Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 923097, 13 pages
http://dx.doi.org/10.1155/2015/923097

Research Article

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, Shenyang, Liaoning 110819, China

Correspondence should be addressed to Xiangguo Zhao; zhaoxiangguo@mail.neu.edu.cn

Received 21 August 2014; Accepted 16 October 2014

Academic Editor: Tao Chen

With the exponentially increasing volume of XML data, centralized learning solutions are unable to meet the requirements of mining applications with massive training samples. In this paper, a solution to distributed learning over massive XML documents is proposed, 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 Extreme Learning Machine (ELM) feature space. Extensive experiments are conducted on massive XML documents datasets to verify the effectiveness and efficiency for both classification and clustering applications.

1. Introduction

Classification and clustering are two major problems of XML documents mining tasks. One of the most important parts of mining XML documents is to convert XML documents into *representation model*. Most traditional representation models are designed for plain text mining applications, taking no account of structural information of XML documents. Vector Space Model (VSM) [1] is one of the most classic and popular representation models of plain text. Previous work proposed some approaches to solve the problem of considering both semantic and structural information for XML document classification, among which Structured Link Vector Model (SLVM) [2] extends VSM to generate a matrix by recording the attribute values of each element in an XML document. Reduced Structured Vector Space Model (RS-VSM) proposed in [3] achieves a higher performance due to its feature subset selection method. Different weights are assigned to different elements according to their priority and representation ability. Distribution based Structured Vector Model (DSVM) [4] further improves the calculation of traditional Term Frequency Inverse Document Frequency (TFIDF) values and takes two factors into consideration, namely, Among Classes Discrimination and Within Class Discrimination.

Extreme Learning Machine (ELM) was proposed by Huang et al. in [5, 6] based on generalized single-hidden layer feedforward networks (SLFNs). With its variants [7–11], ELM achieves extremely fast learning capacity and good generalization capabilities usually in many application fields, including text classification [12], multimedia recognition [13–15], bioinformatics [16], and mobile objects [17]. Recently, Huang et al. in [18] pointed out that (1) the maximal margin property of Support Vector Machine (SVM) [19] and the minimal norm of weights theory of ELM are consistent; (2) from the standard optimization method point of view, ELM for classification and SVM are equivalent. Furthermore, it is proved in [20] that (1) ELM provides a unified learning platform with a widespread type of feature mappings; (2) ELM can be implemented in regression and multiclass classification applications in one formula directly. ELM can be linearly extended to SVMs [18] and SVMs can apply ELM kernel to get better performance due to its *universal approximation capability* [10, 11, 21] and *classification capability* [20].

It is generally believed that all the ELM based algorithms consist of two major stages [22]: (1) random feature mapping

and (2) output weights calculation. The first stage generating feature mapping randomly is the key concept in ELM theory which differs from other feature learning algorithms. In view of the good properties of the ELM feature mapping, most existing ELM based classification algorithms can be viewed as *supervised learning in ELM feature space*. While, in [23], the *unsupervised learning in ELM feature space* is studied, drawing the conclusion that the proposed ELM k-Means algorithm and ELM NMF (nonnegative matrix factorization) clustering can get better clustering results than traditional algorithms in original feature space.

Recently, the volume of XML documents keeps explosively increasing in various kinds of web applications. Since the larger the training sample is, generally the better the learning model will be trained [24], it is a great challenge to implement distributed learning solutions to process massive XML datasets in parallel. MapReduce [25], introduced by Google to process parallelizable problems across huge datasets on clusters of computers, 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 global representation model and (2) implementing learning algorithms in ELM feature space.

To our best knowledge, this paper is the *first* to discuss massive XML documents mining problems. We present a distributed solution to XML representation and learning in ELM feature space. Since the raw XML datasets are stored on distributed file system, we propose algorithm DXRC to convert the XML documents into training samples in the form of XML representation model using a MapReduce job. With the converted training samples, we apply PELM [26] and POS-ELM [27] to realize supervised learning and propose a distributed k-Means in ELM feature space based on ELM k-Means proposed in [23]. The contributions can be summarized as follows.

(1) A *distributed representing algorithm* 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* is proposed based on ELM k-Means [23] to realize distributed clustering over massive XML documents in *ELM feature space*.

(4) Empirical and extensive comparison experiments are conducted on clusters to verify the performance of our solution.

The remainder of this paper is structured as follows. Section 2 introduces XML documents representation models and proposes a distributed converting algorithm to represent XML documents stored on distributed file system. Extreme Learning Machine 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 makes performance comparison among distributed classification algorithms and evaluates the proposed distributed clustering algorithm. Section 7 draws conclusions of this paper.

2. Distributed XML Representation

In this section, we first introduce representation model of XML documents and then propose a distributed converting algorithm, which is able to generate global feature vectors for all the XML documents stored on distributed file system.

2.1. XML Representation Model. For learning problems of texts, such as XML and plain documents, the first important task is to convert original documents into representation model. Vector Space Model (VSM) [1] is often used to represent plain text documents, which takes term occurrence statistics as feature vectors. However, representing an XML document in VSM directly will lose the structural information. Structured Link Vector Model (SLVM) is proposed in [2] based on VSM to represent semistructured documents, which contains both semantic and structural information. SLVM is defined as

$$\mathbf{d}_{\text{slvm}} = \langle \mathbf{d}_1, \ldots, \mathbf{d}_n \rangle, \tag{1}$$

where $\mathbf{d}_i$ is a feature vector of the ith XML element calculated as

$$\mathbf{d}_i = \sum_i \left(\text{TF}\left(w_i, \text{doc}.e_j\right) \times \varepsilon_j \right) \text{IDF}\left(w_i\right), \tag{2}$$

where w_i is the ith term and ε_j is a unit vector corresponding to the element e_j.

In SLVM, each $\mathbf{d}_{\text{slvm}}$ is a feature matrix $\mathbf{R}^{n\times m}$, which is viewed as an array of VSMs. $\mathbf{d}_i$ consists of the feature terms corresponding to the same XML element, which is an m-dimensional feature vector in each element unit.

Based on SLVM, in [3], we proposed Reduced Structured Vector Space Model (RS-VSM), which not only inherits the advantages of representing structural information of SVLM, but also achieves a better performance due to the feature subset selection based on information gain. We also proposed Distribution Based Structured Vector Model (DSVM) in [4] to further strengthen the ability of representation. Two improved interact factors were designed, including Among Classes Discrimination (ACD) and Within Class Discrimination (WCD). Revised IDF was also introduced to indicate the importance of a feature term in other classes more precisely.

