the weight vector connecting the input nodes to the ith hidden node, b_i is the bias of the ith hidden node, g_i denotes the output function i.e. activation function $G(\boldsymbol{a}_i, b_i, \boldsymbol{x})$ of the ith hidden node, and $\boldsymbol{\beta}_i$ is the weight vector linking the ith hidden node to the output nodes. For N arbitrary distinct samples $(\boldsymbol{x}_j, \boldsymbol{t}_j) \in \mathbb{R}^d \times \mathbb{R}^m$, SLFNs with L hidden nodes can approximate these N samples with zero error means that there exist $(\boldsymbol{a}_i, b_i)$ and $\boldsymbol{\beta}_i$ such that

$$\sum_{i=1}^{L} \boldsymbol{\beta}_i G(\boldsymbol{a}_i, b_i, \boldsymbol{x}_j) = \boldsymbol{t}_j,\ j = 1, \cdots, N. \tag{2}$$

The above N equations can be written compactly as

$$\boldsymbol{H}\boldsymbol{\beta} = \boldsymbol{T} \tag{3}$$

where

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{h}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{h}(\boldsymbol{x}_N) \end{bmatrix} = \begin{bmatrix} G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_1) \cdots G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_1) \\ \vdots \quad \ddots \quad \vdots \\ G(\boldsymbol{a}_1, b_1, \boldsymbol{x}_N) \cdots G(\boldsymbol{a}_L, b_L, \boldsymbol{x}_N) \end{bmatrix}_{N \times L}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^\top \\ \vdots \\ \boldsymbol{\beta}_L^\top \end{bmatrix}_{L \times m}, \ \boldsymbol{T} = \begin{bmatrix} \boldsymbol{t}_1^\top \\ \vdots \\ \boldsymbol{t}_N^\top \end{bmatrix}_{N \times m}$$

$\boldsymbol{H}$ is the hidden layer output matrix of the SLFN, and the ith column of $\boldsymbol{H}$ is the ith hidden node output with respect to inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N$. While the jth row of $\boldsymbol{H}$, i.e., $\boldsymbol{h}(\boldsymbol{x}_j)$ is the hidden layer feature mapping with respect to the jth input $\boldsymbol{x}_j$. As the hidden node parameters $(\boldsymbol{a}_i, b_i)$ can be randomly generated and remain fixed, the only unknown parameters in ELM are the output weights vectors $\boldsymbol{\beta}_i$ between the hidden layer and the output layer, which can simply

be resolved by ordinary least-square directly. Since ELM aims to minimize the training error $\| \boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{T} \|$ and the norm of weights $\| \boldsymbol{\beta} \|$, the smallest norm least-squares solution of the above linear system is

$$\widehat{\boldsymbol{\beta}} = \boldsymbol{H}^{\dagger}\boldsymbol{T} \tag{4}$$

where $\boldsymbol{H}^{\dagger}$ is the Moore–Penrose generalized inverse of matrix $\boldsymbol{H}$ [6]. Hence, the prediction value matrix $\boldsymbol{Y}$ can be expressed by

$$\boldsymbol{Y} = \boldsymbol{H}\widehat{\boldsymbol{\beta}} = \boldsymbol{H}\boldsymbol{H}^{\dagger}\boldsymbol{T} \tag{5}$$

The error matrix can be described as

$$e = \| \boldsymbol{Y} - \boldsymbol{T} \|^2 = \| \boldsymbol{H}\boldsymbol{H}^{\dagger}\boldsymbol{T} - \boldsymbol{T} \|^2 \tag{6}$$

2.2 Bag of Little Bootstraps

The Bag of Little Bootstraps (BLB), similar to the classical bootstrap, quantifies uncertainty on a statistical estimate, but is better suited for implementation on modern parallel and distributed computing platforms due to its structure. From the input data of size n, it samples without replacement subsamples of size $b = n^{\gamma}$ for typically $0.5 < \gamma \leq 0.9$, which results in a relatively small number of distinct points per subsample compared to the total input size n. From each subsample, resamples of size n are sampled with replacement, and the statistical estimator function is computed on each resample. The differences between the estimates made on the resamples for each subsample are quantified, typically with a standard deviation or variance calculation. The algorithm's output is the average of the error measurements on each of the subsamples.

