Hindawi
Applied Computational Intelligence and Soft Computing
Volume 2017, Article ID 5134962, 13 pages
https://doi.org/10.1155/2017/5134962

Research Article

Distributed Nonparametric and Semiparametric Regression on SPARK for Big Data Forecasting

Jelena Fiosina and Maksims Fiosins

Clausthal University of Technology, Clausthal-Zellerfeld, Germany

Correspondence should be addressed to Jelena Fiosina; jelena.fiosina@gmail.com

Received 22 July 2016; Accepted 22 November 2016; Published 8 March 2017

Academic Editor: Francesco Carlo Morabito

Forecasting in big datasets is a common but complicated task, which cannot be executed using the well-known parametric linear regression. However, nonparametric and semiparametric methods, which enable forecasting by building nonlinear data models, are computationally intensive and lack sufficient scalability to cope with big datasets to extract successful results in a reasonable time. We present distributed parallel versions of some nonparametric and semiparametric regression models. We used MapReduce paradigm and describe the algorithms in terms of SPARK data structures to parallelize the calculations. The forecasting accuracy of the proposed algorithms is compared with the linear regression model, which is the only forecasting model currently having parallel distributed realization within the SPARK framework to address big data problems. The advantages of the parallelization of the algorithm are also provided. We validate our models conducting various numerical experiments: evaluating the goodness of fit, analyzing how increasing dataset size influences time consumption, and analyzing time consumption by varying the degree of parallelism (number of workers) in the distributed realization.

1. Introduction

The most current methods of data analysis, data mining, and machine learning should deal with big databases. Cloud Computing technologies can be successfully applied to parallelize standard data mining techniques in order to make working with massive amounts of data feasible [1]. For this purpose, standard algorithms should often be redesigned for parallel environment to distribute computations among multiple computation nodes.

One such approach is to use Apache Hadoop, which includes MapReduce for job distribution [2] and distributed file system (HDFS) for data sharing among nodes.

Recently, a new and efficient framework called Apache SPARK [3] was built on top of Hadoop, which allows more efficient execution of distributed jobs and therefore is very promising for big data analysis problems [4]. However, SPARK is currently in the development stage, and the number of standard data analysis libraries is limited.

R software is a popular instrument for data analysts. It provides several possibilities for parallel data processing through the add-on packages [5]. It is possible also to use Hadoop and SPARK inside of R using SPARKR. This is an R package that provides a lightweight front-end to use Apache SPARK within R. SPARKR is still in the developing stage and supports only some features of SPARK but has a big potential for the future of data science [6].

There exist also alternative parallelization approaches, such as Message Passing Interface (MPI) [7]. However, in the present paper, we will concentrate on SPARK because of its speed, simplicity, and scalability [8].

In this study, we consider regression-based forecasting for the case where the data has a nonlinear structure, which is common in real-world datasets. This implies that linear regression cannot make accurate forecasts and, thus, we resort to nonparametric and semiparametric regression methods, which do not require linearity and are more robust to outliers. However, the main disadvantage of these methods is that they are very time-consuming, and therefore the term "big data" for such methods starts much earlier than with parametrical approaches. In the case of big datasets, traditional nonparallel realizations are not capable of processing all the available data. This makes it imperative to adapt to existing techniques and to develop new ones that overcome this disadvantage. The distributed parallel SPARK framework gives us the possibility of addressing this difficulty and increasing the scalability of nonparametric and semiparametric regression methods, allowing us to deal with bigger datasets.

There are some approaches in the current literature to address nonparametric or semiparametric regression models for parallel processing of big datasets [9], for example, using R add-on packages, MPI. Our study examines a novel, fast, parallel, and distributed realization of the algorithms based on the modern version of Apache SPARK, which is a promising tool for the efficient realization of different machine learning and data mining algorithms [3].

The main objective of this study is to enable a parallel distributed version of nonparametric and semiparametric regression models, particularly kernel-density-based and partial linear models to be applied on big data. To realize this, a SPARK MapReduce based algorithm has been developed, which splits the data and performs various algorithm processes in parallel in the map phase and then combines the solutions in the reduce phase to merge the results.

More specifically, the contribution of this study is (i) to design novel distributed parallel kernel density regression and partial linear regression algorithms over the SPARK MapReduce paradigm for big data and (ii) to validate the algorithms, analyzing their accuracy, scalability, and speedup by means of numerical experiments.

The remainder of this paper is organized as follows. Section 2 reviews the traditional regression models to be analyzed. Section 3 reviews the existent distributed computation frameworks for big datasets. In Section 4, we propose parallel versions of kernel-density-based and partial linear regression model algorithms, based on SPARK MapReduce paradigm. In Section 5, we present the experimental setup and in Section 6 we discuss the experimental framework and analysis. Section 7 concludes the paper and discusses future research opportunities.

2. Background: Regression Models

2.1. Linear Multivariate Regression. We start with linear regression, which is the only regression model realized in the current version of SPARK to compare the results of the proposed methods.

Let us first consider the classical multivariate linear regression model $E(Y \mid \mathbf{X}) = \mathbf{X}\boldsymbol{\beta}$ [10, 11]:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad (1)$$

where n is a number of observations, d is the number of factors, $\mathbf{Y}_{n\times1}$ is a vector of dependent variables, $\boldsymbol{\beta}_{d\times1}$ is a vector of unknown parameters, $\boldsymbol{\varepsilon}_{n\times1}$ is a vector of random errors, and $\mathbf{X}_{n\times d}$ is a matrix of explanatory variables. The rows of the matrix $\mathbf{X}$ correspond to observations and the columns correspond to factors. We suppose that $\{\varepsilon_i\}$ are mutually independent and have zero expectation and equal variances.

```
(1) while not converged do
(2)     for all $j \in \overline{0,d}$ do
(3)         $\widehat{\boldsymbol{\beta}}_j \leftarrow \widehat{\boldsymbol{\beta}}_j - \alpha \frac{\partial}{\partial \widehat{\boldsymbol{\beta}}_j} J(\widehat{\boldsymbol{\beta}}; \mathbf{X}, \mathbf{Y})$
(4)     end for
(5) end while
```

Algorithm 1: Stochastic Gradient Descent algorithm.

