Distributed Learning over Massive XML Documents in ELM Feature Space

Xin Bi, Xiangguo Zhao*, Guoren Wang, Zhen Zhang, and Shuang Chen

College of Information Science and Engineering,
Northeastern University, Liaoning, Shenyang, China 110819
edijasonbi@gmail.com,
zhaoxiangguo@mail.neu.edu.cn

Abstract. Since centralized learning solutions are unable to meet the requirements of mining applications with massive training samples, a solution to distributed learning over massive XML documents is proposed in this paper, which provides distributed conversion of XML documents into representation model in parallel based on MapReduce, and a distributed learning component based on Extreme Learning Machine for mining tasks of classification or clustering. Within this framework, training samples are converted from raw XML datasets with better efficiency and information representation ability and taken to distributed learning algorithms in ELM feature space. Extensive experiments are conducted on massive XML documents datasets to verify the effectiveness and efficiency for both distributed classification and clustering applications.

Keywords: XML, Extreme Learning Machine, classification, clustering, distributed computing.

1 Introduction

Extreme Learning Machine (ELM) was proposed by Huang, *et al.* in [11,12] based on generalized single-hidden layer feedforward networks (SLFNs). With its variants[10,2,13,7,6], ELM achieves extremely fast learning capacity and good generalization performance due to its *universal approximation capability* and *classification capability*. Recently, paper [8] pointed out that from the optimization method point of view, ELM for classification and SVM are equivalent. Furthermore, it is proved in [9] that ELM provides a unified learning platform with a widespread type of feature mappings.

It is generally believed that all the ELM based algorithms consist of two major stages[5]: 1) random feature mapping; 2) output weights calculation. The first stage is the key concept in ELM theory. Most existing ELM based classification algorithms can be viewed as *supervised learning in ELM feature space*. While in [3], the *unsupervised learning in ELM feature space* is studied, drawing the conclusion that the proposed ELM kMeans algorithm can get better clustering results than in original feature space.

* Corresponding author.

J. Cao et al. (eds.), *Proceedings of ELM-2014 Volume 2,*
Proceedings in Adaptation, Learning and Optimization 4, DOI: 10.1007/978-3-319-14066-7_27

Recently, the volume of XML documents keeps explosively increasing. MapReduce[1] provides tremendous parallel computing power without concerns for the underlying implementation and technology. However, MapReduce framework requires distributed storage of the datasets and no communication among mappers or reducers, which brings challenges to: 1) converting XML datasets into representation model; 2) implementing learning algorithms in ELM feature space.

To our best knowledge, massive XML mining problems in this paper is discussed for the *first* time. The contributions can be summarized as

1. A *distributed representing algorithm* DXRC is proposed to convert massive XML documents into XML representation model in parallel;
2. Existing *distributed supervised learning algorithms in ELM feature space* are implemented to make comparison of massive XML documents classification performance, including PELM and POS-ELM;
3. A *distributed unsupervised learning algorithm* DEK is proposed based on ELM kMeans[3] to realize distributed clustering over massive XML documents in *ELM feature space*;
4. Empirical experiments are conducted on clusters to verify the performance of our solution.

The remainder of this paper is structured as follows. Section 2 proposes a distributed representation converting algorithm. ELM feature mapping is presented in Section 3. Section 4 presents classification algorithms based on distributed ELMs and proposes a distributed clustering algorithm in ELM feature space based on MapReduce. Section 6 shows the experimental results. Section 7 draws conclusions of this paper.

2 Distributed XML Representation

In this section, we propose a distributed converting algorithm, named Distributed XML Representation Converting (DXRC), to calculate TFIDF[14] for DSVM based on MapReduce.

The *map* function in Algorithm 1 accepts key-value pairs as input. The key of key-value pairs is the XML document ID and the value is the corresponding XML document content. A HashMap *mapEle* (Line 1) is used to cache the all the elements of one XML document (Lines 2-11), using element name as key and another HashMap $mapEleTF$ (Line 4) as value. The $mapEleTF$ caches the TF values of all the words in one element (Lines 5-10). That is, for each XML document, there are as many items in *mapEle* as there are elements; for each element, there are as many items in $mapEleTF$ as there are distinct words in this element. Each item in *mapEle* and *mapEle* will be emitted as output in the form of $\langle term, \langle docID, element, times, sum\rangle\rangle$ (Lines 12-17).

