Research Article

Composite Quantile Regression Neural Network for Massive Datasets

Jun Jin and Zhongxun Zhao

Center of Statistical Research, School of Statistics, Southwestern University of Finance and Economics, Chengdu 661130, China

Correspondence should be addressed to Jun Jin; 118020208006@smail.swufe.edu.cn

Received 16 November 2020; Revised 19 March 2021; Accepted 20 April 2021; Published 4 May 2021

Academic Editor: Mukhtaj khan

Copyright © 2021 Jun Jin and Zhongxun Zhao. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Traditional statistical methods and machine learning on massive datasets are challenging owing to limitations of computer primary memory. Composite quantile regression neural network (CQRNN) is an efficient and robust estimation method. But most of existing computational algorithms cannot solve CQRNN for massive datasets reliably and efficiently. In this end, we propose a divide and conquer CQRNN (DC-CQRNN) method to extend CQRNN on massive datasets. The major idea is to divide the overall dataset into some subsets, applying the CQRNN for data within each subsets, and final results through combining these training results via weighted average. It is obvious that the demand for the amount of primary memory can be significantly reduced through our approach, and at the same time, the computational time is also significantly reduced. The Monte Carlo simulation studies and an environmental dataset application verify and illustrate that our proposed approach performs well for CQRNN on massive datasets. The environmental dataset has millions of observations. The proposed DC-CQRNN method has been implemented by Python on Spark system, and it takes 8 minutes to complete the model training, whereas a full dataset CQRNN takes 5.27 hours to get a result.

1. Introduction

With the development of information technology, mobile Internet, social networks, and e-commerce have greatly expanded the boundaries and applications of the Internet. Terabytes of dataset are becoming more common. For example, the National Aeronautics and Space Administration Earth Observing System Terra and Aqua Satellites monitor the Earth atmosphere, oceans, and land, producing approximately 1.5 TB of environmental data per day. According to Intel's forecast, in 2020, a networked self-driving car will generate 4 TB of data every 8 hours of operation. Massive datasets offer researchers both unprecedented challenges and opportunities. The key challenge is that using conventional computing methods to directly apply machine learning and statistical methods to these massive datasets is prohibitive. First, the calculation time is too long to get results quickly. Second, the data can be too big so that the computer primary memory overflowed. In

order to overcome these challenges, researchers have proposed a divide-and-conquer method [1–3], which may be an effective method to analyze massive datasets.

In this paper, we consider a divide-and-conquer method for massive datasets. Fan et al. [4] analyzed the least squares regression of the linear model on a massive dataset using a divide-and-conquer method. Lin [1] considered a divide-and-conquer method for estimating equations in massive datasets. Chen [2] analyzed the generalized linear models in extraordinarily large data using the divide-and-conquer method. Zhang [5] proposed a divide-and-conquer kernel ridge regression. Schifano [3] extended the divide-and-conquer approach in [3] to online updating for stream data. A block average quantile regression (QR) approach for the massive dataset is proposed in [6] by combining the divide-and-conquer method with QR. Jiang [7] extended the work of [6] to composite quantile regression (CQR) for massive datasets. Recently, Chen et al. [8] studied QR under memory constraint for massive

datasets. Chen [9] considered a divide and conquer approach for quantile regression in big data.

As we all know, QR is more robust than ordinary least squares regression (OLS) when error distribution is heavily skewed. However, the relative efficiency of the QR can be arbitrarily small compared to OLS. To this end, CQR, proposed in [10], is always effective regardless of the error distribution and works much better in efficiency when compared with the OLS. Since then, CQR has been extensively studied in the nonlinear model. Reference [11] utilized local polynomials and CQR to estimating nonparametric model and proposed the local polynomial CQR. Kai [12] studied CQR for varying coefficient partially linear model. Guo [13] proposed CQR for partially linear additive model. Jiang [14] studied two-step CQR for the single index model.

Artificial intelligence (AI) does not require any priori assumptions about the model in dealing with nonlinear problems, so that it is a significant advantage compared to the statistical model such as nonparametric model and varying coefficient partially linear model. There has been a lot of research on combining the nice properties of AI and QR or CQR. For instance, Taylor and Cannon [15, 16] proposed the quantile regression neural network (QRNN) by combining the artificial neural network (ANN) with QR. A support vector quantile regression (SVQR) method is proposed in [17] through combining support vector machine (SVM) and QR. A composite quantile regression neural network (CQRNN) method is studied in [18], which adds ANN structure to CQR. But, when the amount of data is large, it is well known that using conventional computing methods to directly compute CQR and ANN are very slow, so the computation of CQRNN is slower. It is shown that computation is a bottleneck for application of CQRNN on massive datasets.

In this paper, our focus is on CQRNN for massive datasets whose size exceeds the limitations of a single computer primary memory. Fortunately, we are not limited to a single computer. In this end, we consider CQRNN for massive datasets by the divide-and-conquer method on the distributed system. A distributed system is composed of multiple computers (called nodes) that can run independently, and each node uses wire protocols (such as RPC and HTTP) to transfer information to achieve a common goal or task. In the distributed system, the communication cost of data communication between different nodes is usually very expensive. A “master and worker” type of distributed system is considered where workers do not communicate directly with each other, which is shown in Figure 1. The concrete steps are illustrated as follows. Step 1: randomize the initial parameters. Step 2: distribute the initial parameters to each worker. Step 3: each worker trains a subset of the data and sends the training results to the master. Step 4: the master takes the weighted average of the training results of each worker as the approximate global training results. Step 5: when there is more data to be processed, return to Step 2.