The well-known least square estimator (LSE) $\widehat{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$ is

$$\widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{Y}. \quad (2)$$

Further, let $(\mathbf{x}_i, y_i)_{i=1}^n$ be the observations sampled from the distribution of $(\mathbf{X}, Y)$. After the estimation of the parameters $\boldsymbol{\beta}$, we can make a forecast for a certain kth (future) time moment as $E(Y_k) = \mathbf{x}_k\widehat{\boldsymbol{\beta}}$, where $\mathbf{x}_k$ is a vector of observed values of explanatory variables for the kth time moment.

For big data, it is a problem to perform the matrix operations in (2). For this purpose, other optimization techniques can be used. One effective option is to use the Stochastic Gradient Descent algorithm [12], which is realized in SPARK. The generalized cost function to be minimized is

$$J(\boldsymbol{\beta}; \mathbf{X}, \mathbf{Y}) = \frac{1}{2n} (\mathbf{X}\boldsymbol{\beta} - \mathbf{Y})^T (\mathbf{X}\boldsymbol{\beta} - \mathbf{Y}). \quad (3)$$

Algorithm 1 presents the Stochastic Gradient Descent algorithm, where α is a learning rate parameter.

Algorithm 1 can be executed iteratively (incrementally) that is easy to parallelize.

2.2. Kernel Density Regression. The first alternative to the linear regression model we want to consider is the kernel density estimation-based regression, which is of big importance in the case when data have a nonlinear structure. This nonparametric approach for estimating a regression curve has four main purposes. First, it provides a versatile method for exploring a general relationship between two variables. Second, it can predict observations without a reference to a fixed parametric model. Third, it provides a tool for finding spurious observations by studying the influence of isolated points. Fourth, it constitutes a flexible method of substitution or interpolation between adjacent X-values for missing values [13].

Let us consider a nonparametric regression model, $m(x) = E(Y \mid X = x)$ [14], with the same Y, X, and ε as for a linear model

$$y = m(x) + \varepsilon. \quad (4)$$

The Nadaraya-Watson kernel estimator [15, 16] of $m(x)$ is

$$\begin{aligned} m_n(x) &= \frac{\sum_{i=1}^n K((x - x_i)/h)\, y_i}{\sum_{i=1}^n K((x - x_i)/h)} \\ &= \frac{\sum_{i=1}^n K_h(x - x_i)\, y_i}{\sum_{i=1}^n K_h(x - x_i)}, \end{aligned} \quad (5)$$

where $K(u)$ is the kernel function of R^d, which is a nonnegative real-valued integrable function satisfying the following two requirements: $\int_{-\infty}^{\infty} K(u)\,du = 1$ and $K(-u) = K(u)$ for all values of u; $h > 0$ is a smoothing parameter called bandwidth, $K_h(u) = (1/h)K(u/h)$. We can see that each value of the historical forecast, y_i, is taken with some weight of the corresponding independent variable value of the same observation, x_i.

In a multidimensional case $E(Y \mid \mathbf{X}) = m(\mathbf{X})$, the kernel function $K_{\mathbf{H}}(\mathbf{x}) = |\mathbf{H}|^{-1}K(\mathbf{H}^{-1}\mathbf{x})$, where $\mathbf{H}$ is a $d\times d$ matrix of smoothing parameters. The choice of the bandwidth matrix $\mathbf{H}$ is the most important factor affecting the accuracy of the estimator, since it controls the orientation and amount of smoothing induced. The simplest choice is $\mathbf{H} = h\mathbf{I}_d$, where h is a unidimensional smoothing parameter and $\mathbf{I}_d$ is $d \times d$ identity matrix. Then, we have the same amount of smoothing applied in all coordinate directions. Another relatively easy to manage choice is to take the bandwidth matrix equal to a diagonal matrix $\mathbf{H} = \operatorname{diag}(h_1, h_2, \dots, h_d)$, which allows for different amounts of smoothing in each of the coordinates. We implemented the latter in our experiments. The multidimensional kernel function, $K(\mathbf{u}) = K(u_1, u_2, \dots, u_d)$, then is easy to present with univariate kernel functions as $K(\mathbf{u}) = K(u_1)\cdot K(u_2)\cdot \dots \cdot K(u_d)$. We used the Gaussian kernel in our experiments. Then, (5) for the multidimensional case can be rewritten as

$$m_n(\mathbf{x}_k) = \frac{\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x}_k - \mathbf{x}_i)\, y_i}{\sum_{i=1}^{n} K_{\mathbf{H}}(\mathbf{x}_k - \mathbf{x}_i)} = \frac{\sum_{i=1}^{n} \prod_{j=1}^{d} K_{\mathbf{H}}(\mathbf{x}_{k,j} - \mathbf{x}_{i,j})\, y_i}{\sum_{i=1}^{n} \prod_{j=1}^{d} K_{\mathbf{H}}(\mathbf{x}_{k,j} - \mathbf{x}_{i,j})}. \quad (6)$$

An important problem in a kernel density estimation is the selection of the appropriate bandwidth, h. It has an influence on the structure of the neighborhood in (5): the bigger the value of h selected, the higher the significant influence that x_i points have on the estimator $m_n(\mathbf{x}_k)$. In the multidimensional case, $\mathbf{H}$ also regulates the balance between factors. The most popular heuristic methods of bandwidth selection are plug-in and resampling methods. However, those heuristics require a substantial amount of computations, which is not possible for big datasets. For our case, we used a well-known rule-of-thumb approach, proposed by Silverman [17], which works for Gaussian kernels. In the case of no correlation between explanatory variables, there is a simple and useful formula for bandwidth selection with Scott's rule [18]: $h_j = n^{-1/(d+4)}\sigma_j$, $j = 1, 2, \dots, d$, where σ_j is a variance of the jth factor.