After the $\langle term, \langle docID, element, times, sum\rangle\rangle$ pairs are emitted by map function, all the key-value pairs with the same key, which are also the key-value pairs of the same word in XML documents, are combined and passed to the same *reduce* function in Algorithm 2 as input. For each key-value pair processed by

Algorithm 1. Mapper of DXRC

```
   Input: ⟨docID, content⟩
   Output: ⟨term, ⟨docID, element, times, sum⟩⟩
 1 Initiate HashMap mapEle;
 2 foreach element ∈ content do
 3     Initiate sum = 0;
 4     Initiate HashMap mapEleTF;
 5     foreach term ∈ element do
 6         sum++;
 7         if mapEleTF.containsKey(term) then
 8             mapEleTF.put(term, mapEleTF.get(term)+1);
 9         else
10             mapEleTF.put(term,1);
11     mapEle.put(element, mapEleTF);
12 foreach itrEle ∈ mapEle do
13     element = ele.getKey();
14     foreach itrEleTF ∈ itrEle do
15         term = itrEleTF.getKey();
16         times = itrEleTF.getValue();
17         emit(term, ⟨docID, element, times, sum⟩);
```

reduce function, two HashMaps $mapDocEleTF$ (Line 1) and $mapTDocs$ (Line 2) are initiated. The HashMap $mapDocEleTF$ is to cache the tf values of a word in each element in the corresponding XML document and $mapTDocs$ is to cache the number of documents containing this word. The total number of document N (Line 3) and the vector $weights$ (Line 4), which indicates the wights of all the elements in each XML document, are obtained through distributed cache defined in MapReduce job configuration. Since Reduce now have all the tf values grouped by XML elements along with their weights, weighted tf values (Line 6) and the number of documents containing each word are calculated and cached in $mapDocEleTF$ and $mapTDocs$ respectively (Lines 5-12). Then the idf value can be calculated (Line 14) and multiplied by each item in $mapDocEleTF$. The output of reduce is the $\langle position, tfidf \rangle$ pairs, of which $position$ is $\langle docID, element \rangle$ indicating the index of DSVM matrix and $tfidf$ is the value of the matrix.

3 ELM Feature Mapping

In ELM, the n input nodes correspond to the n-dimensional data space of original training samples; while L hidden nodes correspond to the L-dimensional *ELM feature space*. With the m-dimensional output space, the decision function output the class label of the training sample.

Algorithm 2. Reducer of DXRC

Input: $\langle term, list(\langle docID, element, times, sum\rangle)\rangle$
Output: training samples matrix in the form of $\langle position, tfidf\rangle$

```
Initiate HashMap mapDocEleTF;
Initiate HashMap mapTDocs;
N=DistributedCache.get("totalDocsNum");
weights = DistributedCache.get("elementWeightsVector");
foreach itr ∈ list do
    weightedDocEleTF = weights[docId, element] * itr.times/itr.sum;
    mapDocEleTF.put(⟨docId, element⟩, weightedDocEleTF);
    if mapTDocs.containsKey(docId) then
        newTimes = mapTF.get(docId) + irt.getValue().times;
        mapTDocs.put(docID, newTimes);
    else
        mapTDocs.put(docID,itr.getValue().times);
docsNumber = mapTDocs.size();
idf = log(mapTDocsSize/N);
foreach itrDocEleTF ∈ mapDocEleTF do
    position = itrDocEleTF.getKey();
    tfidf = itrDocEleTF.getValue()*idf;
    emit(position, tfidf);
```

The *ELM feature mapping* denoted as $\mathbf{H}$ is calculated as

$$\mathbf{H} = \begin{bmatrix} \mathrm{h}(\mathbf{x}_1) \\ \vdots \\ \mathrm{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} \mathrm{G}(\mathbf{w}_1, b_1, \mathbf{x}_1) & \cdots & \mathrm{G}(\mathbf{w}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ \mathrm{G}(\mathbf{w}_1, b_1, \mathbf{x}_N) & \cdots & \mathrm{G}(\mathbf{w}_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N \times L} \tag{1}$$

4 Distributed Classification in ELM Feature Space

In this section, we introduce the learning procedure of classification problems in ELM feature space, and distributed implementations based on two existing representative distributed ELM algorithms, which are PELM[4] and POS-ELM[15].