Traditional statistical methods and machine learning on massive datasets are challenging owing to limitations of computer primary memory. CQRNN is an efficient and robust estimation method. But, most of existing

FIGURE 1: A “master and worker” type of distributed system.

computational algorithms cannot solve CQRNN for massive datasets reliably and efficiently. In this paper, we propose a DC-CQRNN method to extend CQRNN on massive datasets. The major idea is to divide the overall dataset into some subsets, applying the CQRNN for data within each subsets, and final results through combining these training results via weighted average. The proposed DC-CQRNN method can significantly reduce the computational time and the required amount of primary memory, and the training results will be as effective as analyzing the full data at the same time. For illustration, we used Monte Carlo simulation to compare the performance of DC-CQRNN method with CQRNN [18], QRNN [15, 16], the artificial neural network (ANN) [19], SVM [20], and random forest (RF) [21]. In addition, an environmental dataset application verifies and illustrates that our proposed approach performs well for CQRNN on massive datasets, where the environmental dataset has millions of observations.

The remainder of this paper is organized as follows. In Section 2, we present the DC-CQRNN for massive datasets in detail. In Section 3, we used Monte Carlo simulation to illustrate the finite sample performance of the proposed DC-CQRNN method. A detailed presentation of the environmental dataset analysis is in Section 4. In Section 5, we conclude the paper.

2. Methodology

2.1. Our Motivation. In recent years, China GDP has grown rapidly. This has also intensified the contradiction between national economic development and the environment. At the same time, smog pollution has occurred in parts of North China, Northeast China, and Central China, which had a huge effect on China. Therefore, it is extremely important to accurately report air quality to the public and to prevent smog measures in advance. At present, a large number of air quality monitoring stations are established in many places in China such as Beijing, Chengde, Tangshan, and Tianjin. However, for those areas that do not have air monitoring stations, how to accurately predict air quality and report to the public in a timely manner remains a problem.

To this end, we collected environmental datasets from air quality monitoring stations in different places, with a total of

1,018,562 observations. When using the CQRNN method to dealing with environmental datasets, we found that the data can be too big so that the general computer primary memory overflowed, and the computational time is too long to get results quickly. CQRNN method is basically ineffective in dealing with massive data. At the same time, it is obvious that SVM, ANN, RF, etc., also take a long time to processing massive datasets. In this section, we propose a divide-and-conquer CQRNN (DC-CQRNN) method to extend CQRNN on massive datasets. The same idea can be applied to SVM, ANN, and RF.

2.2. Composite Quantile Regression Neural Network. In the real world, there is usually a nonlinear relationship between predicted $\mathbf{y}$ and predictor $\mathbf{x} = (x_1, x_2, \dots, x_p)'$, which can be seen as a stochastic model as follows:

$$\mathbf{y} = f(x_1, x_2, \dots, x_p) + \epsilon \equiv f(\mathbf{x}; \boldsymbol{\theta}) + \epsilon, \quad (1)$$

where ϵ is the model error and $\boldsymbol{\theta}$ is a vector of unknown parameters. There are many regression techniques to solve the unknown parameters of the model, such as OLS, QR, CQR, and their derived methods. Xu et al. [18] proposed CQRNN by combining the nice properties of ANN and CQR. Given predictor $\mathbf{x}_i$ and predicted y_i , the CQRNN objective aims for minimizing the empirical loss function:

$$\text{ECQRNNL}(\boldsymbol{\theta}) = \frac{1}{Q} \sum_{q=1}^Q \text{EQRNNL}(\tau_q) \quad (2)$$

$$= \frac{1}{NQ} \sum_{q=1}^Q \sum_{i=1}^N \rho_{\tau_q}(y_i - \hat{y}_i(\tau_q)),$$

where $\rho_{\tau_q}(u) = \tau_q u I(u \geq 0) + (\tau_q - 1)u I(u < 0)$, $\tau_q = q/(Q+1)$ for $q = 1, \dots, Q$ (see [10]), and $\hat{y}_i(\tau_q)$ is the conditional quantile of y at τ_q and can be estimated by the following two steps. Firstly, the outcome of the j th hidden-layer node $g_{j,l}(\tau_q)$ is calculated by an activation function $f^{(h)}$ to the inner product among $\mathbf{x}_i$ and the hidden-layer weights $w_{jl}^{(h)}$ plus the hidden layer bias $b_{jl}^{(h)}$.

$$g_{j,l}(\tau_q) = f^{(h)}\left(\sum_{j=1}^p x_{ij} w_{jl}^{(h)} + b_{jl}^{(h)}\right), \quad (3)$$

where $f^{(h)}$ denotes an activation function generally using the hyperbolic tangent, $\mathbf{w}^{(h)} = (w_{11}^{(h)}, w_{12}^{(h)}, \dots, w_{pL}^{(h)})'$ is a weight vector of hidden layer, and $\mathbf{b}^{(h)} = (b_{11}^{(h)}, b_{12}^{(h)}, \dots, b_{pL}^{(h)})^T$ is a bias vector of the hidden layer. Secondly, an estimate of predicted $\hat{y}_i$ is consequently given by

$$\hat{y}_i(\tau_q) = f^{(o)}\left(\sum_{l=1}^L w_l^{(o)} g_{j,l}(\tau_q) + b^{(o)}\right), \quad (4)$$

where $f^{(o)}$ is the output layer activation function, $\mathbf{w}^{(o)} = (w_1^{(o)}, w_2^{(o)}, \dots, w_L^{(o)})'$ is the output later weight vector, and $b^{(o)}$ is the output layer bias. Let $\boldsymbol{\theta} \equiv$

FIGURE 2: Divide-and-conquer composite quantile regression neural network.