It is known [14] that the curse of dimensionality is one of the major problems that arises when using nonparametric multivariate regression techniques. For the practitioner, an additional problem is that, for more than two regressors, graphical illustrations or interpretations of the results are hard or even impossible. Truly multivariate regression models are often far too flexible and general for making a detailed inference.

A very suitable property of the kernel function is its additive nature. This property makes the kernel function easy to use for distributed data [13]. Unfortunately, such models, even with current parallelization possibilities, remain very time-consuming.

2.3. *Partial Linear Models.*

Another alternative to the linear regression model is a semiparametric partial linear regression model (PLM). Currently, several efforts have been allocated to developing methods that reduce the complexity of high dimensional regression problems [14]. The models allow easier interpretation of the effect of each variable and may be preferable to a completely nonparametric model. These refer to the reduction of dimensionality and provide an allowance for partly parametric modelling. On the other hand, PLMs are more flexible than the standard linear model because they combine both parametric and nonparametric components. It is assumed that the response variable $\mathbf{Y}$ depends on variable $\mathbf{U}$ in a linear way but is nonlinearly related to other independent variables $\mathbf{T}$ [19]. The resulting models can be grouped together as so-called semiparametric models. PLM of regression consists of two additive components, a linear and a nonparametric part. PLM of regression is given as

$$E(Y \mid \mathbf{U}, \mathbf{T}) = \mathbf{U}\boldsymbol{\beta} + m(\mathbf{T}), \quad (7)$$

where $\boldsymbol{\beta}_{p\times 1}$ is a finite dimensional vector of parameters of a linear regression part and $m(\cdot)$ is a smooth function. Here, we assume the decomposition of the explanatory variables $\mathbf{X}$ into two vectors, $\mathbf{U}$ and $\mathbf{T}$. The vector $\mathbf{U}$ denotes a p-variate random vector which typically covers categorical explanatory variables or variables that are known to influence the index in a linear way. The vector, $\mathbf{T}$, is a q-variate random vector of continuous explanatory variables that is to be modelled in a nonparametric way, so $p + q = d$. Economic theory or intuition should ideally guide the inclusion of the regressors in $\mathbf{U}$ or $\mathbf{T}$, respectively.

An algorithm for the estimation of the PLMs was proposed in [13], which is based on the likelihood estimator and known as generalized Speckman [20] estimator. We reformulated this algorithm (Algorithm 2) in terms of functions and data structures, which can be easily parallelizable in Section 4.

PLM function is the primary function of Algorithm 2, which takes as parameters the training dataset $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$, bandwidth vector $\mathbf{h}$, and a test dataset, $[\mathbf{U}', \mathbf{T}']$. The first step in the estimation is to execute the function, *SmoothMatrix*, to compute a smoother matrix, $\mathbf{S}$, based on the training data of the nonlinear parts $\mathbf{T}$ and $\mathbf{h}$. This helps us to obtain the smoother matrix, which transforms the vector of observations into fitted values. Next, we estimate the linear coefficients of the linear part of the model with the *LinearCoefficients* function. First, we take into account the influence of the nonlinear part on the linear part of the independent variable, $\widetilde{\mathbf{U}} \leftarrow (\mathbf{I} - \mathbf{S})\mathbf{U}$, and on the dependent variable, $\widetilde{\mathbf{Y}} \leftarrow (\mathbf{I} - \mathbf{S})\mathbf{Y}$. Then, we use the ordinary equation (2) or Algorithm 1 to obtain the linear coefficients. With these coefficients, we calculate the linear part of the forecast. Next, we calculate the nonlinear part of the forecast, using Nadaraya-Watson estimator (6). Here, we recalculate a smoother matrix, taking into account the test-set data of the nonlinear part. We take also into account the influence

```
(1)  function SmoothMatrix($\mathbf{T}, \mathbf{T}', \mathbf{h}$)
(2)      $n \leftarrow rows(\mathbf{T}), n' \leftarrow rows(\mathbf{T}')$
(3)      for all $i \in \overline{1,n}$, $j \in \overline{1,n'}$ do
(4)          $\mathbf{W} : W_{i,j} \leftarrow K_H(\mathbf{T}_i - \mathbf{T}'_j)$, $\mathbf{S} : S_{i,j} \leftarrow W_{i,j} / \sum_{i=1}^{n} W_{i,j}$
(5)      end for
(6)      return $\mathbf{S}$
(7)  end function
(8)
(9)  function LinearCoefficients($\mathbf{S}, [\mathbf{Y}, \mathbf{U}, \mathbf{T}]$)
(10)     $\widetilde{\mathbf{U}} \leftarrow (\mathbf{I} - \mathbf{S})\mathbf{U}, \widetilde{\mathbf{Y}} \leftarrow (\mathbf{I} - \mathbf{S})\mathbf{Y}$
(11)     return $\boldsymbol{\beta} \leftarrow (\widetilde{\mathbf{U}}^T \widetilde{\mathbf{U}})^{-1} \widetilde{\mathbf{U}}^T \widetilde{\mathbf{Y}}$
(12) end function
(13)
(14) function KernelPart($\boldsymbol{\beta}, [\mathbf{Y}, \mathbf{U}, \mathbf{T}], \mathbf{h}, \mathbf{T}'$)
(15)     $\mathbf{S} \leftarrow SmoothMatrix(\mathbf{T}, \mathbf{T}', \mathbf{h})$
(16)     return $m(\mathbf{T}') \leftarrow \mathbf{S}(\mathbf{Y} - \mathbf{U}\boldsymbol{\beta})$
(17) end function
(18)
(19) function PLM($[\mathbf{Y}, \mathbf{U}, \mathbf{T}], \mathbf{h}, [\mathbf{U}', \mathbf{T}']$)
(20)     Smooth matrix $\mathbf{S} \leftarrow SmoothMatrix(\mathbf{T}, \mathbf{T}, \mathbf{h})$
(21)     $\boldsymbol{\beta} \leftarrow LinearCoefficients(\mathbf{S}, [\mathbf{Y}, \mathbf{U}, \mathbf{T}])$
(22)     $m(\mathbf{T}') \leftarrow KernelPart(\boldsymbol{\beta}, [\mathbf{Y}, \mathbf{U}, \mathbf{T}], \mathbf{h}, \mathbf{T}')$
(23)     return $\mathbf{Y}' \leftarrow \mathbf{U}'\boldsymbol{\beta} + m(\mathbf{T}')$
(24) end function
```

