International Journal on Engineering Technologies and Informatics

A Distributed Feature Extraction Method Based on MapReduce

Research Article

Volume 5 Issue 2- 2024

Author Details

Yongchang Wang, Hongqiang Hu, Xuning Liu*

Shijiazhuang University, China

*Corresponding author

Yongchang Wang, Shijiazhuang University, China

Article History

Received: March 18, 2024 Accepted: March 21, 2024 Published: March 25, 2024

Abstract

For data with complex dimensions, Kernel Principal Component Analysis (KPCA) is a common method of feature extraction, it has a better generalization performance. However, it often face with significant difficulties in computation on large datasets. Therefore, we developed a parallel KPCA algorithm based on MapReduce (MRKPCA) in this paper, it's a parallel computing method which can run in a cluster. We apply our algorithm to an experiment of datasets of face recognition, the experimental results show that our algorithm can scale well and has better timeliness for some specific problems which require a lot of computation, and it can significantly reduce computation time.

Keywords: Dimension Reduction, Feature Extraction, KPCA, MapReduce

Introduction

In recent years, automatic face recognition has become a hot topic in the field of pattern recognition and image processing, researchers have proposed many methods for face recognition. The technology use camera to collect the image or video stream, and the image is detected and tracked automatically [1]. How to extract the feature of face image effectively is the key problem in the process of automatic face recognition. In the process of face recognition, the methods of extracting features include principal component analysis (PCA), kernel principal component analysis (KPCA), independent component analysis (ICA), linear discriminant analysis (LDA),...,etc. All these methods of feature extraction belong to the fields of dimension reduction [2].

Dimension reduction is a key step of feature extraction. Among the methods of dimension reduction, PCA is interesting because it serves as a foundation for many algorithms. However, PCA is a linear method, so many nonlinear PCA algorithms have been developed for overcoming this problem. Kernel PCA may be the most popular nonlinear method based on PCA by using a kernel function. With the arrival of the era of big data, the dimension of big data are higher and more complicated, the large volume of datasets makes computation by KPCA on very large scale datasets a challenging task. MapReduce is the most used model for big data processing and distributed computing. A large number of matrix calculations in KPCA can be paralleled by MapReduce, so we developed a distributed KPCA algorithm based on MapReduce to improve the efficiency of feature extraction on big data.

The Principle of PCA

The calculation process of PCA is as follows:

Let's define the variable $X=[x_1,x_2,\ldots,x_N]$, where X is a matrix of d -by- N and the dimension of each sample in X is d. For dimension reduction, we should use the data with k dimension to represent the original data of d dimension (k<d).When we use centered data (which means $\sum_i x_i=0$), we can define a covariance matrix of C as:

$$C = \frac{1}{N}\sum_i x_i x_i^T = \frac{1}{N}XX^T \qquad (1)$$

For calculating the eigenvalue of C, we can get

$$CU = U\Lambda \Rightarrow C = U\Lambda U^T = \sum_\alpha \lambda_\alpha u_\alpha u_\alpha{}^T \qquad (2)$$

Where $U=[u_1,u_2,..,u_d]$, Λ= diag $(\lambda_1,\lambda_2,\ldots,\lambda_d)$ and we should note that C, U, $\Lambda$are matrix of d -by- N.

For dimension reduction, we can use the corresponding eigenvectors of the first K eigenvalues of Λ ,and we get $U_k=[u1,u2,..,u_k]$. When projecting the data xi of d dimension to the space of principal component of k dimension, we can have the data in new coordinate, denoted by:

$$y_i = u_k{}^T x_i \qquad (3)$$

The new data y_i of k dimension (k<d) can represent the principle component of the original data of d dimension, so we achieve the purpose of dimension reduction, and the algorithm of PCA is described as algorithm 1.