Obviously, BLB only requires repeated computation on small subsets of the original dataset and avoids the bootstrap's problematic need for repeated computation of the estimate on resamples having size comparable to that of the original dataset. A standard and straightforward calculation reveals that each bootstrap resample contains approximately $0.632n$ distinct data points, which is large if n is large [2]. On the contrary, as previously mentioned, each BLB resample contains at most b distinct data points, and b can be chosen to be much smaller than n or $0.632n$. As a result, the cost of computing the estimate on each BLB resample is commonly substantially lower than the cost of computing the estimate on each bootstrap resample, or on the full dataset. Moreover, BLB typically requires less total computation (across multiple data subsets and resamples) than the bootstrap to reach comparably high accuracy.

3 b-ELM Classifier

In this section we introduce our ELM ensemble classifier, i.e., b-ELM. Our goal is to obtain a scalable, efficient means of classification for large-scale data. b-ELM employs the Bag of Little Bootstraps technique owing to its significantly

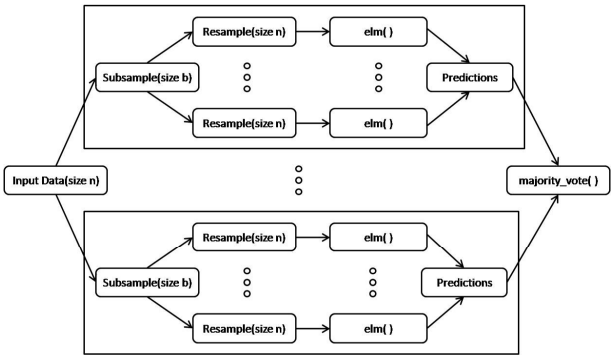

Fig. 1. The workflow of b-ELM. Subsamples are subsampled without replacement, while resamples are drawn with replacement.

computational gains and effective scalability. BLB can capture the diversities of base classifiers from relatively small subsets of data. More concretely, b-ELM is a method of generating training sets for ELM base classifiers by bootstrapping multiple small subsets of a larger original training dataset, subsequently constructing an ensemble of the predictors trained on those training sets, and then making decisions for testing samples through majority voting with the ensemble, i.e., the class that obtains the highest votes is considered as the predicted label. The workflow of b-ELM can be seen in Fig. 1.

The pseudo-code of b-ELM is described in Algorithm 1. There are two pairs of nested loops in the algorithm: the first pair of nested loops is to find the best parameters (e.g., the hidden node number L) for those base classifiers based on a k-fold cross-validation (CV); the other pair is to train the base classifiers and get their predictions for testing dataset. The aggregation phase includes a majority vote to obtain the final decision. The parameter $b = n^{\gamma}$ in the algorithm is the size of subsamples sampled without replacement from the entire original training dataset. According to prior knowledge, we might take $b = n^{\gamma}$ for typically $0.5 < \gamma \leq 0.9$. Obviously, b-ELM has a more favorable space profile than Bagging-ELM. The parameter s indicates the number of subsamples, while r represents the number of resamples bootstrapped from each subsample. In addition, both s and r determine the total number of predictions for the testing dataset.

Owing to its much smaller subsample and resample sizes, b-ELM is notably more amenable than Bagging-ELM to distribution of different subsamples and resamples and their associated computations to independent compute nodes; thus, b-ELM allows for simple parallel and distributed implementations, enabling additional large computational gains.

In the large-scale data setting, training on the entire training dataset often requires simultaneous distributed training across multiple compute nodes, among which the observed dataset is partitioned. Given the large size of each bootstrap resample, training on even a single such resample in turn also requires the use of a comparably large cluster of compute nodes; Bagging-ELM requires repetition

Algorithm 1. b-ELM

Input:
$\boldsymbol{T}$: training dataset $\{x_1, x_2, \ldots, x_n\}$
$\boldsymbol{y_T}$: labels of training dataset
$\boldsymbol{S}$: testing dataset
$\boldsymbol{y_S}$: labels of testing dataset
$\boldsymbol{P}$: parameters of ELM
b: subset size
s: number of sampled subsets
r: number of Monte Carlo iterations
Output: the aggregated prediction results $\widehat{\boldsymbol{y}}_{\boldsymbol{S}}$