Algorithm 2: PLM estimation, training set: $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$, test set: $[\mathbf{Y}', \mathbf{U}', \mathbf{T}']$.

of the linear part, $\mathbf{Y} - \mathbf{U}\boldsymbol{\beta}$. Finally, we sum the linear and nonlinear part of the forecast to obtain the final result, $\mathbf{Y}' \leftarrow \mathbf{U}'\boldsymbol{\beta} + m(\mathbf{T}')$.

3. Background: MapReduce, Hadoop, and SPARK

MapReduce [2] is one of the most popular programming models to deal with big data. It was proposed by Google in 2004 and was designed for processing huge amounts of data using a cluster of machines. The MapReduce paradigm is composed of two phases: map and reduce. In general terms, in the map phase, the input dataset is processed in parallel producing some intermediate results. Then, the reduce phase combines these results in some way to form the final output. The most popular implementation of the MapReduce programming model is Apache Hadoop, an open-source framework that allows the processing and management of large datasets in a distributed computing environment. Hadoop works on top of the Hadoop Distributed File System (HDFS), which replicates the data files in many storage nodes, facilitates rapid data transfer rates among nodes, and allows the system to continue operating uninterruptedly in case of a node failure.

Another Apache project that also uses MapReduce as a programming model, but with much richer APIs in Java, Scala, Python, and R, is SPARK [3]. SPARK is intended to enhance, not replace, the Hadoop stack. SPARK is more than a distributed computational framework originally developed in the UC Berkeley AMP Lab for large-scale data processing that improves the efficiency by the use of intensive memory. It also provides several prebuilt components empowering users to implement applications faster and easier. SPARK uses more RAM instead of network and disk I/O and is relatively fast as compared to Hadoop MapReduce.

From an architecture perspective, Apache SPARK is based on two key concepts; Resilient Distributed Datasets (RDDs) and directed acyclic graph (DAG) execution engine. With regard to datasets, SPARK supports two types of RDDs: parallelized collections that are based on existing Scala collections and Hadoop datasets that are created from the files stored by the HDFS. RDDs support two kinds of operations: transformations and actions. Transformations create new datasets from the input (e.g., map or filter operations are transformations), whereas actions return a value after executing calculations on the dataset (e.g., reduce or count operations are actions). The DAG engine helps to eliminate the MapReduce multistage execution model and offers significant performance improvements.

RDDs [21] are distributed memory abstractions that allow programmers to perform in-memory computations on large clusters while retaining the fault tolerance of data flow models like MapReduce. RDDs are motivated by two types of applications that current data flow systems handle inefficiently: iterative algorithms, which are common in graph applications and machine learning, and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude. To achieve fault tolerance efficiently, RDDs provide a highly restricted form of shared memory; they are read-only datasets that can only be constructed through bulk operations on other RDDs. This in-memory processing is a faster process as there is

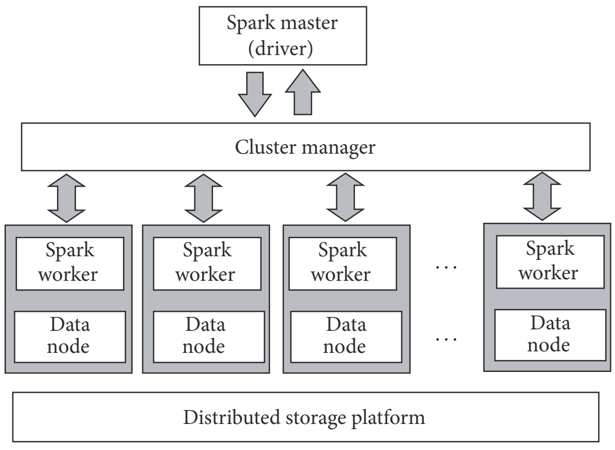

Figure 1: SPARK distributed architecture.

no time spent in moving the data/processes in and out of the disk, whereas MapReduce requires a substantial amount of time to perform these I/O operations, thereby increasing latency.

SPARK uses a master/worker architecture. There is a driver that talks to a single coordinator, called the master, that manages workers in which executors run (see Figure 1).

SPARK uses HDFS and has high-level libraries for stream processing, machine learning, and graph processing, such as MLlib [22]. For this study, linear regression realization included in MLlib [23] was used to evaluate the proposed distributed PLM algorithm.

In this paper, Apache SPARK is used to implement the proposed distributed PML algorithm, as described in Section 4.