Algorithm 1: Principal Component Analysis (PCA)

Given: n samples $x_i \in R_d$, i=1,…,n

1. Make the original data into a matrix of X d_x N
2. Each line of X should minus the average value of the line
3. Compute the covariance matrix C=1/N $\sum_i x_i x_i^T$ =1/N XX^T
4. Compute the eigenvalues λi and corresponding eigenvectors U_i of C
5. According to size of the λi, make the corresponding eigenvector Ui into a matrix row by row, take the first k rows of the matrix U, denoted by U_k
6. Compute and get the result of $y_i = \Lambda u_k \Lambda^T x_i$

The Principle of KPCA

Many nonlinear PCA algorithms have been developed to solve the limitation of PCA [3]. Kernel PCA (KPCA) is one of them. The method does not directly calculate the eigenvectors of the sample covariance matrix, it is transformed into the problem of finding eigenvalues and eigenvectors of the kernel matrix, so it need not compute feature vectors in the whole feature space. Compared with other nonlinear feature extraction methods, the KPCA method does not need to solve nonlinear optimization problems. Commonly used kernel functions include [4]:

(1) Polynomial kernel function

$$K(x_i, x_j) = (x_i \cdot x_j + b)^p \quad (4)$$

(2) Radial basis function

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right) \quad (5)$$

(3) Sigmoid kernel function

$$K(x, x_i) = tanh(\beta x_i + b) \quad (6)$$

(4) Linear kernel function

$$K(x, x_i) = x \cdot x_i \quad (7)$$

Firstly KPCA use Φ(x_k) to make nonlinear transformation for each sample in x_k (K =1, 2,…,m), Φ realize the mapping from sample space Rn to feature space F [5], for the new sample space, the covariance matrix now become:

$$C = \frac{1}{m} \sum_i^m \Phi(x_i)\, \Phi(x_i)^T \quad (8)$$

The eigenvalues and eigenvectors of the formula (8) are satisfied

$$\lambda V = CV \quad (9)$$

Multiply both sides of the formula (9) byΥ(x k), we have

$$\lambda(\Phi(x_k)V) = \Phi(x_k)CV \quad (10)$$

For all feature vectors thatλ≠0, we have

$$V = \sum_i^m \alpha_i \Phi(x_i) \quad (11)$$

By introducing kernel function Kij, we get

$$K_{ij} = K(x_i, x_j) = \Phi(x_i)\Phi(x_j) \quad (12)$$

Compute eigenvalues and eigenvectors of the kernel matrix K, we have

$$\lambda\alpha = K\alpha \quad (13)$$

α is the feature vector of K matrix, for any vector x, the projection on principal direction Φ (x) in the feature space is

$$V \cdot \Phi(x) = \sum_i^m \alpha_i \Phi(x_i)\, \Phi(x) = \sum_i^m \alpha_i\, K(x_i, x) \quad (14)$$

The general rules for selecting the number of principal element s is

$$\left(\sum_{k=1}^s \lambda_k / \sum_{k=1}^m \lambda_k\right) > E \quad (15)$$

The value of E is usually greater than 85%. The derivation process above is under the assumption that $\sum_{k=1}^m \Phi(x_k) = 0$ but this is not the case in actual situation, so the K in formula 6 should be replaced by K ¯, we have

$$\bar{K} = K - LK - KL + LKL = (1 - L)K(1 - L) \equiv PKP \quad (16)$$

Introduced the projection matrix P=1−L and L is a m-by-m unit matrix whose coefficient of 1/m, so the centered version of the kernel matrix become K¯=PKP. Combining formula (8), (10), (11), and (16) gives:

$$\bar{K}\bar{K}\alpha = m\lambda\bar{K}\alpha \quad (17)$$

Now solution become as bellow:

$$\bar{K}\alpha = m\lambda\alpha \quad (18)$$

From formula (18), we can convert the nonlinear problem to the linear one by introducing kernel function.

Although KPCA is an effective method for solving nonlinear spatial problems, the computation of KPCA on large scale datasets is so hard and time consuming [6], so we developed a new KPCA algorithm based on MapReduce, and our algorithm takes polynomial kernel as kernel function. The principle of MapReduce and our method is discussed below.