$\mathbf{d}^k_{u_i^j}$ is the kth term feature described as

$$\mathbf{d}^k_{u_i^j} = \sum_{j=1}^{m} \left(\text{TF}\left(w_k, \text{doc}.e_j\right) \cdot \varepsilon_j \right) \cdot \text{IDF}_{\text{ex}}\left(w_k, c\right) \cdot \rho_{\text{CD}}, \tag{3}$$

where m is the number of elements in document doc, $\text{doc}.e_j$ is the jth element e_j of doc, and ε_j, which is the unit vector

```
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 = itrEle.getKey();
(14)      foreach itrEleTF ∈ itrEle.getValue() do
(15)            term = itrEleTF.getKey();
(16)            times = itrEleTF.getValue();
(17)            emit(term, ⟨docID, element, times, sum⟩);
```

Algorithm 1: Mapper of DXRC.

```
Input: ⟨term, list(⟨docID, element, times, sum⟩)⟩
Output: training samples matrix in the form of ⟨position, tfidf⟩
(1) Initiate HashMap mapDocEleTF;
(2) Initiate HashMap mapTDocs;
(3) N = DistributedCache.get("totalDocsNum");
(4) weights = DistributedCache.get("elementWeightsVector");
(5) foreach itr ∈ list do
(6)       weightedDocEleTF = weights[docId, element] * itr.times/itr.sum;
(7)       mapDocEleTF.put(⟨docId, element⟩, weightedDocEleTF);
(8)       if mapTDocs.containsKey(docId) then
(9)             newTimes = mapTF.get(docId) + irt.getValue().times;
(10)            mapTDocs.put(docID, newTimes);
(11)      else
(12)            mapTDocs.put(docID, itr.getValue().times);
(13) docsNumber = mapTDocs.size();
(14) idf = log (mapTDocsSize/N);
(15) foreach itrDocEleTF ∈ mapDocEleTF do
(16)      position = itrDocEleTF.getKey();
(17)      tfidf = itrDocEleTF.getValue() × idf;
(18)      emit(position, tfidf);
```

Algorithm 2: Reducer of DXRC.

of doc.e_j in SLVM, is now the dot product of s-dimensional unit vector and s-dimensional weight vector. $\mathrm{IDF}_{\mathrm{ex}}(w_i, c)$ is the revised IDF. The factor ρ_{CD} is the distribution modifying factor, which equals the reciprocal of arithmetic product of WCD and ACD. The detailed calculation of $\mathbf{d}^k_{u_i^j}$ can be found in [4].

2.2. Distributed Converting Algorithm. In this section, we propose a distributed converting algorithm, named Distributed XML Representation Converting (DXRC), to calculate TFIDF [28] values of DSVM based on MapReduce. Since the volume of XML documents is so large that the representation model cannot be generated on a single machine, DXRC realizes the representation of XML documents in the form of DSVM in parallel. The map function and reduce function of DXRC are presented as Algorithms 1 and 2, respectively.

The map function in Algorithm 1 accepts key-value pairs and the MapReduce job context 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 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, the numbers of *mapEle* items and XML elements are the same;

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 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 documents N (Line 3) and the vector *weights* (Line 4), which indicates the weights of all the elements in each XML document, are obtained through distributed cache defined in MapReduce job configuration. Since reduce now has 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. Finally, the XML representation DSVM can be built by this matrix and factor ρ_{CD}, uploaded onto distributed file system, and used as the input of the training model.

3. ELM Feature Mapping

Extreme Learning Machine (ELM) randomly generates parameters of the single-hidden layer feedforward networks without iteratively tuning to gain extremely fast learning speed. The output weights can be calculated by matrix multiplication after the training samples are mapped into *ELM feature space*.

Given N arbitrary samples $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^{n\times m}$, ELM is modeled as

$$\sum_{i=1}^{L}\beta_i G(\mathbf{w}_i, b_i, \mathbf{x}) = \boldsymbol{\beta} h(\mathbf{x}), \tag{4}$$

where L is the number of hidden layer nodes, β_i is the output weight from the ith hidden node to the output node, $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^T$ is the input weight vector, and b_i is the bias of ith hidden node. $G(\mathbf{x})$ is the activation function to generate mapping neurons, which can be any nonlinear piecewise continuous functions [22], including Sigmoid function (5) and Gaussian function (6), as follows:

$$G_{\mathrm{Sigmoid}}(\mathbf{w}, b, \mathbf{x}) = \frac{1}{1+\exp\left(-\left(\mathbf{w}^T\mathbf{x}+b\right)\right)}, \tag{5}$$

$$G_{\mathrm{Gaussian}}(\mathbf{w}, b, \mathbf{x}) = \exp\left(-b\left\|\mathbf{x}-\mathbf{a}\right\|\right). \tag{6}$$

Figure 1 shows the structure of ELM with multiple output nodes and the feature mapping process. The three layers

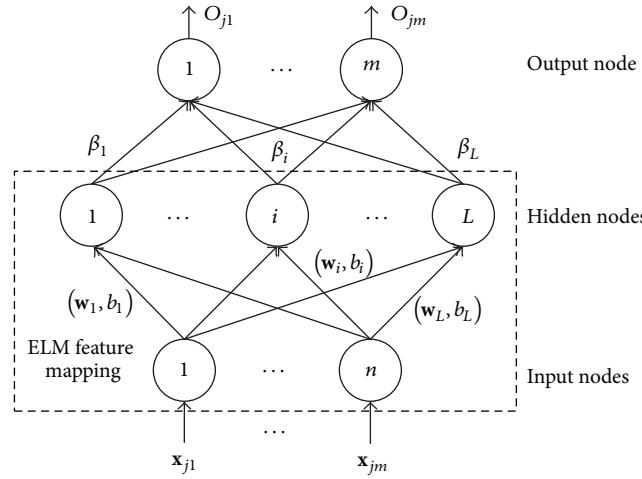


Figure 1: ELM structure and ELM feature mapping.

of ELM network are input layer, hidden layer, and output layer. The n input nodes correspond to the n-dimensional data space of original samples, while L hidden nodes correspond to the L-dimensional *ELM feature space*. With the m-dimensional output space, the decision function outputs the class label of the samples.

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

$$\mathbf{H} = \begin{bmatrix} h(\mathbf{x}_1) \\ \vdots \\ h(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} G(\mathbf{w}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{w}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{w}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{w}_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N\times L}. \tag{7}$$

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 [26] and POS-ELM [27].

4.1. Supervised Learning in ELM Feature Space. Most of the existing ELM algorithms aim at supervised learning, that is, classification and regression. In supervised learning applications, ELM is to minimize the training error and the norm of the output weights [5, 6], that is,

$$\text{Minimize: } \left\|\mathbf{H}\boldsymbol{\beta}-\mathbf{T}\right\|^2, \ \left\|\boldsymbol{\beta}\right\|, \tag{8}$$

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

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

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}, \tag{9}$$

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

ELM for classification is presented as Algorithm 3.