$[(\mathbf{w}^{(h)})'; (\mathbf{w}^{(o)})'; (\mathbf{b}^{(h)})'; b^{(o)}]'$ contains all coefficients including weights and biases, to be trained or estimated. The main purpose of CQRNN is to estimate $\boldsymbol{\theta}$ using

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \text{ECQRNNL}(\boldsymbol{\theta}). \quad (5)$$

Remark 1. (1) According to the suggestion of [16], we use the Huber norm to overcome the problem that $\rho_{\tau}(u)$ is not always differentiable. (2) To avoid over fitting the model, we add a penalty term in equation (5) to obtain equation (6).

$$\text{ECQRNNL}(\boldsymbol{\theta}) = \frac{1}{NK} \sum_{k=1}^K \sum_{i=1}^N \rho_{\tau}(y_i - Q_{y_i}(\tau_k)) + \lambda \frac{1}{pL} \|\mathbf{w}^{(h)}\|_2^2, \quad (6)$$

where $\|\cdot\|_2$ is the L_2 -norm and λ is a regularization parameter. According to the suggestion of [18], we choose L and λ through EBIC-like criterion:

$$\text{EBIC}(L, \lambda) = \ln \text{ECQRNNL}(\boldsymbol{\theta}) + \frac{\ln N}{N} \cdot \text{df} + \frac{\ln p}{2N} \cdot \text{df}, \quad (7)$$

where df is the number of selected variables corresponding to λ and L .

2.3. Divide-and-Conquer Composite Quantile Regression Neural Network. When the sample size N is too big, using conventional computing methods to directly solve the optimization problem in (5) is infeasible. Based on the ideas of [1, 2], our method is to divide the overall dataset into some subsets with each containable in the computer primary memory. Then, we implement the CQRNN for data within each subset. Final, we obtain results through combining these training results via weighted average.

In detail, we proposed DC-CQRNN which can be obtained by the following concrete steps:

Step 1: randomize the initial parameters and divide the full dataset into K subsets, so that the k th subset contains n_k observations: $(\mathbf{x}_{k,i}, y_{k,i}), i = 1, \dots, n_k$.

Step 2: distribute the initial parameters to each worker.

Step 3: each worker trains a subset of the full data set; obtains the estimators $\hat{\theta}_k, k = 1, \dots, K$, using the methodology in solving equation (5); and sends $\hat{\theta}_k$ to the master.

Step 4: the master implements the weighted average of $\hat{\theta}_k, k = 1, \dots, K$ to obtain $\hat{\theta}$ as the resulting estimator of θ :

$$\hat{\theta} = \left(\sum_{k=1}^K \mathbf{x}_k^T \mathbf{x}_k \right)^{-1} \sum_{k=1}^K \mathbf{x}_k^T \mathbf{x}_k \hat{\theta}_k, \quad (8)$$

Step 5: when there are more data to be processed, return to Step 2. The detailed process is shown in Figure 2.

Remark 2. Obviously, if the number of K is large, the subsets' data size n_k will be very small, so the correlation among the values of the dataset is destroyed. To this end, we give some restrictions on K to manage correlation. Regularity conditions are as follows:

- (a) The sample size of the k th subset is $n_k = O(N/K), k = 1, \dots, K$ and that $n_{\max}/n_{\min} = O(1)$
- (b) $K = o(n_{\min}^{(1/2)} (\log \log n_{\min})^{(-3/2)})$, where $n_{\max} = \max(n_k)$ and $n_{\min} = \min(n_k)$

3. Numerical Simulations

In this section we exploit Monte Carlo simulation to compare the finite sample performance of the DC-CQRNN method with CQRNN [18], QRNN [15, 16], ANN [19], SVM [20], and RF [21]. The performances of the CQRNN method with different Q are very similar in the simulation of [22]. Thus, we only consider $Q = 4, 9$ as a compromise between the estimation and computational efficiency of the CQRNN method.

3.1. Simulation Data. In order to investigate the performance of DC-CQRNN methods with different structures, we choose the various values of parameters Q and K , namely, $Q = 4, 9$ and $K = 1, 10, 50$. Simultaneously, to illustrate the effectiveness and robustness of our method, we consider three different error distributions for ϵ : standard normal distribution ($N(0, 0.25)$), a student t distribution with three degrees of freedom ($t(3)$), and a Chi-square distribution with three degrees of freedom ($\chi^2(3)$) and two cases for ϵ .

Case 1 (i.i.d): the data generate from

$$\mathbf{y}_i = \sin(2\mathbf{x}_{1i}) + 2 \exp(-16\mathbf{x}_{2i}^2) + 0.5\epsilon_i, \quad i = 1, 2, \dots, N, \quad (9)$$

where $\mathbf{x}_{1i}$ and $\mathbf{x}_{2i}$ are independent and follow $N(0, 1)$.

Case 2 (non-i.i.d): the data are generated from

$$\mathbf{y}_i = (1 - \mathbf{x}_i + 2\mathbf{x}_i^2) \exp(-0.5\mathbf{x}_i^2) + \sigma(\mathbf{x}_i)\epsilon_i, \quad i = 1, 2, \dots, N, \quad (10)$$

where $\mathbf{x}_i \sim U(-4, 4)$ and $\sigma(\mathbf{x}_i) = ((1 + 0.2\mathbf{x}_i)/5)$ guarantees that $\sigma(\mathbf{x}_i)\epsilon_i$ is a non-i.i.d one.

We generated 100,000 samples with the same observations and variables under case 1 and case 2 with three error distributions, respectively, and randomly assigned 50,000 samples as a training dataset (In sample) and the remaining samples as testing datasets (out of sample). In order to test the performance of the competitive model, all of the simulations are run for 100 replicates.

3.2. Prediction Performance. We use the mean absolute error (MAE) and root mean square error (RMSE) to compare the prediction performance of DC-CQRNN, CQRNN, DC-QRNN, QRNN, ANN, RF, and SVM both in sample and out of sample, where

$$\begin{aligned} \text{MAE} &= \frac{1}{N} \sum_{i=1}^N |y_i - \hat{f}(\mathbf{x}_i)|, \\ \text{RMSE} &= \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{f}(\mathbf{x}_i))^2}, \end{aligned} \quad (11)$$

where $\hat{f}(\mathbf{x}_i)$ is an estimator of $f(\mathbf{x}_i)$. We employ EBIC in (7) to select L and λ , and the results are showed in Table 1. To reduce the computational load, we consider all the combinations of $\lambda = 0, 0.05, \dots, 1$ and $L = 1, 2, \dots, 10$. The predicted performance is listed in Tables 2 and 3.