4.1 Supervised Learning in ELM Feature Space

In supervised learning applications, since ELM is to minimize the training error and the norm of the output weights[11,12], that is

$$\text{Minimize:} \quad \|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|^2 \quad \text{and} \quad \|\boldsymbol{\beta}\| \tag{2}$$

where $\mathbf{T} = \left[\mathbf{t}_1^{\mathrm{T}}, ..., \mathbf{t}_L^{\mathrm{T}}\right]_{m \times L}^{\mathrm{T}}$ is the vector of class labels.

The matrix $\boldsymbol{\beta}$ is the output weight, which is calculated as

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \tag{3}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose Inverse of $\mathbf{H}$.

If the number of training samples is much larger than the dimensionality of the feature space, the output weight calculation equation can be rewritten as

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^{\mathrm{T}}\mathbf{H}\right)^{-1} \mathbf{H}^{\mathrm{T}}\mathbf{T} \tag{4}$$

4.2 Distributed Implementations

The original ELM was parallelized by PELM in [4]; Online sequential ELM (OS-ELM) was implemented on MapReduce as POS-ELM in [15].

Parallel ELM. Since the major cost in ELM is the calculation of generalized inverse of matrix $\mathbf{H}$, The matrix multiplication $\mathbf{U} = \mathbf{H}^{\mathrm{T}}\mathbf{H}$ and $\mathbf{V} = \mathbf{H}^{\mathrm{T}}\mathbf{T}$ can be calculated by a MapReduce job. In map function, each term of $\mathbf{U}$ and $\mathbf{V}$ is calculated in parallel. In reduce function, all the intermediate results are merged and summed to the corresponding elements of the result matrix.

Parallel Online Sequential ELM. The basic idea of Parallel Online Sequential ELM (POS-ELM) is to calculate $\mathbf{H}_1, \cdots, \mathbf{H}_B$ in parallel. POS-ELM takes advantage of the calculation of partial ELM feature matrix $\mathbf{H}_i$ with a chunk of training data of OS-ELM, in each map function calculates its $\mathbf{H}_i$ with its own data chunk. The reduce function collects all the $\mathbf{H}_i$ and calculates $\boldsymbol{\beta}_{k+1}$ as

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \mathbf{P}_{k+1}\mathbf{H}_{k+1}^{\mathrm{T}}(\mathbf{T}_{k+1} - \mathbf{H}_{k+1}\boldsymbol{\beta}_k) \tag{5}$$

where

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k\mathbf{H}_{k+1}^{\mathrm{T}}(\mathbf{I} + \mathbf{H}_{k+1}\mathbf{P}_k\mathbf{H}_{k+1}^{\mathrm{T}})^{-1}\mathbf{H}_{k+1}\mathbf{P}_k \tag{6}$$

5 Distributed Clustering in ELM Feature Space

Generally, kMeans algorithm in ELM feature space, as ELM kMeans for short, has two major steps: 1) transform the original data into ELM feature space; 2) implement traditional clustering algorithm directly. Clustering in the ELM feature space is much more convenient than kernel based algorithms.

We present Distributed ELM k-Means (DEK) based on ELM kMeans[3]. Algorithm 3 presents the map function of DEK. For each sample stored on this mapper (Line 1), the distance between the sample and each cluster centroids is calculated (Lines 2, 3). Then each sample is assigned to the cluster whose centroid is the nearest one to this sample (Line 4). The intermediate key-value pair is emitted in the form of $\langle c_{max}, \mathbf{x}_i \rangle$ (Line 5), in which $\mathbf{x}_i$ is the specific sample and c_{max} is the assigned cluster of $\mathbf{x}_i$.