Distributed KPCA Algorithm Based on MapReduce

MapReduce is a programming model based on Hadoop which developed by Google, mainly used for big data processing [7]. As the name suggests, the two main stages of MapReduce include a map phase and a reduce phase, corresponding to map function and a reduce function [8]. The framework of MapReduce can be described as Figure1. From Figure 1, we can see that the data input by user is mapped into an intermediate form by the map function, then the intermediate results can be made as the input of the reduce function. Finally, the reduce function outputs the result of the program. All this is done by is the basic data structure of MapReduce called <key, value> pair. A map function is invoked only once for each input<key, value>pair in map phase, the intermediate results are generated in pairs and are shuffled by the underlying system [9]. In reduce phase, a reduce function will merge, group and sort the values by keys, and generate <key, value> pairs.

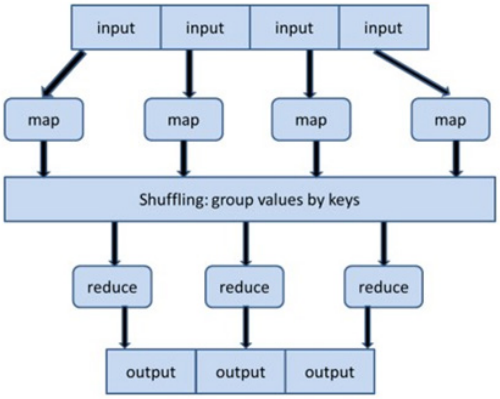

Figure1: Framework of MapReduce.

The process of change for <key, value> pairs in the map stage is shown in the formula (19):

$$map\ \langle key1,value1\rangle \Rightarrow list\langle key2,value2\rangle \quad (19)$$

The process of change for <key, value> pairs in the map stage can be described by formula 20:

$$reduce\ \langle key2,list(value2)\rangle \Rightarrow \langle key2,value3\rangle \quad (20)$$

We know that the multiply of matrix can be parallel executed, so we develop a distributed KPCA algorithm based on MapReduce, we call it MRKPCA, the algorithm is composed of three sub algorithms, Namely Map-function, Combine-function and Reduce-function.

A. Map-function.The input datasets is stored in HDFS in the form of <key, value> pairs, each pair represents a record in the datasets [10]. The key is the offset in bytes to the start point of this record in the data file, and the value is the content of this record in String. Datasets are split and broadcast to all map-function. For each map task, MRKPCA construct an array containing the information about the covariance matrix, then a map-function can can compute the index of each sample. The intermediate result are split into two parts: the row index and the column index of the value. The executing process of map function is shown as algorithm 2.

Algorithm2: map (key, value)

Input: the row position of the covariance matrixφ(x) andφT(x), the column position of the covariance matrixφ(x) andφT(x), the value of the covariance matrixφ(x) andφT(x)

Output: <key, value> pair, where the key is the position index of the matrixφ(x) andφT(x) and value is the intermediate value of the covariance matrixφ(x) andφT(x)

1. Construct the covariance matrixφ(x) andφT(x) from the original data.
2. Define a String array value to store the position index and the value ofφ(x) andφT(x).
3. Define three Strings such as “rowindex”, “colindex”, “elevalue” to initialize value;
4. For i=0 to matrixφ(x). length do
5. collect (rowindex + i +colindex+elevalue));
6. End For;
7. Repeat the process of step 4;
8. Compute the value of the sum of different dimensions;
9. Output < key, value > pair;
10. 9. End

Step 4 and Step 6 record location information for each value, in which the function collect returns the information of each data of matrix on each node; The intermediate result outputs in < key ,value > pair at data step 8.

B. Combine-function. We use a combiner to merge the intermediate data when each map task is finished. In combine function, we calculate key and value of the points assigned to the same cluster. To calculate the value of the matrix multiplication in each cluster, we record the number of samples in same cluster at each map task stage. The process of out combine function is shown in algorithm 3.