The output weight of ELM can also be calculated as

$$\boldsymbol{\beta} = \mathbf{H}^T\left(\frac{\mathbf{I}}{C}+\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T}, \tag{10}$$

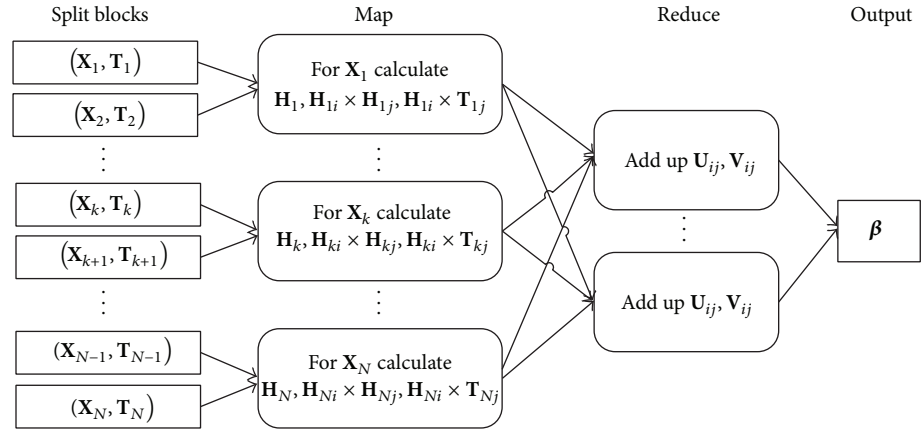


Figure 2: Parallel ELM.

(1) **for** $i = 1$ to L **do**
(2) Randomly assign input weight $\mathbf{w}_i$ and bias b_i;
(3) Calculate ELM feature space $\mathbf{H}$;
(4) Calculate $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$;

Algorithm 3: Extreme Learning Machine for classification.

where, according to the ridge regression theory [29], the diagonal of a symmetric matrix can be incremented by a biasing constant $1/C$ to gain better stability and generalization performance [18].

For the case in which the number of training samples is much larger than the dimensionality of the feature space, considering the computation cost, the output weight calculation equation can be rewritten as

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}. \tag{11}$$

4.2. Distributed Implementations. Some existing works have introduced distributed implementations of various ELM algorithms. The original ELM was parallelized by PELM in [26]; Online Sequential ELM (OS-ELM) was implemented on MapReduce as POS-ELM in [27].

4.2.1. Parallel ELM. In the original ELM algorithm, in the case that the number of training samples is much larger than the dimensionality of ELM feature space and $\mathbf{H}^T\mathbf{H}$ is nonsingular, the major cost is the calculation of Moore-Penrose generalized inverse of matrix $\mathbf{H}$, where the orthogonal projection method is used as (11). Thus the matrix multiplication $\mathbf{U} = \mathbf{H}^T\mathbf{H}$ and $\mathbf{V} = \mathbf{H}^T\mathbf{T}$ can be calculated by a MapReduce job. In map function, each term of $\mathbf{U}$ and $\mathbf{V}$ can be expressed as follows [26]:

$$\begin{aligned}\mathbf{U}[i][j] &= \sum_{k=1}^{N}\mathbf{H}[k][i] \times \mathbf{H}[k][j],\\ \mathbf{V}[i][j] &= \sum_{k=1}^{N}\mathbf{H}[k][i] \times \mathbf{T}[k][j].\end{aligned} \tag{12}$$

In reduce function, all the intermediate results are merged and added up according to the corresponding elements of the result matrix. Since the training input matrix $\mathbf{X}$ is stored by sample on different machines, the calculation can be parallelized and executed by the MapReduce job. The calculation procedure is demonstrated as Figure 2.

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

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \mathbf{P}_{k+1}\mathbf{H}_{k+1}^T\left(\mathbf{T}_{k+1} - \mathbf{H}_{k+1}\boldsymbol{\beta}_k\right), \tag{13}$$

where

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k\mathbf{H}_{k+1}^T\left(\mathbf{I} + \mathbf{H}_{k+1}\mathbf{P}_k\mathbf{H}_{k+1}^T\right)^{-1}\mathbf{H}_{k+1}\mathbf{P}_k. \tag{14}$$

The calculation procedure of POS-ELM is shown in Figure 3.

5. Distributed Clustering in ELM Feature Space

In this section, in order to improve the efficiency of clustering massive XML datasets, we also propose a parallel implementation of ELM k-Means algorithm, named Distributed ELM k-Means (DEK) in this section.

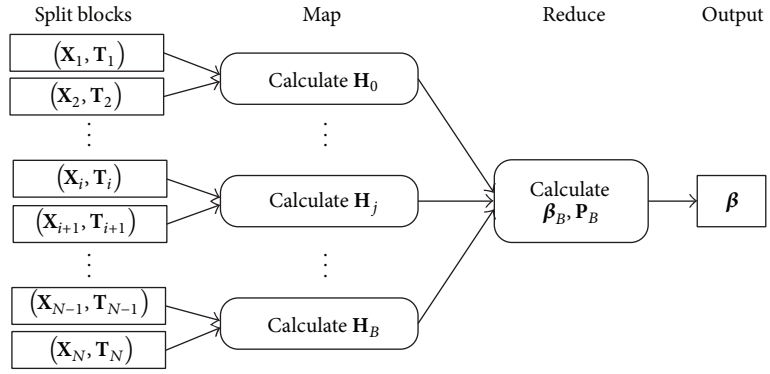


Figure 3: Parallel Online Sequential ELM.

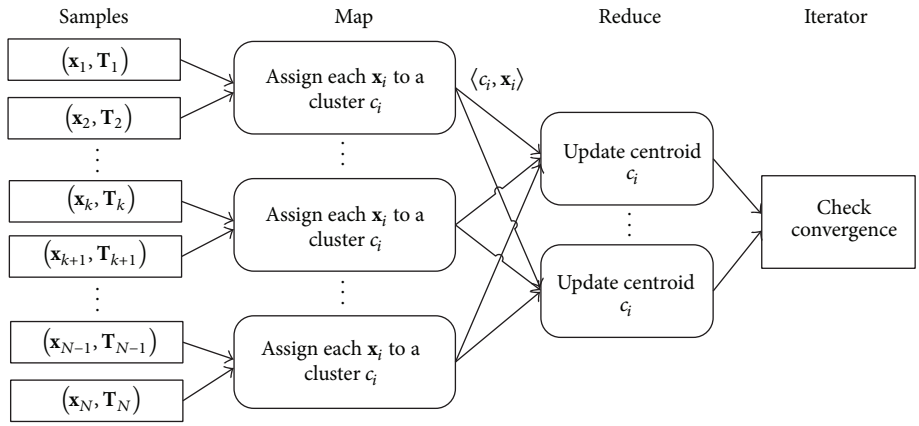


Figure 4: Distributed ELM k-Means.

5.1. Unsupervised Learning in ELM Feature Space. It is believed that transforming nonlinear data into some high dimensional feature space increases the probability of the linear separability. However, many Mercer kernel based clustering algorithms are usually not efficient for computation, since the feature mapping is always implicit and cannot be guaranteed to satisfy the universal approximation condition. Thus, [23] holds that explicit feature mapping like ELM feature mapping is more appropriate.