In Tables 1 and 3, the DC-CQRNN₉ and DC-CQRNN₄ denote DC-CQRNN with $Q = 9$ and $Q = 4$, respectively, and the CQRNN₉ and CQRNN₄ denote CQRNN with $Q = 9$ and $Q = 4$, respectively. For QRNN and DC-QRNN, we only report the results at $\tau = 0.5$.

From Tables 2 and 3, we can see that

- (1) All of the case, CQRNN has higher prediction accuracy than QRNN, ANN, RFR, and SVM, especially when $Q = 9$
- (2) The results of DC-CQRNN and CQRNN are very close, indicating that DC-CQRNN method can be used as an effective approximation of CQRNN method under massive datasets

3.3. Running Time Performance. To assess the computational efficiency of the DC-CQRNN method, we record the running time of all method implemented in Python programming language and carried out on a Intel(R) Core(TM) i9-8950HK CPU (2.90 GHz) processors and 16 GB RAM. For fair comparison, we recorded only the average CPU time used by 100 repetitions of each method. The average CPU time of all method is presented in Table 4.

It can be seen from Table 4 that the CPU time of CQRNN is longer than that of QRNN. At the same time, DC-CQRNN

TABLE 1: The optimal values of parameters selected by EBIC for simulation data.

Case	Errors	CQRNN ₉		CQRNN ₄		QRNN		ANN	
		L	λ	L	λ	L	λ	L	λ
1	$N(0, 0.25)$	5	0.2	5	0.2	5	0.2	5	0.2
	$t(3)$	5	0.1	5	0.1	5	0.1	5	0.1
	$\chi^2(3)$	5	0.2	5	0.2	5	0.2	5	0.2
2	$N(0, 0.25)$	5	0.05	5	0.05	5	0.05	5	0.05
	$t(3)$	5	0.1	5	0.1	5	0.1	5	0.1
	$\chi^2(3)$	5	0.05	5	0.05	5	0.05	5	0.05

runs faster than QRNN and CQRNN, and the CPU time of DC-CQRNN decreases as K increases. The main advantage of DC-CQRNN is that each worker in the master and worker type of distributed system is independent, so the CQRNN on each worker can be executed in parallel. As a result, the computing time of DC-CQRNN in massive datasets is greatly reduced compared to CQRNN.

In addition, to compare the computational efficiency of the DC-CQRNN method in both sequential and parallel distributed environment, in Table 5, the computing time of DC-CQRNN and DC-QRNN under sequential distributed environment (SDE) and parallel distributed environment (PDE) is recorded. It can be seen from Table 5 that the computational efficiency of PDE has obvious advantages compared with SDE.

4. Real-World Data Applications

4.1. Environmental Dataset. The research areas of this paper contain Baoding, Beijing, Chengde, Shijiazhuang, Tangshan, Tianjin, Xingtai, and Zhangjiakou in China from January 1, 2015, to July 1, 2019, totally 1,018,561 observations. Our study collected hourly historical monitoring PM2.5 from website—the Ministry of Environmental Protection of the People's Republic of China (<http://datacenter.mep.gov.cn>)—and collected meteorological data from website—National Oceanic and Atmospheric Administration (<https://www.noaa.gov/>). Each sample includes PM2.5 concentration, temperature, pressure, humidity, wind direction, wind speed, and visibility, respectively, measured at the air quality monitoring stations. Details for the dataset are shown in Table 6.

4.2. Normalization. Data normalization is an important step for many machine-learning estimators, particularly when dealing with neural networks. Dataset with a wider range is likely to cause instability when training models. Standardization was used to standardize the features by deducting the mean and scaling the data, with the variance of variable (Z_i) calculated as

$$Z_i = \frac{x_i - \bar{x}}{\delta_x}, \quad (12)$$

where x_i , $\bar{x}$, and δ_x are the sample values, mean, and standard deviation, respectively.

4.3. Empirical Results. The aim of our experiments is to examine the effectiveness of the proposed DC-CQRNN for the spatial prediction of PM2.5 concentration. We also consider models including ANN, QRNN, CQRNN, RF, and SVM. For the PM2.5 concentration monitoring dataset, it is divided into a training set and a testing set according to a ratio of 7 : 3. To get the results in a most objective way, we have repeated 100 times the experiments of training and testing at randomly chosen composition of training and testing data. The final results of training and testing are the average of all trials. For parameters setting in intelligent model, we use the EBIC approach, and Table 7 presents the results of parameters selection, and for machine learning model, we consider cross-validation [23] for their parameter setting.

Setting $K = 1, 10, 50$ and $Q = 4, 9$. For DC-QRNN, we only report the results at $\tau = 0.5$. The results of our experiments measured by RMSE, MAE, and CPU time are reported in Table 8.

For the prediction accuracy of the model, the training results of DC-QRNN and DC-CQRNN are close to the QRNN and CQRNN based on full sample, respectively. The CQRNN is significantly better than the ANN, RF, and SVM. In particular, the CQRNN performs best when $K = 9$ for both in sample and out of sample.

Considering the CPU time of the model, the CPU time of the ANN based on full sample training is higher than RF and SVM. In the DC-QRNN and DC-CQRNN, the training of each piece of data is independent, so parallel calculation can be performed on each piece of data, which significantly reduces the computational time. The experimental results show that the CPU time of DC-QRNN and DC-CQRNN are much lower than QRNN and CQRNN based on full sample, respectively.

The environmental dataset includes 1,018,562 observations. The entire methodology has been implemented by Python on Spark system. By using the proposed DC-CQRNN method with $K = 50$ and $Q = 9$ on the Spark system, it takes 8 minutes to complete the model training, whereas a full sample CQRNN method with $Q = 9$ takes 5.27 hours to get a result.

5. Results and Discussion

Massive datasets offer researchers both unprecedented challenges and opportunities. The key challenge is that using conventional computing methods to directly apply machine learning and statistical methods to these massive datasets is

TABLE 2: Comparison results for case 1.