4. Distributed Parallel Partial Linear Regression Model

4.1. Distributed Partial Linear Model Estimation. In this section, we continue to discuss kernel-density-based and partial linear models (PLMs), described in Section 2. Next, we consider kernel-density-based regression as a specific case of PLM, when the parametric linear part is empty. The general realization of PLM algorithm allows us to conduct experiments with nonparametric kernel-density-based regression. We discuss how to distribute the computations of Algorithm 2 to increase the speed and scalability of the approach. As we can see from the previous subsection, PLM presupposes several matrix operations, which requires a substantial amount of computational demands [24] and, therefore, is not feasible for application to big datasets. In this section, we focus our attention on (1) how to organize matrix computations in a maximally parallel and effective manner and (2) how to parallelize the overall computation process of the PLM estimation. SPARK provides us with various types of parallel structures. Our first task is to select the appropriate SPARK data structures to facilitate the computation process. Next, we develop algorithms, which execute a PLM estimation (Algorithm 2) using MapReduce and SPARK principles.

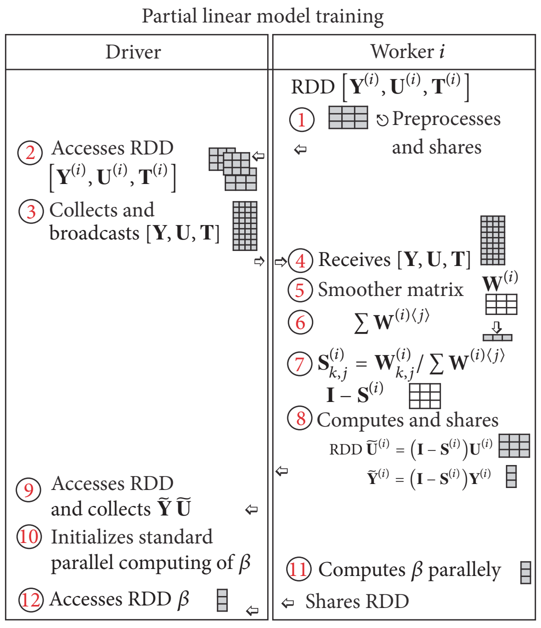
Partial linear model training
Driver
Worker i
RDD $\left[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}\right]$
1 Preprocesses and shares
2 Accesses RDD $\left[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}\right]$
3 Collects and broadcasts $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$
4 Receives $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$
5 Smoother matrix $\mathbf{W}^{(i)}$
6 $\sum \mathbf{W}^{(i)\langle j\rangle}$
7 $\mathbf{S}^{(i)}_{k,j} = \mathbf{W}^{(i)}_{k,j} / \sum \mathbf{W}^{(i)\langle j\rangle}$
$\mathbf{I} - \mathbf{S}^{(i)}$
8 Computes and shares
RDD $\widetilde{\mathbf{U}}^{(i)} = \left(\mathbf{I} - \mathbf{S}^{(i)}\right)\mathbf{U}^{(i)}$
$\widetilde{\mathbf{Y}}^{(i)} = \left(\mathbf{I} - \mathbf{S}^{(i)}\right)\mathbf{Y}^{(i)}$
9 Accesses RDD and collects $\widetilde{\mathbf{Y}}\,\widetilde{\mathbf{U}}$
10 Initializes standard parallel computing of β
11 Computes β parallely
12 Accesses RDD β
Shares RDD

Figure 2: PLM parallel training.

Traditional methods of data analysis, especially based on matrix computations, must be adapted for running on a cluster, as we cannot readily reuse linear algebra algorithms that are available for single-machine situations. A key idea is to distribute operations, separating algorithms into portions that require matrix operations versus vector operations. Since matrices are often quadratically larger than vectors, a reasonable assumption is that vectors are stored in memory on a single machine, while for matrices it is not reasonable or feasible. Instead, the matrices are stored in blocks (rectangular pieces or some rows/columns), and the corresponding algorithms for block matrices are implemented. The most challenging of tasks here is the rebuilding of algorithms from single-core modes of computation to operate on distributed matrices in parallel [24].

Similar to a linear regression forecasting process, PLM forecasting, because of its parametric component, assumes training and estimating steps. Distributed architectures of Hadoop and SPARK assume one driver computer and several worker computers, which perform computations in parallel. We take this architecture into account while developing our algorithms.

Let us describe the training procedure with the purpose of computing the $\boldsymbol{\beta}$ parameter, presented in Figure 2. This part was very computationally intensive and could not be calculated without appropriate parallelization of computations.

First, the driver computer reads the training data, $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$, from a file or database, divides the data into various RDD partitions, and distributes among the workers, $[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}]$. Next, the training process involves the following steps:

(1) Each worker makes an initial preprocessing and transformation of $[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}]$, which include scaling and formatting.