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

5.2. Distributed ELM k-Means. For the applications of massive XML documents clustering, implementation of unsupervised learning methods to MapReduce is a key part of the problem. Since ELM feature mapping is extremely fast with good generalization performance and universal approximation ability, in this section, we propose Distributed ELM k-Means (DEK) based on ELM k-Means [23].

In DEK algorithm, the training samples of XML documents are distributedly stored on distributed file system. Each $(\mathbf{x}_i, T_i)$ represents a training sample $\mathbf{x}_i$ *in ELM feature space* with its corresponding class label T_i. With a set of initiated k cluster centroids, in the *map* phase, the distances between each centroid and each training sample stored on its own site are calculated. Then the sample is assigned to the centroid with the shortest distance. In the *reduce* phase, all the samples assigned to the same centroid are collected in the same reducer. Then a new centroid of each cluster is calculated, with which the set of k cluster centroids are updated. That is, a round of MapReduce job updates the set of k cluster centroids once. In the next round of MapReduce job, the updated set of centroids is updated again and this procedure will be repeated until convergence or up to maximum number of iterations (Figure 4).

Algorithm 4 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 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 5 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

Input: Training samples $\mathbf{X}$, k centroids $\mathbf{C}$
Output: ⟨centroid $c_{\max}$, sample $\mathbf{x}_i$⟩
(1) **foreach** $\mathbf{x}_i \in \mathbf{X}$ **do**
(2) **foreach** $c_j \in \mathbf{C}$ **do**
(3) Calculate distance d_{ij} between $\mathbf{x}_i$ and c_j;
(4) Assign $\mathbf{x}_i$ to the cluster $c_{\max}$ with $\max_j(d_{ij})$;
(5) Emit $\langle c_{\max}, \mathbf{x}_i \rangle$;

Algorithm 4: Mapper of DEK.

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

6. Performance Evaluation

All the experiments are conducted to compare the performance in the following aspects:

(i) *scalability* evaluation of proposed Distributed XML Representation Converting (DXRC),

(ii) *scalability* comparison between PELM and POS-ELM in massive XML documents classification problem,

(iii) *supervised learning performance* of PELM and POS-ELM in massive XML documents classification problem,

(iv) *scalability* evaluation of proposed Distributed ELM k-Means (DEK),

(v) *unsupervised learning performance* of DEK in massive XML documents clustering problem.

6.1. Experiments Setup

6.1.1. Environment. All the experiments on distributed XML representation converting and distributed learning in ELM feature space are conducted on a Hadoop (Apache Hadoop, http://hadoop.apache.org/) cluster of nine machines, which consists of one master node and eight slave nodes. Each machine is equipped with an Intel Quad Core 2.66 GHZ CPU, 4 GB of memory, and CentOS 5.6 as operating system. All the computers are connected via a high speed Gigabit network. The MapReduce framework is configured with Hadoop version 0.20.2 and Java version 1.6.0_24.

6.1.2. Datasets. Three datasets of XML documents are used as original datasets, which are *Wikipedia XML Corpus* provided by INEX, *IBM DeveloperWorks* (http://www.ibm.com/developerworks/) articles, and *ABC News* (http://abcnews.go.com/). *Wikipedia XML Corpus* is composed of around 96,000 XML documents classified into 21 classes. We also fetched RSS feeds of news and articles in the format of XML from *IBM DeveloperWorks* and *ABC News* official web sites. Each XML document of the RSS feeds is composed of elements of title, author, summary, publish information, and so forth. In order to compare the performance of the algorithms over different datasets, we choose the same numbers of XML documents out of all the three datasets, which are 6 classes and 500 documents in each class.

6.1.3. Parameters. According to the universal approximation conditions and classification capability of ELM, a large number of hidden nodes guarantee that the data can be linearly separated [23], especially for the learning problems on high-dimensional training samples like XML documents. Thus, after a set of experiments for parameter setting, the only parameter of learning algorithms in ELM feature space, that is, the number of hidden nodes L, is set to 800.

6.1.4. Evaluation Criteria. To clearly evaluate the performance, three sets of evaluation criteria are utilized.

(1) For scalability evaluation, we compare the criteria of speedup, sizeup, and scaleup. *Speedup* indicates the scalability when increasing the number of running machines, which is measured as

$$\text{Speedup}\,(m) = \frac{\text{running time on 1 machine}}{\text{running time on } m \text{ machines}}. \quad (15)$$

Sizeup indicates the scalability when increasing the data size, which is measured as

$$\text{Sizeup}\,(m) = \frac{\text{running time over } m \text{ units of data}}{\text{running time over one unit of data}}. \quad (16)$$

Scaleup is to measure the scalability of processing m-times larger data on an m-times larger cluster, which is calculated as

$$\text{Scaleup}\,(m) = \frac{\text{running time over one unit of data on one machine}}{\text{running time over } m \text{ units of data on } m \text{ machines}}. \quad (17)$$

(2) For classification problems, accuracy, recall, and F-measure are used to evaluate the performance of supervised learning performance in ELM feature space. *Accuracy* indicates the overall ratio of correctly classified samples, which is measured as

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total samples number}}. \quad (18)$$

Recall is the ratio of the samples with a specific class label to the ones classified into this class, which is measured as