for each $\boldsymbol{P}$ **do**
  for each fold in k-fold CV **do**
    $\boldsymbol{M} = elm\_train(\boldsymbol{T}, \boldsymbol{y_T}, \boldsymbol{P})$
  end for
end for
Save the best-performed parameters $\boldsymbol{P}_{best}$
for $i = 1$ to s **do**
  //Subsample the data
  Randomly sample a subset $\boldsymbol{B}_i$ of b indices from $\{1, 2, \ldots, n\}$ without replacement
  for $j = 1$ to r **do**
    Draw a random pseudo-sample $\boldsymbol{D}_j$ of size n from $\boldsymbol{B}_i$ with replacement
    //Training
    $\boldsymbol{M}^{(i-1)\times r+j} = elm\_train(\boldsymbol{D}_j, \boldsymbol{y}_{\boldsymbol{D}_j}, \boldsymbol{P}_{best})$
    //Testing
    $\widehat{\boldsymbol{y}}_{\boldsymbol{S}}^{(i-1)\times r+j} = elm\_test(\boldsymbol{M}^{(i-1)\times r+j}, \boldsymbol{S}, \boldsymbol{y_S})$
    Store prediction $\widehat{\boldsymbol{y}}_{\boldsymbol{S}}^{(i-1)\times r+j}$
  end for
end for
//Aggregation
Majority Vote $\widehat{\boldsymbol{y}}_{\boldsymbol{S}}^{k} \Rightarrow \widehat{\boldsymbol{y}}_{\boldsymbol{S}},\ k = 1, 2, \ldots, s \times r$
return $\widehat{\boldsymbol{y}}_{\boldsymbol{S}}$

of this training process for multiple resamples. Each training process is thus quite costly, and the aggregate computational costs of this repeated distributed training process are quite high. Indeed, the training for each bootstrap resample requires use of an entire cluster of compute nodes and incurs the associated overhead.

Conversely, b-ELM straightforwardly allows to train on multiple (or even all) subsamples and resamples simultaneously in parallel: because subsamples and resamples of b-ELM can be significantly smaller than the original training dataset, they can be transferred to, stored by, and processed on individual (or very small sets of) compute nodes. For instance, we could naturally utilize modern hierarchical distributed architectures by distributing subsamples to different compute nodes and subsequently using intra-node parallelism to train across different

resamples generated from the same subsample. Hence, compared with Bagging-ELM, b-ELM both decreases the total computational cost of classification and allows more natural use of parallel and distributed computational resources.

4 Experiments

In this section, the performance of the proposed b-ELM is compared with the single ELM and Bagging-ELM. Simulations are conducted on simulated data and real data. All experiments are implemented and executed using MATLAB on a single processor. More specifically, the current version of basic ELM is employed here, and the sigmoid function $g(x) = 1/(1 + e^{-\lambda x})$ is selected as the activation function. Motivated by the need for good performance, 5-fold cross-validation is performed to select meta-parameters of basic ELM on each training set, i.e., one fold is used as testing set for a classifier built on the remaining four in each cross. The grid search is implemented once for each dataset in advance to avoid overfitting, and the parameter tuning time is not considered here. For each dataset, fifty trials are conducted for all the algorithms and the average results are reported in this paper.

4.1 Datasets

The experiments are performed on several datasets, including a simulated dataset ("3-Covers"), and nine other classical datasets from the UCI Machine Learning repository which has been extensively used in testing the performance of different kinds of classifiers. The basic information of the datasets is given in Table 1. The training and testing data of these 10 datasets are fixed for all trials of simulations. Moreover, all the attributes have been normalized into the range [0, 1].

Table 1. Specifications of classification datasets

Datasets	# Classes	# Attributes	# Training data	# Testing data
Balance	3	4	400	225
Car	4	6	1,200	528
Waveform	3	21	3,000	2,000
Mushroom	2	22	4,000	4,124
Digits	10	16	7,494	3,498
Letter	26	16	16,000	4,000
3-Covers	3	7	10,000	7,895
Adult	2	14	10,000	38,842
Covertype	7	54	15,120	565,892
Poker	10	10	25,010	1,000,000

4.2 Evaluation Metrics

Considering that the *accuracy* measure may be vulnerable to the class unbalance, we employ both the standard F_1 and *accuracy* metrics to evaluate the prediction

performance of different classifiers on each dataset. *Accuracy* is the proportion of true results in the population. For each known class we calculate F_1^i as follows,

$$F_1^i = \frac{2 \times precison_i \times recall_i}{precison_i + recall_i}, \quad i \in \{1, \cdots, k\}, \tag{7}$$

where $precison_i$ and $recall_i$ are the precision and recall on the i-th known class. Then,

$$F_1 = \sum_{i=1}^{k} F_1^i / k, \tag{8}$$

where k is the number of known classes.