Algorithm 3: combine (key, V)

Input: key (the index of the cluster), N (the list of the samples assigned to the same cluster)

Output: < key, value > pair

1. Initialize the list N;
2. Initialize a counter num =0;
3. For i=0 to matrixφ(x). length do
4. int result = 0;
5. result +=φ(x) [i] *φ^T(x) [i];
6. collect (key, new Text (Integer.toString(result)));
7. End For
8. Replaceφ(x) with a kernel matrix K and repeat Step 3
9. Compute eigenvalues and eigenvectors of the kernel matrix K and get the projection vector Vj (j=1, 2,…,n) of principle component in feature space
10. Take key as key value in < key, value > pair ;
11. Compute the value of the sum of different dimensions;
12. Output < key, value > pair;
13. 9. End

C. Reduce-function. The Reduce function takes the result of the combine function as input. In this function, we compute the sum of all samples and compute the number of samples that assigned to same cluster. So the process of reduce function can be described as in Algorithm 4.

Algorithm 4: reduce (key, R)

Input: key (the index of the cluster), R (the list of partial sums from different host)

Output: < key, value > pair

1. Initialize one array A to compute the samples in the list R;
2. Initialize a counter num =0;
3. While(R.hasNext()){//when there is sample in R
4. Compute the number of samples N;
5. A=sum(N);//sum the number of different dimensions and assign to the array A
6. num++;
7. }
8. Dividing the array according to num;
9. Take key as key value in < key, value > pair ;
10. Compute the value of the sum of different dimensions;
11. Output < key, value > pair;
12. 9. End

The dependent variable should multiply with the corresponding element of hidden node vector. However, the intermediate value can be a string and the result of multiplying the dependent variable in the reduce function. MapReduce can work well on computing matrix in parallel, so our method has greatly improved the efficiency of KPCA. To examine the result of our method, we made an experiment, as discussed below.

Experiment

We use the ORL standard face database as experimental datasets in our experiment. The datasets consist of 40 images taken at different times with different change such as pose, angle, scale, expression and eyes. Each person's face is made up of 10 images with 112 x 92 pixels. In our experiment, we choose 5 images per person as training samples, and the other 5 images are selected as test samples. In order to verify the effectiveness of the proposed method, the recognition efficiency of PCA, KPCA and MRKPCA methods in ORL face databases are compared. The experimental condition of one PC in a cluster and measures are shown in table 1.

Table 1: Experimental condition and measures.

Operating System	CentOS 6
CPU	I712700h
Memory	8G
Software	Hadoop 2.2, Java 1.8
measures	speedup, scaleup and size up, cumulative contribution rate

The Evaluation Measure

As we mentioned in the preceding paragraph, we use scaleup, sizeup and speedup to evaluate the performance of our algorithm. The three measurements are explained as follows:

Scaleup: Scaleup measure is defined by formula (21):

$$\text{Scaleup}(data, m) = \frac{T_1}{T_{mm}} \quad (21)$$

Sizeup: Sizeup measure is defined by formula (22):

$$\text{Sizeup}(data, m) = \frac{T_m}{T_1} \quad (22)$$

Speedup: Sizeup measure is defined by formula (23):

$$\text{Speedup} = \frac{T_1}{T_p} \quad (23)$$

Where T_1 is the execution time of on single core, T_m is the execution time for processing m*data. T_p is the execution time of the parallel algorithm with p processors.

In solving practical problems, if there are p principal components altogether, we generally don't need all of it, but only get the first K principal components according to the size of cumulative contribution rate.

Definition 1. In the principal component analysis, we call ratio of variance of the i-th principal component to sum of variance of all principal components as contribution rate, denoted by:

$$\eta_i = \lambda_i / \sum_{i=1}^{p} \lambda_i \quad (24)$$

Where p is the number of principal components, the value of η_i is bigger, then the corresponding principal component will have a more powerful information.

Definition 2. According to Definition 1, we call $\eta_{(k)}$ as cumulative contribution rate of the first k principal components, it is computed by formula (25)

$$\eta_{(k)} = \sum_{i=1}^{k} \lambda_i / \sum_{i=1}^{p} \lambda_i \quad (25)$$

If $\eta_{(k)}$ >= 85%, it means the first k principal components contains all measurement indexes of the information, this not only reduces the number of variables but also facilitates the analysis and research of practical problems.