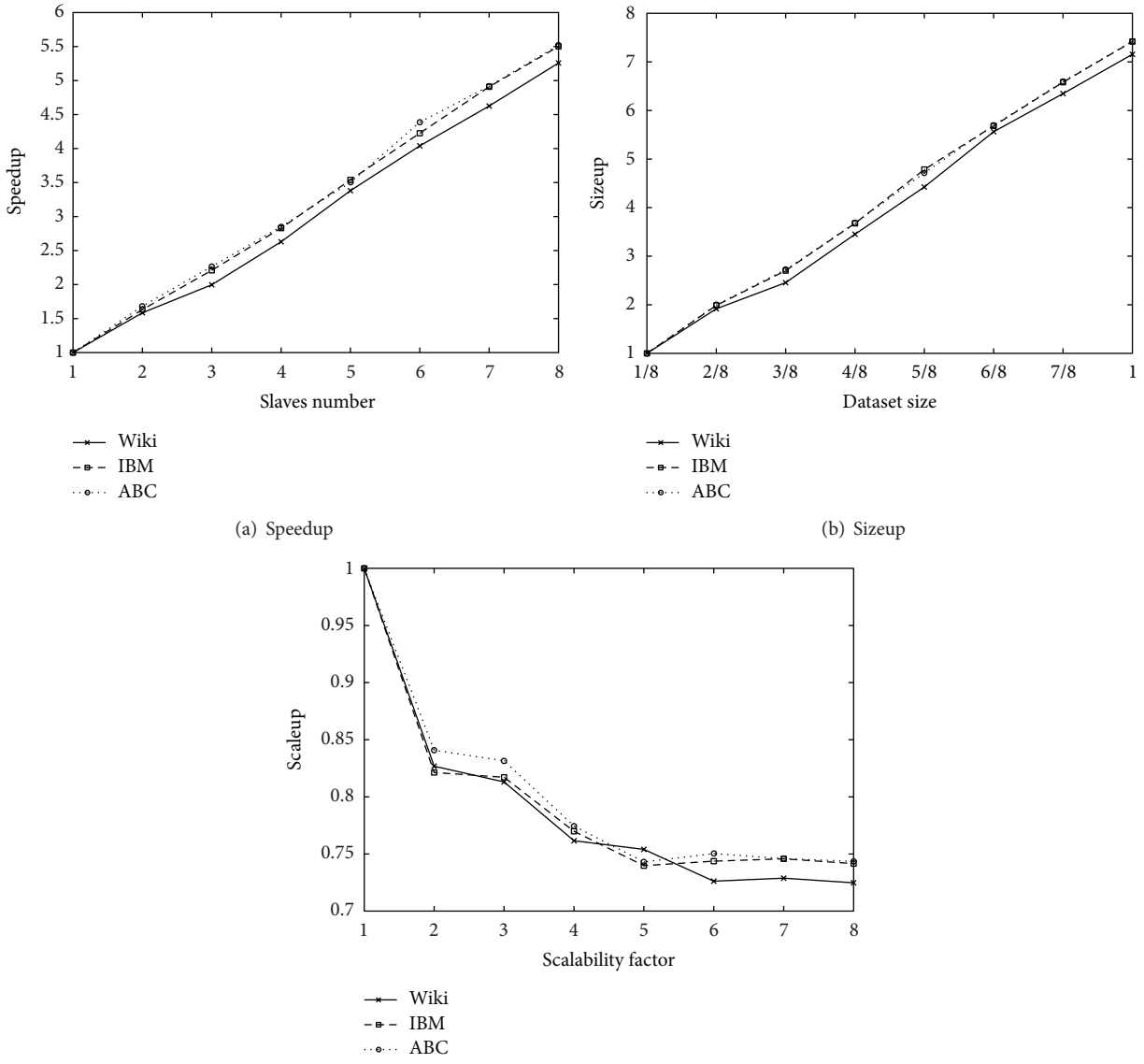
$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False}}. \quad (19)$$

F-measure is to measure the overall performance considering both accuracy and recall, which is calculated as

$$F\text{-measure} = \frac{2 \times \text{accuracy} \times \text{recall}}{\text{accuracy} + \text{recall}}. \quad (20)$$

Input: ⟨centroid c_j, samples list($\mathbf{x}$)⟩
Output: Updated set of centroids $\mathbf{C}_{\text{updated}}$
(1) **foreach** $\mathbf{x}_i \in$ list($\mathbf{x}$) **do**
(2) Add $\mathbf{x}_i$ to squared sum S;
(3) Calculate c_j^{updated} of cluster c_j as $c_j^{\text{updated}} = S/\text{list}(\mathbf{x}).length$;

Algorithm 5: Reducer of DEK.



(a) Speedup

(b) Sizeup

(c) Scaleup

Figure 5: Scalability of DXRC.

(3) For clustering problems, since each sample in the datasets we used in our experiments is assigned with a class label, we treat this class label as the cluster label. Thus, the same evaluation criteria are used for clustering problems as for classification problems.

6.2. Evaluation Results

6.2.1. Scalability of DXRC. The scalability of representation converting algorithm DXRC is first evaluated. Figure 5(a) demonstrates the speedup of DXRC. As the number of

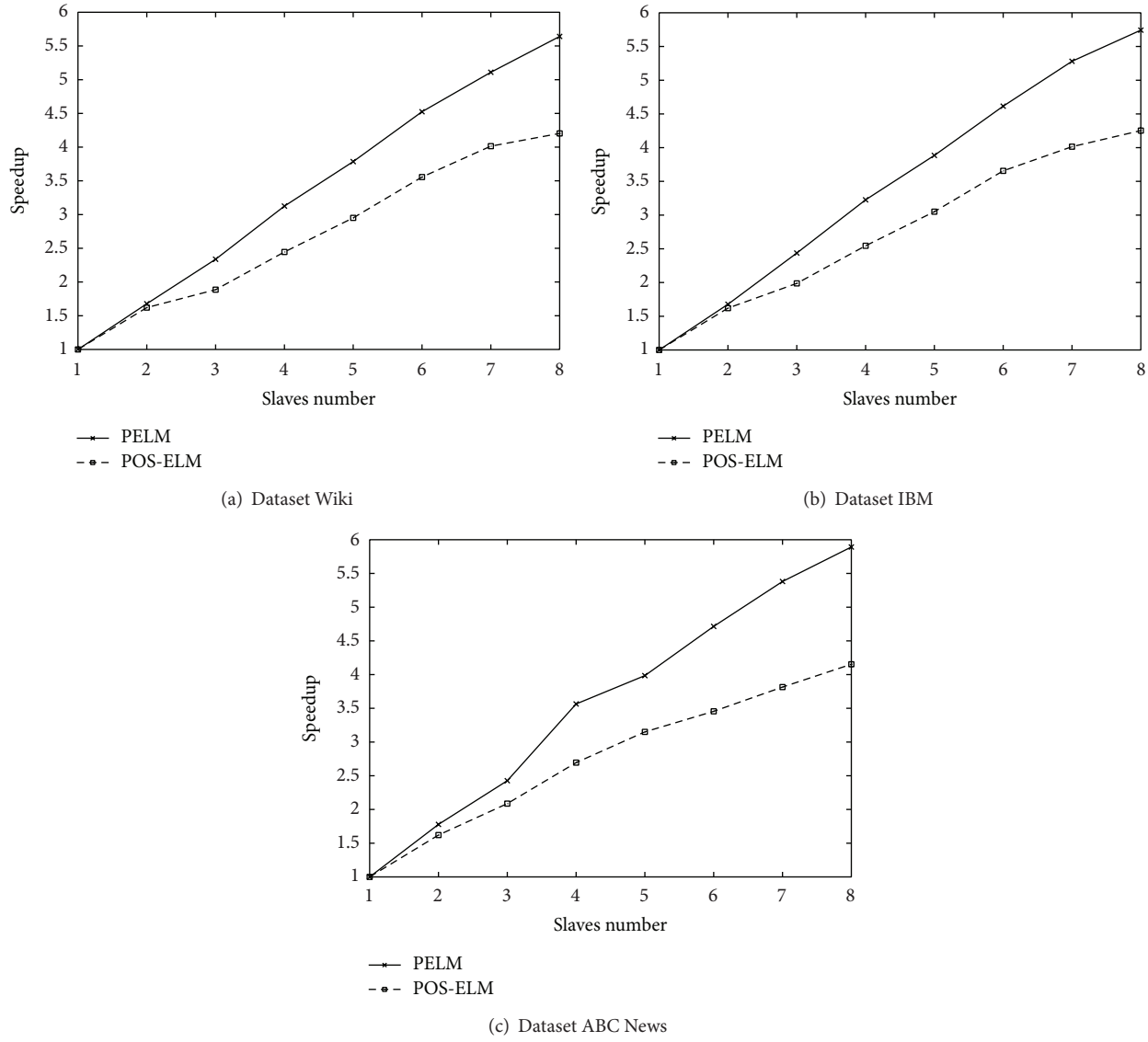


Figure 6: Comparison of speedup between PELM and POS-ELM.