4.3 Simulated Data

We first utilize the simulated 3-Covers dataset to evaluate the performance characteristics of b-ELM. More concretely, we study the prediction and computational properties of b-ELM as well as Bagging-ELM. To maintain consistency of notation, we henceforth refer to the basic ELM as Single-ELM. Moreover, we consider $b = n^\gamma$ where $\gamma \in \{0.6, 0.7, 0.8, 0.9\}$ in runs of b-ELM. For both b-ELM and Bagging-ELM, identical evaluation criterions have been used to evaluate the classification quality in the experiments.

From Fig. 2, we can see that b-ELM succeeds in converging to high *accuracy* value significantly more quickly than Bagging-ELM for all values of b considered. When computing on a single processor, b-ELM generally requires less time, and hence less total computation, than Bagging-ELM to attain comparably high classification accuracy. Those results only hint at b-ELM's superior ability to scale computationally to large-scale data. As seen in Fig. 3, b-ELM also outperforms Bagging-ELM in terms of achieving higher F_1 value with less time on 3-Covers dataset.

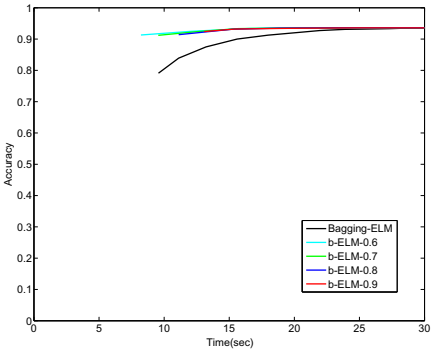

Fig. 2. *Accuracy* vs. *processingtime* for both b-ELM and Bagging-ELM classification on 3-Covers dataset. For b-ELM, $b = n^\gamma$ with the value of γ for each trajectory are given in the legend.

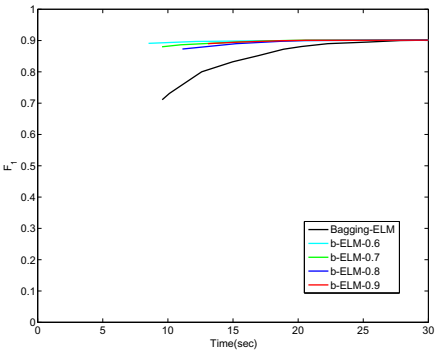

Fig. 3. F_1 vs. *processingtime* for both b-ELM and Bagging-ELM classification on 3-Covers dataset. For b-ELM, $b = n^\gamma$ with the value of γ for each trajectory are given in the legend.

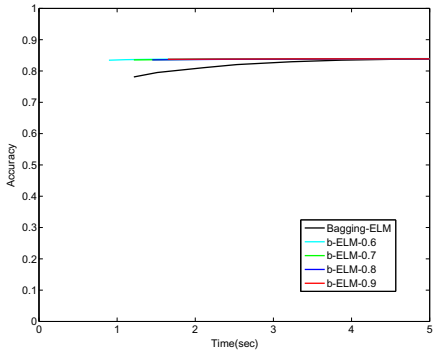

Fig. 4. *Accuracy* vs. *processingtime* for both b-ELM and Bagging-ELM classification on Poker dataset. For b-ELM, $b = n^{\gamma}$ with the value of γ for each trajectory are given in the legend.

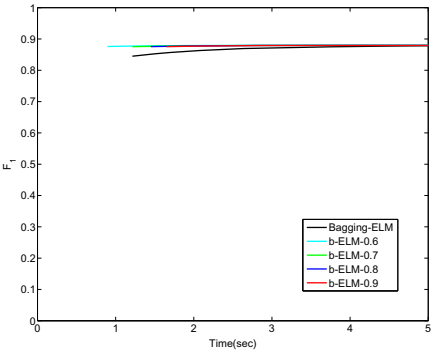

Fig. 5. F_1 vs. *processingtime* for both b-ELM and Bagging-ELM classification on Poker dataset. For b-ELM, $b = n^{\gamma}$ with the value of γ for each trajectory are given in the legend.

4.4 Real Data

We now present the results of applying b-ELM along with Bagging-ELM to several different real datasets from the UCI Machine Learning repository. As expected, the performances of b-ELM, remain substantially better than those of Bagging-ELM. Figs. 4 and 5 show *accuracy* or F_1 vs. *processingtime* for both b-ELM and Bagging-ELM classifications on Poker dataset, respectively. For b-ELM, $b = n^{\gamma}$ with the value of $\gamma \in \{0.6, 0.7, 0.8, 0.9\}$ for each trajectory are given in the legend. Notably, the outputs of b-ELM for all values of b considered, and the output of Bagging-ELM, are tightly clustered around the same value; additionally, as expected, b-ELM converges more quickly than Bagging-ELM. Furthermore, We have obtained qualitatively similar results on other additional datasets from the UCI dataset repository.