Algorithm 4 presents the reduce function of DEK. We add up the sum distance in Euclidean space of all the samples $\mathbf{x}_i$ in list($\mathbf{x}$) (Lines 1, 2), and then calculate the mean value to represent the new version of the centroid $c_j^{updated}$ of cluster c_j

Algorithm 3. Mapper of DEK

Input: Training samples $\mathbf{X}$, k centroids $\mathbf{C}$
Output: $\langle$centroid c_{max}, sample $\mathbf{x}_i\rangle$

```
1 foreach x_i ∈ X do
2     foreach c_j ∈ C do
3         Calculate distance d_ij between x_i and c_j;
4     Assign x_i to the cluster c_max with max_j(d_ij);
5     Emit ⟨c_max, x_i⟩;
```

(Line 3). When all the k cluster centroids are updated, if this version of centroids are the same as the older one, or if the maximum iteration number is reached, DEK holds that the clustering job is done; otherwise, DEK continues to the next round of MapReduce job until convergence.

Algorithm 4. Reducer of DEK

Input: $\langle$centroid c_j, samples list($\mathbf{x}$)$\rangle$
Output: Updated set of centroids $\mathbf{C}_{updated}$

```
1 foreach x_i ∈ list(x) do
2     Add x_i to squared sum S;
3 Calculate c_j^updated of cluster c_j as c_j^updated = S/list(x).length;
```

6 Performance Evaluation

6.1 Experiments Setup

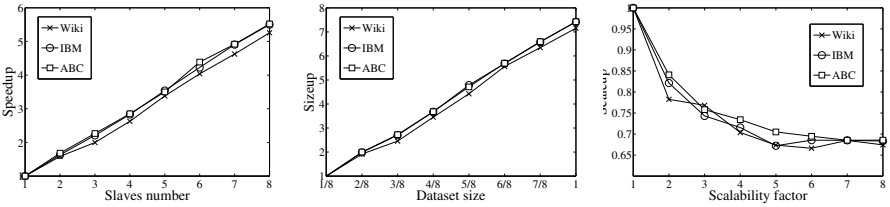
Three datasets *Wikipedia XML Corpus* provided by INEX, *IBM DeveloperWorks* articles and *ABC News* are used in our paper. We choose the same numbers of XML documents out of all the three datasets, that is 6 classes and 500 documents in each class. The only parameter of learning algorithms in ELM feature space, i.e., the number of hidden nodes L, is set to 800.

To evaluate the performance, three sets of evaluation criteria are utilized: 1) for scalability evaluation, we compare the criteria of speedup, sizeup and scaleup; 2) for classification problems, accuracy, recall and F-measure are used; 3) for clustering problems, since we treat this class label as the cluster label, the same evaluation criteria are used as classification problems.

All the experiments are conducted on a Hadoop cluster of nine machines, each of which is equipped with an Intel Quad Core 2.66GHZ CPU, 4GB of memory and CentOS 5.6 as operating system. The MapReduce framework is configured with Hadoop version 0.20.2 and Java version 1.6.0_24.

6.2 Evaluation Results

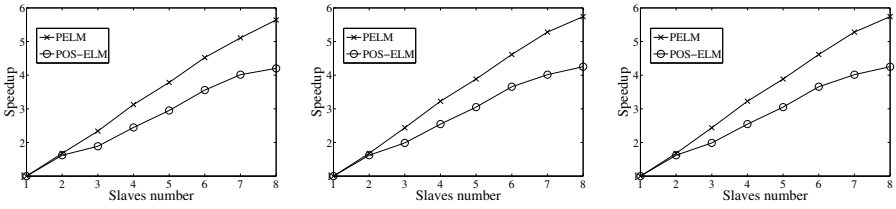
Scalability of DXRC. The scalability of representation converting algorithm DXRC is first evaluated. Figure 1 demonstrates good speedup, sizeup and scaleup of DXRC. The representation ability DSVM applied in DXRC can be found in our previous work [16].



(a) Speedup (b) Sizeup (c) Scaleup

Fig. 1. Scalability of DXRC

Scalability of Massive XML Classification in ELM Feature Space. The speedup comparison between PELM and POS-ELM on three datasets are presented in Figure 2.