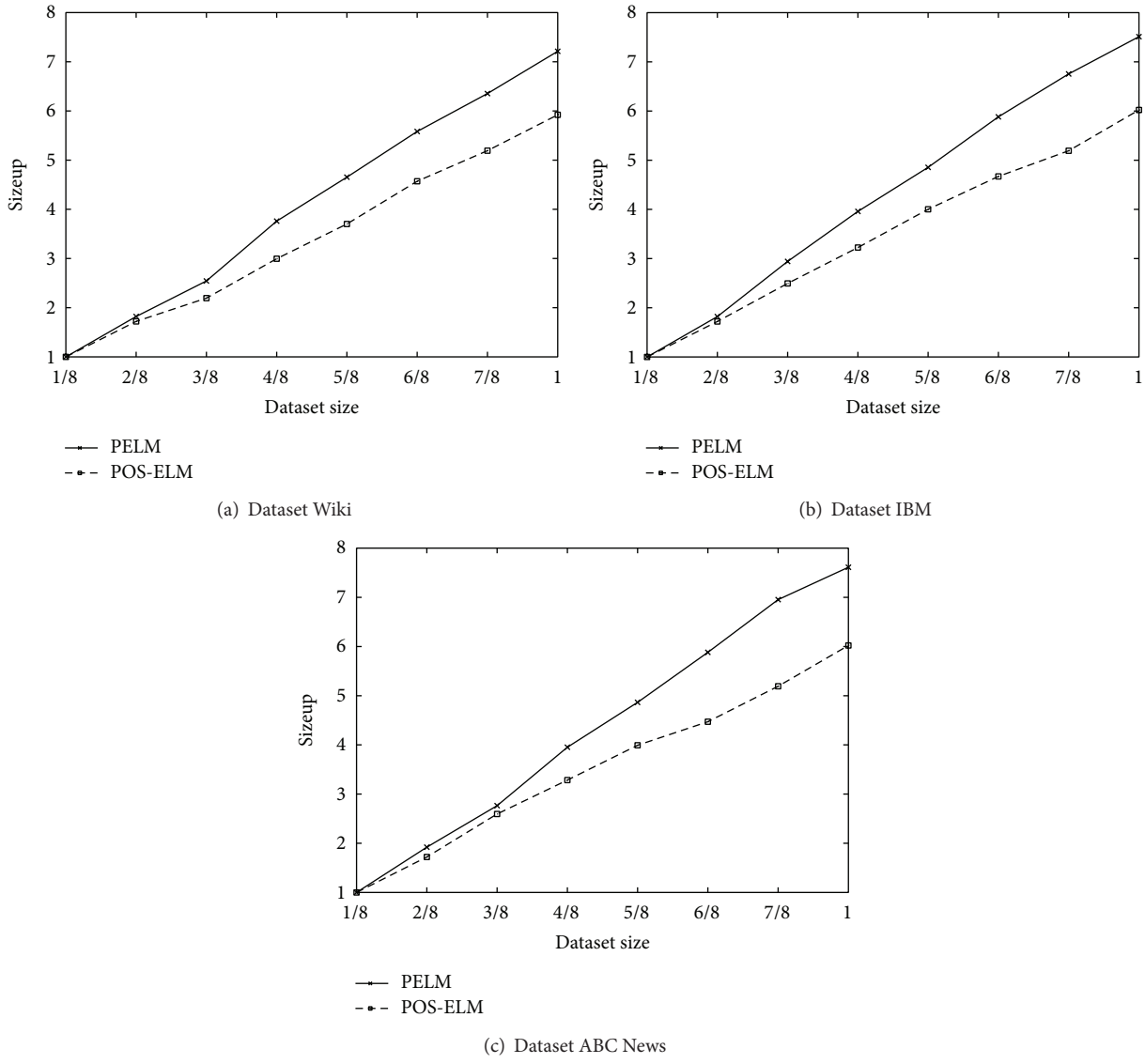
the slave nodes varies from one to eight, the speedup tends to be approximately linear at first, but the growth slows down due to the increasing cost of network communications among more and more working machines. But, in general, DXRC gains good speedup. Figure 5(b) presents the sizeup of DXRC. The x-axis denotes the percentage against the whole datasets. That is, 1 is the full size of original dataset; 0.5 indicates half of the original dataset, in which the samples are randomly chosen. With a fixed number of slave machines, which is eight, the evaluation result shows good sizeup of DXRC. Since the scaleup in distributed implementation cannot stick to 1 in practice, in Figure 5(c), the scaleup of DXRC drops slowly when the number of slave nodes and the size of dataset increase, which indicates a good scaleup of DXRC.

Note that the representation ability and performance influence on XML documents classification of DSVM applied in DXRC can be found in our previous work [4].

6.2.2. Scalability of Massive XML Classification in ELM Feature Space. With the training samples converted by algorithm DXRC, a classifier of massive XML documents can be trained based on MapRedcue. The speedup comparison between PELM and POS-ELM on three datasets is presented in Figure 6.

Algorithm PELM, which implements original ELM on MapReduce, requires calculating the inverse of ELM feature space matrix, while PEO-ELM makes use of the idea of online sequential processing to realize parallel computation without communication and requires calculating output weight $\boldsymbol{\beta}$ and the auxiliary matrix $\mathbf{P}$ iteratively in a single reducer. 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 7 demonstrates the sizeup comparison between PELM and POS-ELM. From this figure, we find that



(a) Dataset Wiki

(b) Dataset IBM

(c) Dataset ABC News

Figure 7: Comparison of sizeup between PELM and POS-ELM.

the sizeup of PELM is better than POS-ELM on all the three datasets.

For the scaleup comparison, Figure 8 demonstrates that both PELM and POS-ELM have good scaleup performance, and PELM outperforms POS-ELM on each of the three datasets.

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

6.2.3. Performance of Massive XML Classification in ELM Feature Space. The parallel implementation of both PELM and POS-ELM does not invade the computation theory of original ELM and OS-ELM, respectively; that is, the classification performances of PELM and POS-ELM are nearly the same as their corresponding centralized algorithms, respectively. The classification results are shown in Table 1.

From the table we can see that PELM slightly outperforms POS-ELM, because the iterative matrix operations of output weight $\boldsymbol{\beta}$ in POS-ELM cause loss of calculation accuracy. However, for massive XML documents classification applications, since both the extraction and reduction of XML document features are complicated, both PELM and POS-ELM provide satisfactory classification performance.