Samples	Errors	Indices	CQRNN ₉		DC-CQRNN ₉		CQRNN ₄		DC-CQRNN ₄		QQRNN		DC-QQRNN		ANN	RF	SVM
			$K = 1$	$K = 50$	$K = 10$	$K = 50$	$K = 1$	$K = 50$	$K = 10$	$K = 50$	$K = 1$	$K = 50$	$K = 10$	$K = 50$			
In sample	$N(0, 0.25)$	MAE	0.2085	0.2117	0.2117	0.2158	0.3185	0.3283	0.3283	0.3284	0.4182	0.4183	0.4164	0.4183	0.3137	0.3038	0.3568
		(SD)	(0.0007)	(0.0010)	(0.0010)	(0.0019)	(0.0012)	(0.0019)	(0.0019)	(0.0023)	(0.0037)	(0.0037)	(0.0051)	(0.0091)	(0.0159)	(0.0017)	(0.0015)
		RMSE	0.3489	0.3530	0.3530	0.3572	0.4489	0.4509	0.4509	0.4598	0.4999	0.4999	0.5012	0.5052	0.4512	0.4473	0.4607
		(SD)	(0.0021)	(0.0021)	(0.0021)	(0.0011)	(0.0020)	(0.0027)	(0.0027)	(0.0027)	(0.0019)	(0.0019)	(0.0072)	(0.0086)	(0.0115)	(0.0018)	(0.15)
		MAE	0.5583	0.5629	0.5629	0.5755	0.6583	0.6632	0.6632	0.6750	0.7581	0.7581	0.7586	0.7512	0.7585	0.7291	0.7676
		(SD)	(0.0036)	(0.0088)	(0.0088)	(0.0013)	(0.0039)	(0.0042)	(0.0042)	(0.0066)	(0.0068)	(0.0068)	(0.0077)	(0.0078)	(0.0042)	(0.0020)	(0.0033)
	$t(3)$	RMSE	0.8607	0.8642	0.8642	0.8796	0.9607	0.9643	0.9643	0.9737	1.1107	1.1107	1.1110	1.1079	1.0614	1.0437	1.1807
		(SD)	(0.0029)	(0.0031)	(0.0031)	(0.0050)	(0.0024)	(0.0037)	(0.0037)	(0.0052)	(0.0029)	(0.0029)	(0.0057)	(0.0062)	(0.0299)	(0.0137)	(0.0311)
		MAE	0.7222	0.7361	0.7361	0.7392	0.8926	0.8909	0.8909	0.8915	1.0192	1.0192	1.0082	1.0095	1.0263	1.0066	1.0269
		(SD)	(0.0023)	(0.0017)	(0.0017)	(0.0023)	(0.0024)	(0.0025)	(0.0025)	(0.0045)	(0.0022)	(0.0022)	(0.0076)	(0.0085)	(0.0052)	(0.0012)	(0.0031)
		RMSE	0.8634	0.8726	0.8726	0.8975	1.0263	1.0266	1.0266	1.0278	1.2637	1.2637	1.2511	1.2113	1.2234	1.1866	1.2924
		(SD)	(0.0046)	(0.0040)	(0.0040)	(0.0052)	(0.0046)	(0.0059)	(0.0059)	(0.0072)	(0.0043)	(0.0043)	(0.0079)	(0.0098)	(0.0051)	(0.0013)	(0.0047)
Out of sample	$N(0, 0.25)$	MAE	0.2087	0.2114	0.2114	0.2123	0.3283	0.3283	0.3283	0.3285	0.4185	0.4185	0.4198	0.4242	0.3145	0.4651	0.3564
		(SD)	(0.0003)	(0.0026)	(0.0026)	(0.0027)	(0.0009)	(0.0019)	(0.0019)	(0.0028)	(0.0013)	(0.0013)	(0.0048)	(0.0091)	(0.0167)	(0.0019)	(0.0020)
		RMSE	0.3491	0.3526	0.3526	0.3571	0.4498	0.4529	0.4529	0.4613	0.5090	0.5090	0.5133	0.5101	0.4522	0.5498	0.4587
		(SD)	(0.0007)	(0.0027)	(0.0027)	(0.0027)	(0.0007)	(0.0022)	(0.0022)	(0.0021)	(0.0066)	(0.0066)	(0.0075)	(0.0087)	(0.0130)	(0.0041)	(0.0041)
		MAE	0.5576	0.5633	0.5633	0.5803	0.6596	0.6635	0.6635	0.6764	0.7676	0.7676	0.7649	0.7622	0.7582	0.8224	0.7682
		(SD)	(0.0031)	(0.0041)	(0.0041)	(0.0058)	(0.0033)	(0.0037)	(0.0037)	(0.0052)	(0.0033)	(0.0033)	(0.0070)	(0.0087)	(0.0047)	(0.0025)	(0.0027)
	$t(3)$	RMSE	0.8647	0.8685	0.8685	0.8715	0.9646	0.9687	0.9687	0.9880	1.1147	1.1147	1.1129	1.1198	1.1545	1.2852	1.2003
		(SD)	(0.0025)	(0.0044)	(0.0044)	(0.0065)	(0.0046)	(0.0073)	(0.0073)	(0.0095)	(0.0059)	(0.0059)	(0.0075)	(0.0095)	(0.0102)	(0.0148)	(0.0105)
		MAE	0.7336	0.7461	0.7461	0.7531	0.8940	0.8945	0.8945	0.9032	1.0230	1.0230	1.0089	1.0120	1.0276	1.1924	1.0786
		(SD)	(0.0010)	(0.0013)	(0.0013)	(0.0022)	(0.0029)	(0.0038)	(0.0038)	(0.0049)	(0.0032)	(0.0032)	(0.0071)	(0.0077)	(0.0045)	(0.0040)	(0.0031)
		RMSE	0.8502	0.8552	0.8552	0.8611	1.0707	1.0728	1.0728	1.0857	1.2205	1.2205	1.2149	1.2225	1.2289	1.4926	1.3986
		(SD)	(0.0065)	(0.0040)	(0.0040)	(0.0061)	(0.0036)	(0.0078)	(0.0078)	(0.0091)	(0.0043)	(0.0043)	(0.0083)	(0.0101)	(0.0067)	(0.0067)	(0.0069)

TABLE 3: Comparison results for case 2.