(a) Dataset Wiki (b) Dataset IBM (c) Dataset ABCNews

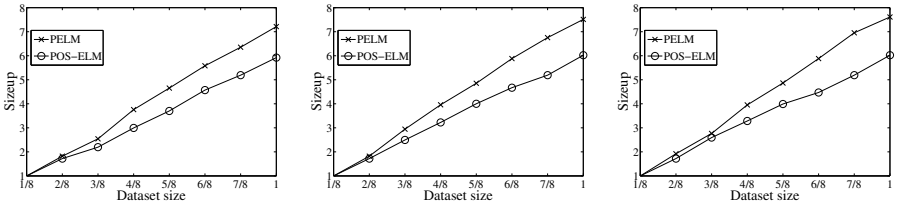
Fig. 2. Comparison of speedup between PELM and POS-ELM

Since the centralized calculation reduces the scalability of both PELM and POS-ELM to some degree, especially for POS-ELM. Thus, the speedup of PELM is better than POS-ELM.

Figure 3 demonstrates the sizeup comparison between PELM and POS-ELM, from which we find that the sizeup of PELM is better than POS-ELM.

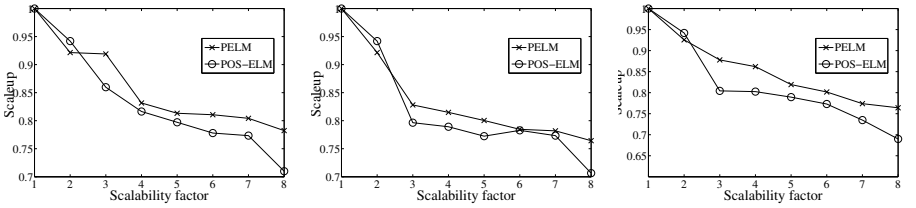
For the scaleup comparison, Figure 4 demonstrates that both PELM and POS-ELM have good scaleup performance, and PELM outperforms POS-ELM on each of the three datasets.

In summary, PELM has better scalability than POS-ELM, but both of them have good scalability for massive XML documents classification applications.



(a) Dataset Wiki (b) Dataset IBM (c) Dataset ABCNews

Fig. 3. Comparison of sizeup between PELM and POS-ELM



(a) Dataset Wiki (b) Dataset IBM (c) Dataset ABCNews

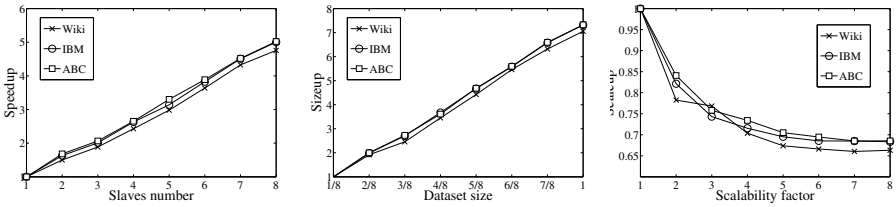
Fig. 4. Comparison of scaleup between PELM and POS-ELM

Performance of Massive XML Classification in ELM Feature Space. The classification results are shown in Table 1, from which we can see that PELM slightly outperforms POS-ELM. However, both PELM and POS-ELM provide satisfactory classification performance.

Table 1. Classification performance comparison between PELM and POS-ELM

Datasets	PELM			POS-ELM		
	Accuracy	Recall	F-measure	Accuracy	Recall	F-measure
Wikipedia	0.7886	0.7563	0.7721	0.7923	0.7745	0.7833
IBM developWorks	0.7705	0.8145	0.7919	0.7711	0.7863	0.7891
ABC News	0.8681	0.8517	0.8598	0.8517	0.8335	0.8425

Scalability of Massive XML Clustering in ELM Feature Space. In theory, the scalability of distributed k-Means in ELM feature space and in original feature space are the same, we only presents the scalability of DEK without comparison with distributed k-Means in original feature space in Figure 5, which demonstrates good scalability of DEK.