Table 2. Prediction performance comparisons among b-ELM, Bagging-ELM, and Single-ELM

Datasets	*Accuracy* (%)			F_1 (%)		
	Single-ELM	Bagging-ELM	b-ELM	Single-ELM	Bagging-ELM	b-ELM
Balance	90.24	91.26	91.27	88.93	89.76	89.83
Car	91.45	94.23	94.61	85.31	88.14	88.17
Waveform	84.79	86.73	86.75	82.27	85.13	85.17
Mushroom	88.82	89.31	89.38	87.93	88.78	88.81
Digits	95.31	97.18	97.21	94.14	96.87	96.86
Letter	93.54	97.18	97.24	92.83	95.75	95.79
3-Covers	87.94	93.54	93.61	86.67	89.91	89.97
Adult	84.12	85.23	85.24	79.57	80.87	80.91
Covertype	79.23	85.37	85.41	69.92	78.53	78.55
Poker	50.77	83.87	84.23	52.31	86.82	87.93

Table 2 gives a summary of prediction properties of b-ELM compared to Bagging-ELM and Single-ELM. Here, we use $b = n^{0.7}$ in all runs of b-ELM. For each dataset, both b-ELM and Bagging-ELM are repeated for 50 times as well as reaching relatively high *accuracy* or F_1 value under appropriate settings. Then, for each dataset, the average *accuracy* or F_1 are recorded. The average performance of Single-ELM is also recorded for comparison with the same parameter settings. Obviously, the classification performances of both b-ELM and Bagging-ELM are much better than that of an individual ELM. Also, b-ELM is more scalable than Bagging-ELM while maintaining favorable classification performance.

5 Conclusions

In this paper, the b-ELM algorithm is proposed, which employs the Bag of Little Bootstraps technique to attain a scalable, efficient means of classification for large-scale data. b-ELM typically has markedly better space and computational profiles than Bagging-ELM. Experimental results confirm the efficiency and effectiveness of the proposed algorithm. Moreover, b-ELM is suited to implementation on modern parallel and distributed computing platforms. We will deploy and demonstrate our procedure over a cluster of compute nodes in the future.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (No. 61035003, 61175052, 61203297), National High-tech R&D Program of China (863 Program) (No. 2012AA011003, 2013AA01A606, 2014AA015105).

References

1. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
2. Efron, B., Tibshirani, R.J.: An introduction to the bootstrap, vol. 57. Chapman and Hall (1993)
3. Huang, G.-B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing 70(16), 3056–3062 (2007)
4. Huang, G.-B., Ding, X., Zhou, H.: Optimization method based extreme learning machine for classification. Neurocomputing 74(1), 155–163 (2010)
5. Huang, G.-B., Li, M.-B., Chen, L., Siew, C.-K.: Incremental extreme learning machine with fully complex hidden nodes. Neurocomputing 71(4), 576–583 (2008)
6. Huang, G.-B., Wang, D.H., Lan, Y.: Extreme learning machines: a survey. International Journal of Machine Learning and Cybernetics 2(2), 107–122 (2011)
7. 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)
8. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2004), vol. 2, pp. 985–990. IEEE (2004)

9. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: theory and applications. Neurocomputing 70(1), 489–501 (2006)
10. Kleiner, A., Talwalkar, A., Sarkar, P., Jordan, M.I.: The big data bootstrap. In: Proceedings of the 29th International Conference on Machine Learning (ICML-2012), pp. 1759–1766 (2012)
11. Lan, Y., Soh, Y.C., Huang, G.-B.: Ensemble of online sequential extreme learning machine. Neurocomputing 72(13), 3391–3395 (2009)
12. Tian, H., Meng, B.: A new modeling method based on bagging elm for day-ahead electricity price prediction. In: IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), pp. 1076–1079. IEEE (2010)
13. Xu, R.-z., Geng, X.-f., Zhou, F.-y.: A short term load forecasting based on bagging-elm algorithm. In: Lu, W., Cai, G., Liu, W., Xing, W. (eds.) Proceedings of the 2012 International Conference on Information Technology and Software Engineering. LNEE, vol. 211, pp. 507–514. Springer, Heidelberg (2013)
14. Ye, R., Suganthan, P.N.: Empirical comparison of bagging-based ensemble classifiers. In: 15th International Conference on Information Fusion, pp. 917–924. IEEE (2012)