Samples	Errors	Indices	CQRNN ₉		DC-CQRNN ₉		CQRNN ₄		DC-CQRNN ₄		QQRNN		DC-QQRNN		ANN	RF	SVM
			$K = 1$	$K = 50$	$K = 10$	$K = 50$	$K = 1$	$K = 50$	$K = 10$	$K = 50$	$K = 1$	$K = 50$	$K = 10$	$K = 50$			
In sample	$N(0, 0.25)$	MAE	0.0300	0.0304	0.0302	0.0302	0.0401	0.0403	0.0403	0.0407	0.0601	0.0605	0.0605	0.0607	0.0603	0.0377	0.0598
		(SD)	(0.0002)	(0.0001)	(0.0003)	(0.0003)	(0.0002)	(0.0002)	(0.0002)	(0.0004)	(0.0002)	(0.0002)	(0.0002)	(0.0005)	(0.0002)	(0.0001)	(0.0002)
		RMSE	0.0452	0.0557	0.0554	0.0554	0.0553	0.0554	0.0554	0.0559	0.0752	0.0754	0.0754	0.0760	0.0753	0.0445	0.0750
		(SD)	(0.0003)	(0.0001)	(0.0004)	(0.0004)	(0.0003)	(0.0003)	(0.0003)	(0.0005)	(0.0003)	(0.0003)	(0.0003)	(0.0006)	(0.0003)	(0.0001)	(0.0003)
		MAE	0.1703	0.1707	0.1707	0.1707	0.2203	0.2208	0.2208	0.2251	0.3203	0.3221	0.3221	0.3263	0.3204	0.2044	0.3201
		(SD)	(0.0011)	(0.0028)	(0.0012)	(0.0012)	(0.0011)	(0.0013)	(0.0013)	(0.0013)	(0.0011)	(0.0016)	(0.0016)	(0.0027)	(0.0014)	(0.0011)	(0.0014)
	$t(3)$	RMSE	0.3327	0.3615	0.3329	0.3329	0.3828	0.3831	0.3831	0.3862	0.4828	0.4809	0.4825	0.4825	0.4833	0.2710	0.4832
		(SD)	(0.0018)	(0.0019)	(0.0019)	(0.0019)	(0.0184)	(0.0184)	(0.0184)	(0.0180)	(0.0185)	(0.0148)	(0.0148)	(0.0134)	(0.0183)	(0.0079)	(0.0183)
		MAE	0.3058	0.3112	0.3061	0.3061	0.3558	0.3562	0.3562	0.3627	0.4558	0.4564	0.4637	0.4637	0.4697	0.2622	0.4558
		(SD)	(0.0018)	(0.0008)	(0.0017)	(0.0017)	(0.0018)	(0.0017)	(0.0017)	(0.0042)	(0.0018)	(0.0019)	(0.0019)	(0.0038)	(0.0020)	(0.0010)	(0.0016)
		RMSE	0.5053	0.5194	0.5026	0.5026	0.5553	0.5526	0.5526	0.5614	0.6554	0.6576	0.6576	0.6631	0.6379	0.3400	0.6554
		(SD)	(0.0033)	(0.0006)	(0.0040)	(0.0040)	(0.0032)	(0.0040)	(0.0040)	(0.0054)	(0.0032)	(0.0047)	(0.0047)	(0.0067)	(0.0030)	(0.0016)	(0.0030)
Out of sample	$N(0, 0.25)$	MAE	0.0301	0.0302	0.0303	0.0303	0.0401	0.0403	0.0403	0.0407	0.0601	0.0603	0.0603	0.0607	0.0602	0.0682	0.0598
		(SD)	(0.0003)	(0.0001)	(0.0002)	(0.0002)	(0.0003)	(0.0002)	(0.0002)	(0.0004)	(0.0002)	(0.0002)	(0.0002)	(0.0004)	(0.0003)	(0.0004)	(0.0004)
		RMSE	0.0453	0.0455	0.0455	0.0455	0.0553	0.0555	0.0555	0.0560	0.0753	0.0754	0.0754	0.0760	0.0753	0.0865	0.0750
		(SD)	(0.0003)	(0.0001)	(0.0003)	(0.0003)	(0.0003)	(0.0002)	(0.0002)	(0.0004)	(0.0003)	(0.0003)	(0.0003)	(0.0005)	(0.0004)	(0.0004)	(0.0004)
		MAE	0.1704	0.1777	0.1708	0.1708	0.2204	0.2209	0.2209	0.2250	0.3204	0.3222	0.3222	0.3263	0.3205	0.3821	0.3202
		(SD)	(0.0009)	(0.0027)	(0.0009)	(0.0009)	(0.0009)	(0.0007)	(0.0007)	(0.0009)	(0.0009)	(0.0016)	(0.0016)	(0.0026)	(0.0010)	(0.0016)	(0.0009)
	$t(3)$	RMSE	0.3244	0.3281	0.3246	0.3246	0.3744	0.3748	0.3748	0.3780	0.4744	0.4833	0.4833	0.4818	0.4739	0.5526	0.4738
		(SD)	(0.0094)	(0.0020)	(0.0094)	(0.0094)	(0.0094)	(0.0093)	(0.0093)	(0.0087)	(0.0094)	(0.0227)	(0.0227)	(0.0106)	(0.0089)	(0.0107)	(0.0088)
		MAE	0.3058	0.3114	0.3061	0.3061	0.3558	0.3561	0.3561	0.3628	0.4559	0.4565	0.4565	0.4639	0.4697	0.5408	0.4553
		(SD)	(0.0015)	(0.0006)	(0.0013)	(0.0013)	(0.0015)	(0.0013)	(0.0013)	(0.0049)	(0.0015)	(0.0020)	(0.0020)	(0.0047)	(0.0024)	(0.0030)	(0.0019)
		RMSE	0.5080	0.5169	0.5053	0.5053	0.5580	0.5553	0.5553	0.5641	0.6581	0.6581	0.6581	0.6635	0.6397	0.7514	0.6569
		(SD)	(0.0032)	(0.0007)	(0.0040)	(0.0040)	(0.0032)	(0.0040)	(0.0040)	(0.0036)	(0.0031)	(0.0052)	(0.0052)	(0.0072)	(0.0038)	(0.0040)	(0.0039)

TABLE 4: Comparison of average CPU time in seconds for 100 repetitions.