(a) Speedup (b) Sizeup (c) Scaleup

Fig. 5. Scalability of DEK

Performance of Massive XML Clustering in ELM Feature Space. The comparison results on three different datasets are presented in Table 2. It can be seen from the comparison results that DEK gets better clustering performance due to its ELM features mapping.

Table 2. Clustering performance of DEK compared with parallel k-Means

Dataset	Parallel k-Means			DEK		
	Accuracy	Recall	F-measure	Accuracy	Recall	F-measure
Wikipedia	0.7602	0.7426	0.7513	0.7737	0.7648	0.7692
IBM developerWorks	0.7985	0.8268	0.8126	0.8124	0.8277	0.8200
ABC News	0.8351	0.8125	0.8201	0.8529	0.8192	0.8357

7 Conclusion

This paper addresses the problem of distributed XML documents learning in ELM feature space, which has no previous work to our best knowledge. Parallel XML documents representation converting problem is discussed by proposing algorithm DXRC. Massive XML documents classification in ELM feature space is studied by implementing PELM and POS-ELM; while for massive XML documents clustering in ELM feature space, a distributed ELM k-Means algorithm DEK is proposed. Experimental results demonstrate that the distributed XML learning in ELM feature space shows good scalability and learning performance.

Acknowledgment. This research is partially supported by the National Natural Science Foundation of China under Grant Nos. 61272181, and 61173030; the National Basic Research Program of China under Grant No. 2011CB302200-G; the 863 Program under Grant No. 2012AA011004; and the Fundamental Research Funds for the Central Universities under Grant No. N120404006.

References

1. Dean, J., Ghemawat, S.: MapReduce: Simplied Data Processing on Large Clusters. In: Operating Systems Design and Implementation, pp. 137–150 (2004)
2. Feng, G., Huang, G.B., Lin, Q., Gay, R.K.L.: Error minimized extreme learning machine with growth of hidden nodes and incremental learning. IEEE Transactions on Neural Networks 20, 1352–1357 (2009)
3. He, Q., Jin, X., Du, C., Zhuang, F., Shi, Z.: Clustering in extreme learning machine feature space. Neurocomputing 128, 88–95 (2014)
4. He, Q., Shang, T., Zhuang, F., Shi, Z.: Parallel extreme learning machine for regression based on mapreduce. Neurocomputing 102, 52–58 (2013)
5. Huang, G., Song, S., Gupta, J., Wu, C.: Semi-supervised and unsupervised extreme learning machines. IEEE Transactions on Cybernetics PP(99), 1–1 (2014)
6. Huang, G.B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing 70, 3056–3062 (2007)
7. Huang, G.B., Chen, L.: Enhanced random search based incremental extreme learning machine. Neurocomputing 71, 3460–3468 (2008)
8. Huang, G.B., Ding, X., Zhou, H.: Optimization method based extreme learning machine for classification. Neurocomputing 74, 155–163 (2010)
9. Huang, G.B., Zhou, H., Ding, X., Zhang, R.: Extreme Learning Machine for Regression and Multiclass Classification. IEEE Transactions on Systems, Man, and Cybernetics 42, 513–529 (2012)
10. Huang, G.B., Zhu, Q.Y., Mao, K.Z., Siew, C.K., Saratchandran, P., Sundararajan, N.: Can threshold networks be trained directly? IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 53, 187–191 (2006)
11. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: International Symposium on Neural Networks, vol. 2 (2004)
12. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: Theory and applications. Neurocomputing 70, 489–501 (2006)
13. Rong, H.J., Huang, G.B., Sundararajan, N., Saratchandran, P.: Online sequential fuzzy extreme learning machine for function approximation and classification problems. IEEE Transactions on Systems, Man, and Cybernetics 39, 1067–1072 (2009)
14. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. Information Processing and Management 24, 513–523 (1988)
15. Wang, B., Huang, S., Qiu, J., Liu, Y., Wang, G.: Parallel Online Sequential Extreme Learning Machine Based on MapReduce. To appear in Neurocomputing (2014)
16. Zhao, X., Bi, X., Qiao, B.: Probability based voting extreme learning machine for multiclass xml documents classification. In: World Wide Web, pp. 1–15 (2013)