6.2.4. Scalability of Massive XML Clustering in ELM Feature Space. In this set of experiments, we evaluate the proposed distributed clustering algorithm in ELM feature space, that is, distributed ELM k-Means. In theory, the scalability of distributed k-Means in ELM feature space and in original feature space is the same, since the only difference is the feature space of the training samples, which has no influence on the computation complexity. Thus, we only present the scalability of DEK without comparison with distributed k-Means in original feature space.

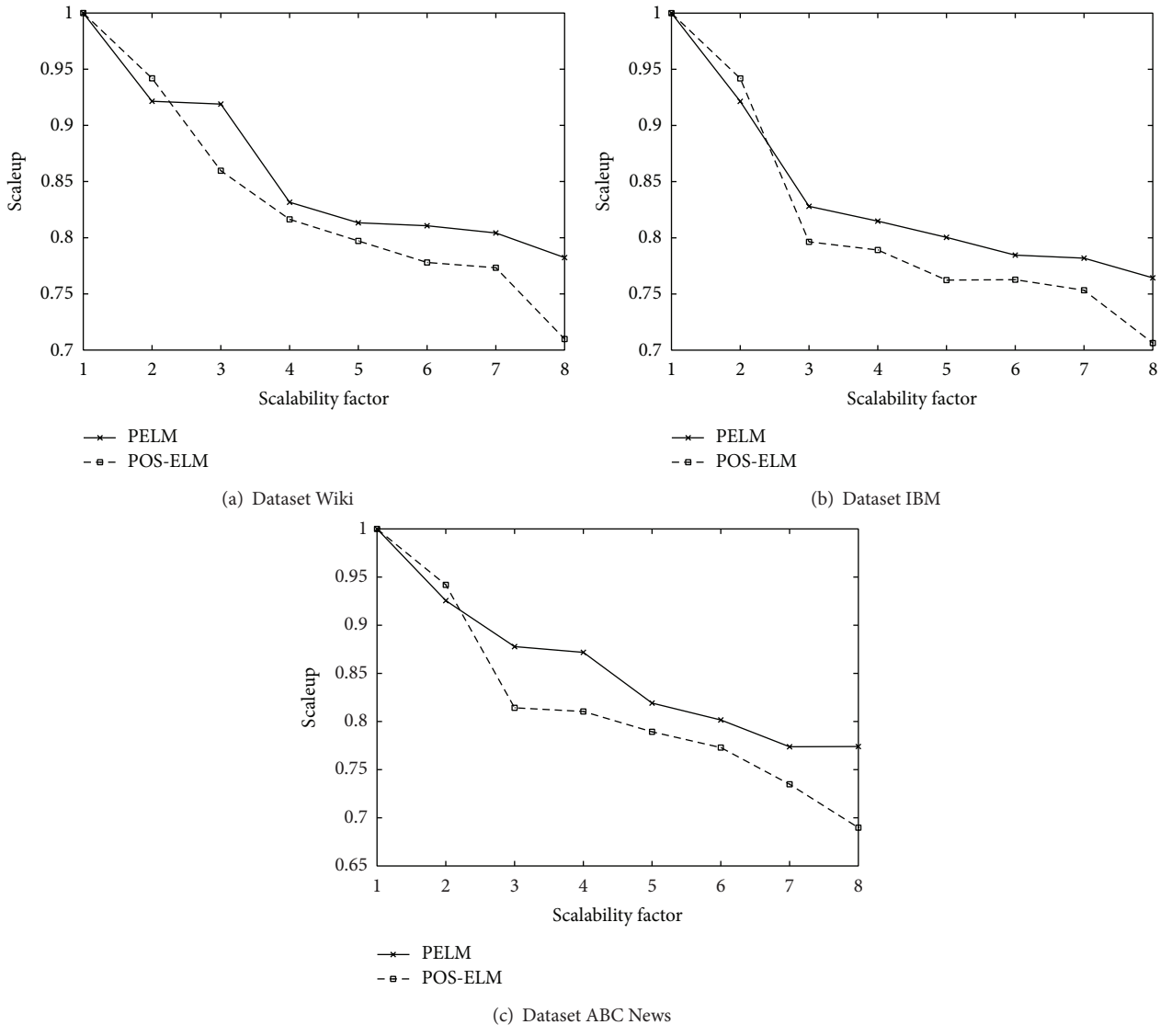


Figure 8: Comparison of scaleup between PELM and POS-ELM.

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

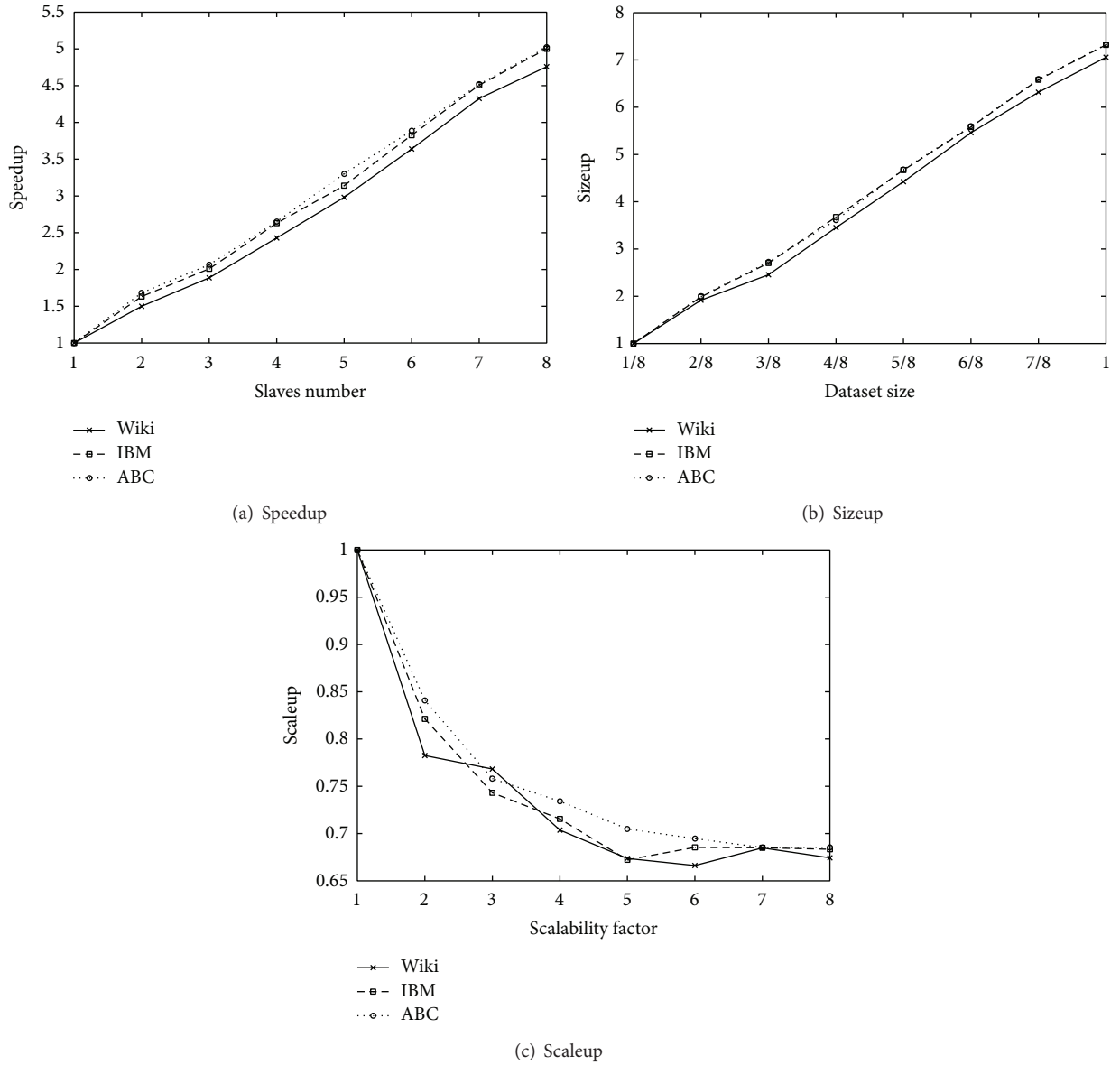
Datasets	PELM			POS-ELM		
	Accuracy	Recall	F-measure	Accuracy	Recall	F-measure
Wikipedia	0.7886	0.7563	0.7721	0.7923	0.7745	0.7833
IBM DeveloperWorks	0.7705	0.8145	0.7919	0.7711	0.7863	0.7891
ABC News	0.8681	0.8517	0.8598	0.8517	0.8335	0.8425