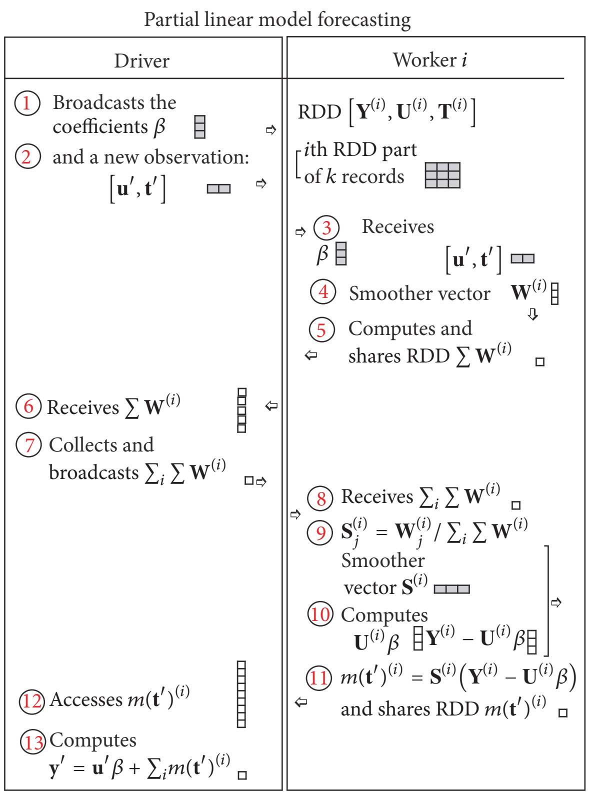
Partial linear model forecasting
Driver
Worker i
1 Broadcasts the coefficients β
2 and a new observation: $[\mathbf{u}', \mathbf{t}']$
RDD $[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}]$
ith RDD part of k records
3 Receives β $[\mathbf{u}', \mathbf{t}']$
4 Smoother vector $\mathbf{W}^{(i)}$
5 Computes and shares RDD $\sum \mathbf{W}^{(i)}$
6 Receives $\sum \mathbf{W}^{(i)}$
7 Collects and broadcasts $\sum_i \sum \mathbf{W}^{(i)}$
8 Receives $\sum_i \sum \mathbf{W}^{(i)}$
9 $\mathbf{S}_j^{(i)} = \mathbf{W}_j^{(i)} / \sum_i \sum \mathbf{W}^{(i)}$ Smoother vector $\mathbf{S}^{(i)}$
10 Computes $\mathbf{U}^{(i)}\beta$ $\mathbf{Y}^{(i)} - \mathbf{U}^{(i)}\beta$
11 $m(\mathbf{t}')^{(i)} = \mathbf{S}^{(i)}(\mathbf{Y}^{(i)} - \mathbf{U}^{(i)}\beta)$ and shares RDD $m(\mathbf{t}')^{(i)}$
12 Accesses $m(\mathbf{t}')^{(i)}$
13 Computes $\mathbf{y}' = \mathbf{u}'\beta + \sum_i m(\mathbf{t}')^{(i)}$

FIGURE 3: PLM parallel forecasting.

(2) The driver program accesses all preprocessed data.

(3) The driver program collects all the preprocessed data and broadcasts them to all the workers.

(4) Then, each worker makes its own copy of the training data $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$.

(5) A smoother matrix, $\mathbf{W}$, is computed in parallel. Since the rows of $\mathbf{W}$ are divided into partitions, each worker then computes its part $\mathbf{W}^{(i)}$.

(6) Each worker computes the sum of elements in each row within its partition $\mathbf{W}^{(i)}$: $\sum_j \mathbf{W}_j^{(i)}$, which is then saved as a RDD.

(7) We do not actually need the normalized matrix $\mathbf{S}$ itself, which could be computed as $\mathbf{W}_j^{(i)} / \sum_j \mathbf{W}_j^{(i)}$, but only $(\mathbf{I} - \mathbf{S})$ as RDD, which is the part directly computed by each worker: $(\mathbf{I} - \mathbf{S}^{(i)})$.

(8) Using the ith part of matrix $(\mathbf{I}-\mathbf{S})$, we multiply it from the right side with the corresponding elements of matrix $\mathbf{U}$. Thus, we obtain the ith part of transformed matrix $\widetilde{\mathbf{U}}^{(i)}$ as a RDD. The ith part of transformed matrix $\widetilde{\mathbf{Y}}^{(i)}$ as RDD is computed analogously.

(9) Finally, the driver program accesses and collects matrices $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{Y}}$.

(10) The driver program initializes computing of the standard linear regression algorithm, *LinearRegressionModelWithSGD*, which is realized in SPARK.

(11) The parameters $\boldsymbol{\beta}$ are computed in parallel.

(12) Then, $\boldsymbol{\beta}$ are accessed by the driver computer.

Let us now describe the forecasting process (Figure 3), which occurs after the parametric part of the PLM has completed its estimation. Note that the kernel part of the model is nonparametric and is directly computed for each forecasted data point.

Since the forecasting is performed after the training process, then the preprocessed training set is already distributed among the workers and is available as a RDD. The PLM distributed forecasting procedure includes the following steps:

(1) The driver program broadcasts the estimated linear part coefficients, $\boldsymbol{\beta}$, to the workers.

(2) Receiving new observation $[\mathbf{u}', \mathbf{t}']$, the driver program broadcasts it to all the workers.

(3) Each worker receives its copy of $[\mathbf{u}', \mathbf{t}']$ and $\boldsymbol{\beta}$.

(4) We compute a smoother matrix $\mathbf{W}$, which is in a vector form. This computation is also partitioned among the workers (see (5)), so the ith worker computes $\mathbf{W}^{(i)}$ as particular columns of $\mathbf{W}$.

(5) Each worker computes partial sums of the rows of the matrix $\mathbf{W}^{(i)}$ elements and shares the partial sums with the driver program.

(6) The driver program accesses the partial sums and performs reducing step to obtain the final sum, $\sum_i \mathbf{W}^{(i)}$.

(7) The driver program broadcasts the final sum to all the workers.

(8) Each worker receives the final sum, $\sum_i \mathbf{W}^{(i)}$.

(9) Each worker computes its columns of the smoother matrix $\mathbf{S}^{(i)}$.

(10) Based on the RDD part $[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}]$ (known from the training step), each worker computes the linear part of the forecast for training data. It was necessary to identify the kernel part of the forecast, performing subtraction of the linear part from the total forecast, the values of which were known from the training set.

(11) Each worker computes the kernel part of the forecast $m(\mathbf{t}')^{(i)}$ as RDD and shares with the driver program.

(12) The driver program accesses the partial kernel forecasts $m(\mathbf{t}')^{(i)}$.

(13) The driver program performs the reducing step to make the final forecast, combining the accessed kernel parts and computing the linear part: $\mathbf{y}' = \mathbf{u}'\boldsymbol{\beta} + \sum_i m(\mathbf{t}')^{(i)}$.