Result Analysis

From Figure 2 we can see that the scaleup fall shortly as the number of cores increase. Its average scalability is higher than 80% for the datasets. The scaleup performance is stable slowly when the datasets becomes larger, that indicate our method has a good scalability.

Figure 3 shows the good sizeup performance of our algorithm on multiple core. When the number of cores is smaller than 4, the sizeup value differ little. Compared to 2 or 4 cores on the same datasets, the sizeup value of 8 cores and 16 cores decreases significantly. This indicate a good linear performance of our algorithm.

We have made four experiments on the datasets whose size range from 1GB to 8GB. We can see from Figure 4 that the speedup performance does not work well on datasets of 1GB. As shown in Figure 5, the overall computing time decreases gradually before the number of reducers reaching 4, and the degree of parallelism increases with increase of reducers, when the number of reducers is 4, it reaches its minimum.

The higher cumulative contribution rate will get a higher the recognition rate, the relation between cumulative contribution rate of these methods and their recognition rate is shown as Figure 6.

Figure 6 shows that image recognition rate is becoming higher with the increase of cumulative contribution rate in the three methods, but recognition rate of our proposed methods is always higher than the other two methods.

Feature extraction speed ratio is the ratio of time consuming of KPCA and MRKPCA for the same set of test samples, table 2 lists the speed ratio of feature extraction for different images using the two methods. As can be seen from table 2, the number of bases is different for different images, that means the dimension number of subspace is different when the training samples of these images are projected onto these subspace. However, the number of bases is less than the number of training samples, indicating that there is a linear correlation between the samples. Actually, both KPCA and MRKPCA have a good effect of feature extraction.

The time efficiency of KPCA and MRKPCA is also studied in our experiment. Under the same conditions, we test the run time of KPCA and MRKPCA on different cumulative contribution rate, the result is shown in Table 3 and Figure 7. From the feature extraction speed ratio in table 1, we can see that MRKPCA is faster than KPCA. The results of table 2 and figure 7 show that MRKPCA runs less time than KPCA under the same conditions, so the time efficiency of our method is higher than KPCA in image recognition.

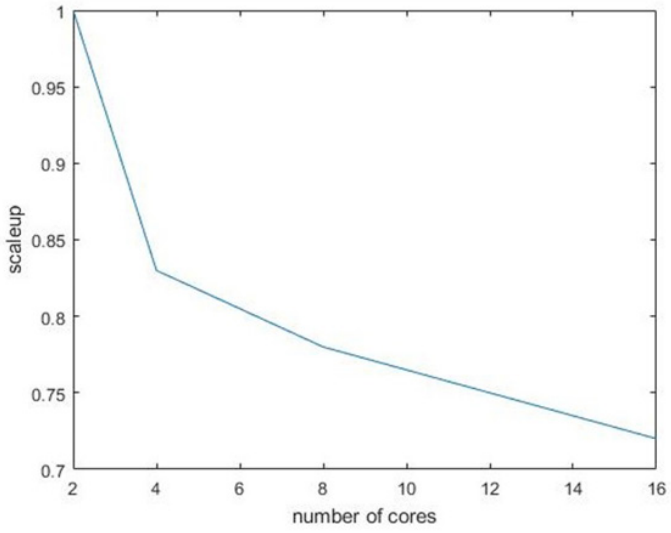

Figure 2: Scaleup of MRKPCA.

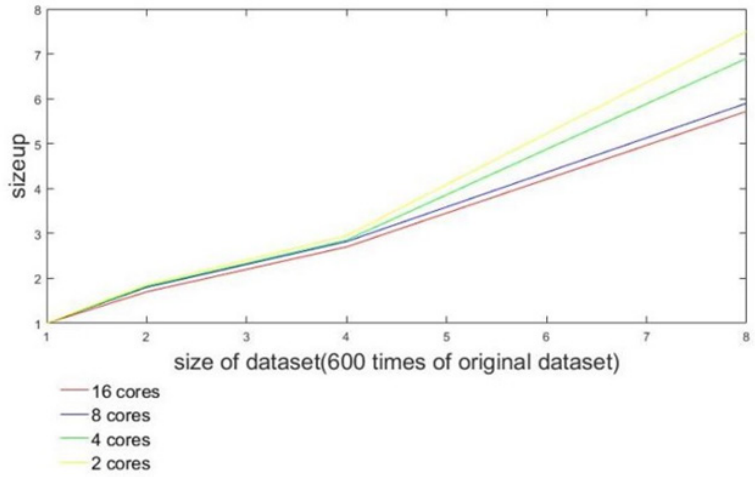

Figure 3: Size up of MRKPCA.