Method	K	t (case 1)			t (case 2)		
		N (0, 0.25)	t (3)	χ^2 (3)	N (0, 0.25)	t (3)	χ^2 (3)
CQRNN ₉	K = 1	96.40	83.20	97.60	85.61	88.62	98.83
DC-CQRNN ₉	K = 10	26.12	26.16	29.72	21.58	31.24	33.61
DC-CQRNN ₉	K = 50	8.77	8.49	9.41	6.66	9.22	9.98
CQRNN ₄	K = 1	70.40	54.00	67.40	56.21	63.42	72.23
DC-CQRNN ₄	K = 10	11.60	10.32	13.06	10.22	11.16	13.12
DC-CQRNN ₄	K = 50	6.72	5.41	6.41	6.44	7.31	8.47
QRNN	K = 1	51.20	41.80	55.00	44.42	46.23	53.14
DC-QRNN	K = 10	6.50	4.50	5.40	8.01	11.98	10.57
DC-QRNN	K = 50	3.30	2.80	3.80	5.34	5.14	6.01
ANN	K = 1	36.40	31.60	37.80	40.82	32.41	34.62
RFR	K = 1	222.00	241.20	247.80	150.42	165.87	174.44
SVM	K = 1	129.40	114.23	113.42	133.21	109.17	108.83

TABLE 5: Comparison of average CPU time in seconds for 100 repetitions.

Method	K		t (case 1)			t (case 2)		
			N (0, 0.25)	t (3)	χ^2 (3)	N (0, 0.25)	t (3)	χ^2 (3)
DC-CQRNN ₉	K = 10	PDE	26.12	26.16	29.72	21.58	31.24	33.61
		SDE	251.42	244.58	285.12	196.51	296.12	305.2
DC-CQRNN ₉	K = 50	PDE	8.77	8.49	9.41	6.66	9.22	9.98
		SDE	388.45	382.51	402.22	312.92	396.61	427.32
DC-CQRNN ₄	K = 10	PDE	11.60	10.32	13.06	10.22	11.16	13.12
		SDE	102.25	97.46	116.42	96.55	99.17	119.23
DC-CQRNN ₄	K = 50	PDE	6.72	5.41	6.41	6.44	7.31	8.47
		SDE	314.25	252.40	298.42	304.23	339.58	402.15
DC-QRNN	K = 10	PDE	6.50	4.50	5.40	8.01	11.98	10.57
		SDE	58.26	38.75	47.56	72.58	102.2	92.68
DC-QRNN	K = 50	PDE	3.30	2.80	3.80	5.34	5.14	6.01
		SDE	148.25	115.45	167.38	241.61	232.48	272.15

TABLE 6: Environmental dataset.

Indicators	Unit	Source	Period
PM2.5	$\mu\text{g}/\text{m}^3$	The Ministry of Environmental Protection of the People’s Republic of China	2015.1–2019.7
Temperature	$^{\circ}\text{C}$		
Pressure	hPa		
Wind speed	m/s	National Oceanic and Atmospheric Administration	
Dew-point temperature	$^{\circ}\text{C}$		
Relative humidity	g/m^3		
Visibility	m		

TABLE 7: The optimal values of parameters selected by EBIC.

Layer	Layer type	Neuron count	Regularization value	Activation function
1	Input	6	0	Relu
2	Hidden	50	0.05	
3	Output	1	0	

TABLE 8: Comparison results for real data.

Method	K	$t(s)$	In sample		Out of sample	
			MAE	RMSE	MAE	RMSE
CQRNN ₉	$K = 1$	18984	17.1320	27.6946	19.9073	29.0148
DC-CQRNN ₉	$K = 10$	1802	17.6858	28.6596	21.1274	29.8764
DC-CQRNN ₉	$K = 50$	482	19.7306	28.8940	21.8771	30.5780
CQRNN ₄	$K = 1$	18393	18.3089	28.1705	20.9148	29.2282
DC-CQRNN ₄	$K = 10$	1675	19.8655	28.9747	21.9289	30.8750
DC-CQRNN ₄	$K = 50$	476	20.7147	30.8786	22.5500	32.0981
QRNN	$K = 1$	14896	20.4303	30.1461	22.6495	31.2272
DC-QRNN	$K = 10$	1602	21.9448	31.4210	23.9496	32.4893
DC-QRNN	$K = 50$	391	22.6668	31.6966	24.2192	33.6168
ANN	$K = 1$	14872	22.9533	31.3591	24.8177	33.1205
RF	$K = 1$	9776	22.4034	31.8750	27.5332	36.3443
SVM	$K = 1$	8666	23.9032	32.6256	26.1106	34.5832

prohibitive. In this paper, the Monte Carlo simulation studies and an environmental dataset application verify and illustrate that our proposed approach performs well for CQRNN on massive datasets. Therefore, the DC-CQRNN method is effective and important. Obviously, the larger the value of K , the more efficient the DC-CQRNN method is. But, it should be noted that if the number of K is large, the subsets data size n_k will be very small, so the correlation among the values of the dataset is destroyed. Therefore, as long as the value of K is not very large, it is ok.