4.2. Case Studies. We illustrate the distributed execution of PLM algorithm with a simple numerical example. First, the driver computer reads the training data, $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$:

$$[\mathbf{Y}, \mathbf{U}, \mathbf{T}] = \left(\begin{array}{c|cc|cc} \mathbf{Y} & \multicolumn{2}{c|}{\mathbf{U}} & \multicolumn{2}{c}{\mathbf{T}} \\ \hline 1 & 1 & 2 & 1 & 2 \\ 2 & 2 & 3 & 2 & 3 \\ 3 & 2 & 1 & 2 & 1 \\ 4 & 2 & 2 & 1 & 1 \end{array}\right). \tag{8}$$

We suppose that the algorithm is being executed in parallel on two workers. The driver accesses $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$ and creates the corresponding RDDs.

First, data preprocessing takes place. This job is shared between the workers in a standard way, and the result is returned to the driver. We do not change our data at this stage for illustrative purposes.

The goal of the training process is to compute the coefficients β of the linear part. First, the matrices $[\mathbf{Y}, \mathbf{U}, \mathbf{T}]$ are divided into the parts by rows:

$$\begin{aligned}[\mathbf{Y}, \mathbf{U}, \mathbf{T}] &= \left(\frac{\mathbf{Y}^{(1)}, \mathbf{U}^{(1)}, \mathbf{T}^{(1)}}{\mathbf{Y}^{(2)}, \mathbf{U}^{(2)}, \mathbf{T}^{(2)}}\right) \\ &= \left(\begin{array}{c|cc|cc} \mathbf{Y}^{(1)} & \multicolumn{2}{c|}{\mathbf{U}^{(1)}} & \multicolumn{2}{c}{\mathbf{T}^{(1)}} \\ 1 & 1 & 2 & 1 & 2 \\ 2 & 2 & 3 & 2 & 3 \\ \hline \mathbf{Y}^{(1)} & \multicolumn{2}{c|}{\mathbf{U}^{(2)}} & \multicolumn{2}{c}{\mathbf{T}^{(2)}} \\ 3 & 2 & 1 & 2 & 1 \\ 4 & 2 & 2 & 1 & 1 \end{array}\right).\end{aligned} \tag{9}$$

Each worker i accesses its part of data $[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}]$, and it has access to the whole data.

Each worker computes its part of the smoother matrix $\mathbf{S}^{(i)}$, in particular:

$$\mathbf{S} = \left(\frac{\mathbf{S}^{(1)}}{\mathbf{S}^{(2)}}\right) = \left(\frac{\mathit{SmoothMatrix}\left(\mathbf{T}^{(1)}, \mathbf{T}\right)}{\mathit{SmoothMatrix}\left(\mathbf{T}^{(2)}, \mathbf{T}\right)}\right). \tag{10}$$

The first worker obtains the elements of smoother matrix $\mathbf{S}^{(1)}$ (it gets directly matrix $\mathbf{I} - \mathbf{S}^{(i)}$); the corresponding elements of matrix $\mathbf{W}^{(1)}$ should be computed first. For example, to get the element $\mathbf{W}^{(1)}_{1,3}$ according to the function *SmoothMatrix*,

$$\begin{aligned}\mathbf{W}^{(1)}_{1,3} &= K_H\left(\mathbf{T}^{(1)}_1 - \mathbf{T}_3\right) = K_H((1,2) - (2,1)) \\ &= K_H((-1,1)) = K((-2,2)) = 0.007.\end{aligned} \tag{11}$$

Then, matrix $\mathbf{W}$ for both workers is the following:

$$\mathbf{W} = \left(\frac{\mathbf{W}^{(1)}}{\mathbf{W}^{(2)}}\right) = \left(\begin{array}{cccc} 0.399 & 0.007 & 0.007 & 0.054 \\ 0.007 & 0.399 & 0.000 & 0.000 \\ \hline 0.007 & 0.000 & 0.399 & 0.054 \\ 0.054 & 0.000 & 0.054 & 0.399 \end{array}\right). \tag{12}$$

To get the element $(\mathbf{I} - \mathbf{S}^{(1)})_{1,3}$ knowing matrix $\mathbf{W}^{(1)}$ according to the function *SmoothMatrix*, we compute

$$\begin{aligned}\left(\mathbf{I} - \mathbf{S}^{(1)}\right)_{1,3} &= -\frac{0.007}{(0.399 + 0.007 + 0.007 + 0.005)} \\ &= -0.016.\end{aligned} \tag{13}$$

Then, matrix $(\mathbf{I} - \mathbf{S})$ for both workers is the following:

$$\begin{aligned}(\mathbf{I} - \mathbf{S}) &= \left(\frac{(\mathbf{I} - \mathbf{S})^{(1)}}{(\mathbf{I} - \mathbf{S})^{(2)}}\right) \\ &= \left(\begin{array}{cccc} 0.147 & -0.016 & -0.016 & -0.115 \\ -0.018 & 0.018 & -0.000 & -0.000 \\ \hline -0.016 & -0.000 & 0.133 & -0.117 \\ -0.106 & -0.000 & -0.107 & 0.213 \end{array}\right).\end{aligned} \tag{14}$$

Then, the corresponding smoothed matrices $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{Y}}$ computed by both workers are

$$\begin{aligned}\widetilde{\mathbf{U}} &= \left(\frac{\widetilde{\mathbf{U}}^{(1)}}{\widetilde{\mathbf{U}}^{(2)}}\right) = \left(\begin{array}{cc} -0.147 & -0.000 \\ 0.018 & 0.019 \\ \hline 0.016 & -0.134 \\ 0.106 & 0.106 \end{array}\right), \\ \widetilde{\mathbf{Y}} &= \left(\frac{\widetilde{\mathbf{Y}}^{(1)}}{\widetilde{\mathbf{Y}}^{(2)}}\right) = \left(\begin{array}{c} -0.393 \\ 0.018 \\ \hline -0.085 \\ 0.426 \end{array}\right).\end{aligned} \tag{15}$$

Matrices $\widetilde{\mathbf{U}}^{(1)}$ and $\widetilde{\mathbf{U}}^{(2)}$ are sent to the driver and collected into the matrix $\widetilde{\mathbf{U}}$, but $\widetilde{\mathbf{Y}}^{(1)}$ and $\widetilde{\mathbf{Y}}^{(2)}$ are collected into the vector $\widetilde{\mathbf{Y}}$.