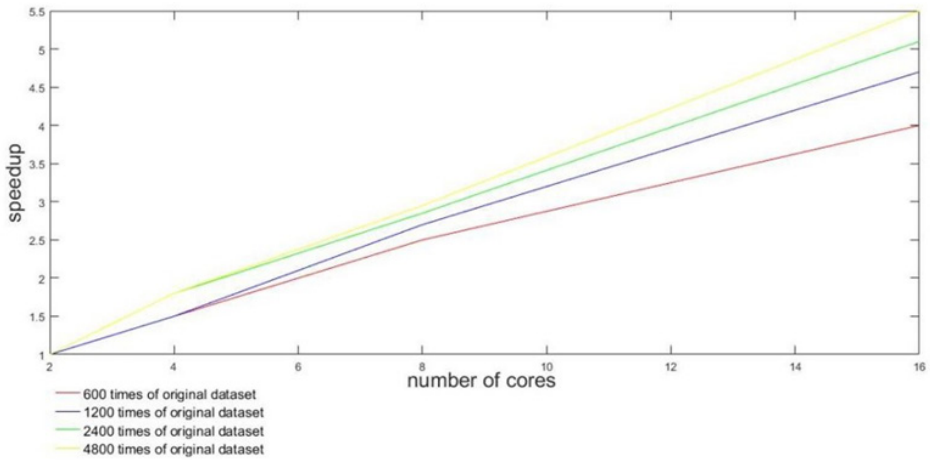

Figure 4: Speedup of MRKPCA.

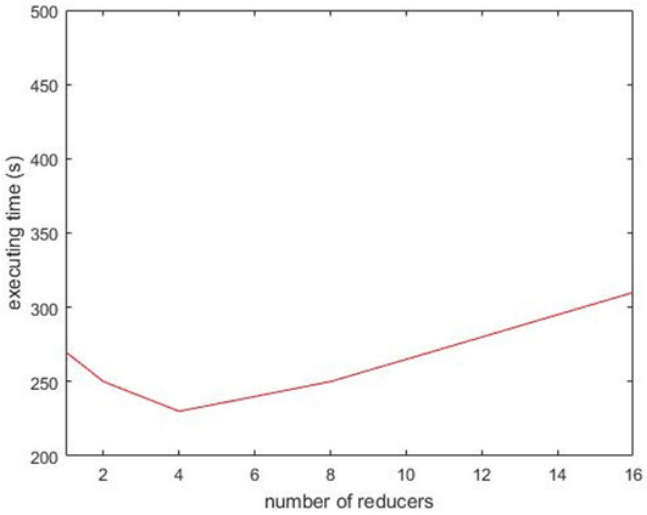

Figure 5: Computing time with different number of reducers.

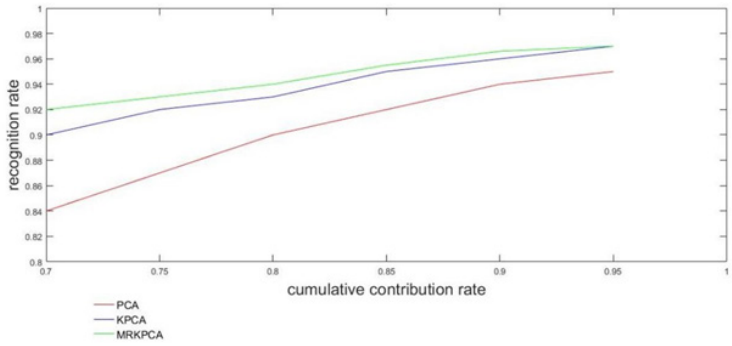

Figure 6: Cumulative contribution rate and image recognition rate in three methods.