The scalability of DEK is evaluated on all the three datasets in terms of speedup in Figure 9(a), sizeup in Figure 9(b), and scaleup in Figure 9(c). The experimental results all demonstrate good scalability of DEK for massive XML documents clustering applications.

6.2.5. Performance of Massive XML Clustering in ELM Feature Space. Clustering performance comparison between distributed clustering in ELM feature space and clustering in original feature space is made for massive XML documents clustering applications in this set of experiments. Note that, since the manual relabeling of the massive XML dataset is infeasible, we only evaluate the clustering quality with the original number of classes, which is six. 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.

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 based on MapReduce is



(a) Speedup

(b) Sizeup

(c) Scaleup

Figure 9: Scalability of DEK.

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

discussed by proposing a distributed XML representation converting algorithm DXRC. The problem of massive XML documents classification in ELM feature space is studied by implementing PELM and POS-ELM, while, for the problem of 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.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research is partially supported by the National Natural Science Foundation of China under Grants 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] G. Salton and M. Mcgill, *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, NY, USA, 1984.

[2] J. Yang and X. Chen, "A semi-structured document model for text mining," *Journal of Computer Science and Technology*, vol. 17, no. 5, pp. 603–610, 2002.

[3] X.-G. Zhao, G. Wang, X. Bi, P. Gong, and Y. Zhao, "XML document classification based on ELM," *Neurocomputing*, vol. 74, no. 16, pp. 2444–2451, 2011.

[4] X. Zhao, X. Bi, and B. Qiao, "Probability based voting extreme learning machine for multiclass XML documents classification," *World Wide Web*, pp. 1–15, 2013.

[5] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. 2, pp. 985–990, July 2004.

[6] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, 2006.

[7] G.-B. Huang, Q.-Y. Zhu, K. Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, "Can threshold networks be trained directly?" *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 3, pp. 187–191, 2006.

[8] G. Feng, G.-B. Huang, Q. Lin, and R. K. L. Gay, "Error minimized extreme learning machine with growth of hidden nodes and incremental learning," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1352–1357, 2009.

[9] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran, "Online sequential fuzzy extreme learning machine for function approximation and classification problems," *IEEE Transactions on Systems, Man, and Cybernetics B: Cybernetics*, vol. 39, no. 4, pp. 1067–1072, 2009.

[10] G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, no. 16–18, pp. 3460–3468, 2008.

[11] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, no. 16-18, pp. 3056–3062, 2007.

[12] X.-G. Zhao, G. Wang, X. Bi, P. Gong, and Y. Zhao, "XML document classification based on ELM," *Neurocomputing*, vol. 74, no. 16, pp. 2444–2451, 2011.

[13] B. Lu, G. Wang, Y. Yuan, and D. Han, "Semantic concept detection for video based on extreme learning machine," *Neurocomputing*, vol. 102, pp. 176–183, 2013.

[14] Y. Lan, Z. Hu, Y. C. Soh, and G.-B. Huang, "An extreme learning machine approach for speaker recognition," *Neural Computing and Applications*, vol. 22, no. 3-4, pp. 417–425, 2013.

[15] W. Zong and G.-B. Huang, "Face recognition based on extreme learning machine," *Neurocomputing*, vol. 74, no. 16, pp. 2541–2551, 2011.

[16] G. Wang, Y. Zhao, and D. Wang, "A protein secondary structure prediction framework based on the extreme learning machine," *Neurocomputing*, vol. 72, no. 1–3, pp. 262–268, 2008.

[17] B. Wang, G. Wang, J. Li, and B. Wang, "Update strategy based on region classification using ELM for mobile object index," *Soft Computing*, vol. 16, no. 9, pp. 1607–1615, 2012.

[18] G.-B. Huang, X. Ding, and H. Zhou, "Optimization method based extreme learning machine for classification," *Neurocomputing*, vol. 74, no. 1–3, pp. 155–163, 2010.

[19] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[20] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 2, pp. 513–529, 2012.

[21] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[22] G. Huang, S. Song, J. N. D. Gupta, and C. Wu, "Semi-supervised and unsupervised extreme learning machines," *IEEE Transactions on Cybernetics*, 2014.

[23] Q. He, X. Jin, C. Du, F. Zhuang, and Z. Shi, "Clustering in extreme learning machine feature space," *Neurocomputing*, vol. 128, pp. 88–95, 2014.

[24] T. Ngo, "Data mining: practical machine learning tools and technique," in *ACM Sigsoft Software Engineering Notes*, I. H. Witten, E. Frank, and M. A. Hell, Eds., pp. 51–52, 3rd edition, 2011.

[25] J. Dean and S. Ghemawat, "MapReduce: simplied data processing on large clusters," in *Operating Systems Design and Implementation*, pp. 137–150, 2004.

[26] Q. He, T. Shang, F. Zhuang, and Z. Shi, "Parallel extreme learning machine for regression based on mapReduce," *Neurocomputing*, vol. 102, pp. 52–58, 2013.

[27] B. Wang, S. Huang, J. Qiu, Y. Liu, and G. Wang, "Parallel online sequential extreme learning machine based on MapReduce," *Neurocomputing*, 2014.

[28] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[29] A. E. Hoerl and R. W. Kennard, "Ridge regression: biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.