6. Conclusion

Using composite quantile regression neural networks to dealing with massive datasets faces two main challenges: first, the calculation time is too long to get results quickly. Second, the data can be too big so that the computer primary memory overflowed. To solve this difficulty, we propose DC-CQRNN by a divide-and-conquer method on a “master and worker” type of distributed system. The proposed DC-CQRNN can significantly reduce the computational time and the required amount of computer primary memory, and the training results are as effective as analyzing the full data simultaneously. The divide-and-conquer ideal also extends to QRNN, ANN, and SVM. In the future, we will try to use the subsampling method to reduce the time-consuming of the neural network training massive data and save the input cost.

Data Availability

The research areas of this paper contains Baoding, Beijing, Chengde, Shijiazhuang, Tangshan, Tianjin, Xingtai, Zhangjiakou in China from January 1, 2015 to July 1, 2019, totally 1,018,561 observations. Our study collected hourly historical monitoring PM_{2.5} from website: the Ministry of Environmental Protection of the People’s Republic of China (<http://datacenter.mep.gov.cn>), and collected meteorological data from website: National Oceanic and Atmospheric Administration (<https://www.noaa.gov/>).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (no. 11471264).

References

- [1] N. Lin and R. Xi, “Aggregated estimating equation estimation,” *Statistics and Its Interface*, vol. 4, no. 1, pp. 73–83, 2011.
- [2] X. Chen and M. Xie, “A split-and-conquer approach for analysis of extraordinarily large data,” *Statistica Sinica*, vol. 24, pp. 1655–1684, 2014.
- [3] E. D. Schifano, J. Wu, C. Wang, J. Yan, and M.-H. Chen, “Online updating of statistical inference in the big data setting,” *Technometrics*, vol. 58, no. 3, pp. 393–403, 2016.
- [4] T.-H. Fan, D. K. J. Lin, and K.-F. Cheng, “Regression analysis for massive datasets,” *Data & Knowledge Engineering*, vol. 61, no. 3, pp. 554–562, 2007.
- [5] Y. Zhang, J. Duchi, and M. Wainwright, “Divide and conquer kernel ridge regression: a distributed algorithm with minimax optimal rates,” *Journal of Machine Learning Research*, vol. 16, pp. 3299–3340, 2015.
- [6] Q. Xu, C. Cai, C. Jiang, F. Sun, and X. Huang, “Block average quantile regression for massive dataset,” *Statistical Papers*, vol. 61, pp. 141–165, 2017.
- [7] R. Jiang, X. Hu, K. Yu, and W. Qian, “Composite quantile regression for massive datasets,” *Statistics*, vol. 52, no. 5, pp. 980–1004, 2018.
- [8] X. Chen, W. Liu, and Y. Zhang, “Quantile regression under memory constraint,” *Annals of Statistics*, vol. 47, pp. 3244–3273, 2019.
- [9] L. Chen and Y. Zhou, “Quantile regression in big data: a divide and conquer based strategy,” *Computational Statistics & Data Analysis*, vol. 144, 2019.
- [10] H. Zou and M. Yuan, “Composite quantile regression and the oracle model selection theory,” *Annals of Statistics*, vol. 36, pp. 1108–1126, 2008.
- [11] B. Kai, R. Li, and H. Zou, “Local composite quantile regression smoothing: an efficient and safe alternative to local polynomial regression,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 72, no. 1, pp. 49–69, 2010.
- [12] B. Kai, R. Li, and H. Zou, “New efficient estimation and variable selection methods for semiparametric varying-coefficient partially linear models,” *Annals of Statistics*, vol. 39, pp. 305–332, 2011.
- [13] J. Guo, M. Tang, M. Tian, and K. Zhu, “Variable selection in high-dimensional partially linear additive models for composite quantile regression,” *Computational Statistics & Data Analysis*, vol. 65, pp. 56–67, 2013.
- [14] R. Jiang, Z.-G. Zhou, W.-M. Qian, and Y. Chen, “Two step composite quantile regression for single-index models,” *Computational Statistics & Data Analysis*, vol. 64, pp. 180–191, 2013.
- [15] J. W. Taylor, “A quantile regression neural network approach to estimating the conditional density of multiperiod returns,” *Journal of Forecasting*, vol. 19, no. 4, pp. 299–311, 2000.
- [16] A. J. Cannon, “Quantile regression neural networks: implementation in R and application to precipitation downscaling,” *Computers & Geosciences*, vol. 37, no. 9, pp. 1277–1284, 2011.

- [17] I. Takeuchi, Q. Le, T. Sears, and A. Smola, "Nonparametric quantile estimation," *Journal of Machine Learning Research*, vol. 12, Article ID 1231C1264, 2006.
- [18] Q. Xu, K. Deng, C. Jiang, F. Sun, and X. Huang, "Composite quantile regression neural network with applications," *Expert Systems with Applications*, vol. 76, pp. 129–139, 2017.
- [19] X. Feng, Q. Li, Y. Zhu, J. Hou, L. Jin, and J. Wang, "Artificial neural networks forecasting of PM_{2.5} pollution using air mass trajectory based geographic model and wavelet transformation," *Atmospheric Environment*, vol. 107, pp. 118–128, 2015.
- [20] X. Li, A. Luo, J. Li, and Y. Li, "Air pollutant concentration forecast based on support vector regression and quantum-behaved particle swarm optimization," *Environmental Modeling & Assessment*, vol. 24, no. 2, pp. 205–222, 2019.
- [21] X. Hu, J. H. Belle, X. Meng et al., "Estimating PM_{2.5} Concentrations in the conterminous United States using the random forest approach," *Environmental Science & Technology*, vol. 51, no. 12, pp. 6936–6944, 2017.
- [22] Y. Tian, Q. Zhu, and M. Tian, "Estimation of linear composite quantile regression using EM algorithm," *Statistics & Probability Letters*, vol. 117, p. 183, 2016.
- [23] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation, and active learning," in *Advances in Neural Information Processing Systems*, pp. 231–238, 1995.