Table 2: Feature extraction speed ratio of KPCA and MRKPCA on different images.

Sample image number	1	2	3	4	5
Number of samples	1000	1000	1000	1000	1000
Number of bases	616	984	730	532	742
Feature extraction ratio	7.04	4.6	12.61	14.59	6.51

Table 3: Run time of the two algorithms on different cumulative contribution rate.

cumulative contribution rate	**Algorithm time-consuming (seconds)**	
	KPCA	MRKPCA
70%	158.6	163.5
75%	187.1	168.9
80%	192.6	170.1
85%	193.2	171.2
90%	194.1	172.9
95%	195.2	173.2

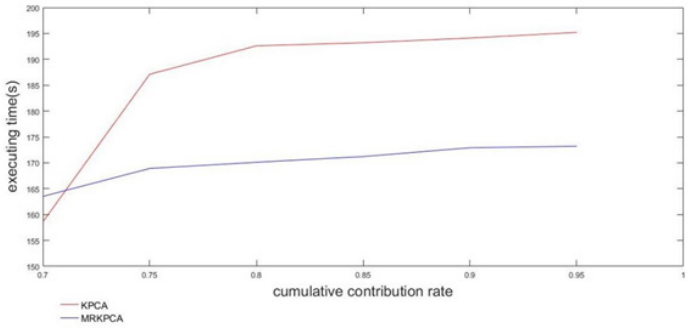

Figure 7: Run time of the two algorithms on different cumulative contribution rate.

Conclusion

To address the computational performance issue of KPCA on large-scale datasets, we proposed a parallel MRKPCA algorithm based on KPCA and MapReduce and implemented it on ORL standard face database. Experimental result show that our proposed MRKPCA algorithm can not only process large scale datasets, but also has a good speedup, scaleup and sizeup performance, and the time efficiency of our method is higher than KPCA. All this give us confidence for research parallel algorithm in the future work.

Acknowledgement

This paper is supported by Doctoral Research Initiation Fund of Shijiazhuang University (No.23BS019).

References

1. BJ Kim (2012) "A Classifier for Big Data" in Proceedings of the 6th International Conference on Convergence and Hybrid Information Technology, Daejeon, Republic of Korea.
2. Wu G, Xu J (2015) Optimized Approach of Feature Selection Based on Information Gain [C]. 2015 International Conference on Computer Science and Mechanical Automation (CSMA) 157-161.
3. Wu G, Wang L, Zhao N, Lin H (2015) Improved Expected Cross Entropy Method for Text Feature Selection[C]. 2015 International Conference on Computer Science and Mechanical Automation (CSMA) 49-54.
4. Tang J and Zhou S (2016) A New Approach for Feature Selection from Microarray Data Based on Mutual Information[J]. IEEE/ACM Transactions on Computational Biology and Bioinformatics 13(6): 1004-1015.
5. Marko S, Nathaniel P, Ee P, Lim JJ (2017) Tweets and Votes: A Study of the 2011 Singapore General Election [C]. International Conference on System Sciences, Singapore 231-243.
6. Pang L (2018) Sentiment Classification using Machine Learning Techniques [C]. Conference on Empirical Methods in Natural Language Processing, Philadelphia 221-231Λ
7. Adam B, Alan FS (2011) Using Twitter to monitor sentiment and predict election results [C]. Proceeding of the Workshop on Sentiment Analysis where Al meets Psychology (SAAIP) Chiang Mai 2-10Λ
8. Bakliwal A, Arora P, Madhappan S, Kapre N, Varma V (2017) Mining sentiments from tweets [C]. Proceeding of 3rd Workshop Computing 11-18.
9. Terrana D, Augello A, Pilato G (2017) Automatic unsupervised polarity detection on a Twitter data stream[C]. International Conference of Semantic Computing, Newport Beach 128-134Λ
10. Saif HY, Alani H (2016) Alleviating data sparsity for Twitter sentiment analysis [C]. Proceeding of CEUΛWorkshop, Urbana 122-129Λ