The driver program calls the standard procedure of regression coefficient calculation, which is shared between workers in a standard way. The resulting coefficients β are collected on the driver:

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} -0.002 \\ 2.753 \\ 1.041 \end{pmatrix}. \tag{16}$$

At the forecasting stage, each worker receives a set of points for forecasting. Now, we illustrate the algorithm for a single point $[\mathbf{u}', \mathbf{t}'] = [(2,1), (1,1)]$ and two workers.

Each worker i accesses its part of training data $[\mathbf{Y}^{(i)}, \mathbf{U}^{(i)}, \mathbf{T}^{(i)}]$ and computes then the elements of matrix $\mathbf{S}^{(i)} = \mathit{SmoothMatrix}(\mathbf{T}^{(i)}, \mathbf{t}')$. First, smoother matrix $\mathbf{W}^{(i)}$ should be computed. For example, worker 1 computes the element

$$\begin{aligned}\mathbf{W}^{(1)}_{1,1} &= K_H((1,2) - (2,1)) = K_H((-1,1)) \\ &= K((-2,2)) = 0.007.\end{aligned} \tag{17}$$

So, workers 1 and 2 obtain matrices

$$\mathbf{W} = \left(\frac{\mathbf{W}^{(1)}}{\mathbf{W}^{(2)}}\right) = \left(\begin{array}{cc} 0.007 & 0.000 \\ \hline 0.399 & 0.054 \end{array}\right). \tag{18}$$

Now, each worker computes partial sums (so for worker 1, $\mathbf{W}_{1,1}^{(1)} + \mathbf{W}_{1,2}^{(1)} = 0.007 + 0.000 = 0.007$; for worker 2, $0.399 + 0.054 = 0.453$) and shares them to the driver program. The driver computes the total sum $0.007 + 0.453 = 0.46$ and sends it to the workers.

Now, each worker computes matrix $\mathbf{S}^{(i)}$. For example, worker 1 computes the element $\mathbf{S}_{1,1}^{(1)} = 0.007/0.46 = 0.016$.

Then, matrices $\mathbf{S}^{(i)}$ for both workers are

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}^{(1)} \\ \hline \mathbf{S}^{(2)} \end{pmatrix} = \begin{pmatrix} 0.016 & 0.000 \\ \hline 0.867 & 0.117 \end{pmatrix}. \tag{19}$$

Now, each worker computes $\mathbf{U}^{(i)}\beta$ and the corresponding difference $\mathbf{Y} - \mathbf{U}^{(i)}\beta$:

$$\mathbf{U}\beta = \begin{pmatrix} \mathbf{U}^{(1)}\beta \\ \hline \mathbf{U}^{(2)}\beta \end{pmatrix} = \begin{pmatrix} 4.83 \\ 8.63 \\ \hline 6.55 \\ 7.59 \end{pmatrix},$$
$$\mathbf{Y} - \mathbf{U}\boldsymbol{\beta} = \begin{pmatrix} \mathbf{Y}^{(1)} - \mathbf{U}^{(1)}\boldsymbol{\beta} \\ \hline \mathbf{Y}^{(2)} - \mathbf{U}^{(2)}\boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} -3.83 \\ -6.63 \\ \hline -3.55 \\ -3.59 \end{pmatrix}. \tag{20}$$

Each worker computes the smoothed value of the kernel part of the forecast:

$$m^{(i)}\left(\mathbf{t}'\right) = \mathbf{S}^{(i)}\left(\mathbf{Y}^{(i)} - \mathbf{U}^{(i)}\boldsymbol{\beta}\right), \tag{21}$$

so $m^{(1)}(\mathbf{t}') = -0.0628$ and $m^{(2)}(\mathbf{t}') = -3.49$. These values are shared with the driver program. It computes their sum, which is considered as a kernel part of the prediction: $m(\mathbf{t}') = -3.56$.

The driver program computes the linear part of the prediction, $\mathbf{u}'\beta = 3.79$, and the final forecast, $\mathbf{y}' = \mathbf{u}'\beta + m(\mathbf{t}') = 0.236$.

In the next sections, we evaluate the performance of the proposed algorithms by various numerical experiments.

5. Experimental Setup

5.1. Performance Evaluation Metrics

5.1.1. Parameters. In our study, we divided the available datasets into two parts: training dataset and test dataset. The training dataset was used to train the model and the test dataset was used to check the accuracy of the results. The sizes of the training and test datasets were the important parameters as they influenced accuracy, scalability, and speed-up metrics.

Another important parameter was the level of parallelism, which we considered in the number of cores used. We varied this parameter between 1 (no parallelism) and 48 cores.

We also considered processing (learning) time as an important parameter to compare the methods. In most of our experiments, we varied one parameter and fixed the remaining parameters.

We conducted three kinds of experiments.

5.1.2. Accuracy Experiments. In the accuracy experiments, we evaluated the goodness of fit that depended on the sizes of training and test datasets. As the accuracy criterion, we used the coefficient of determination R^2, which is usually a quality metric for regression model. It is defined as a relative part of the sum of squares explained by the regression and can be calculated as

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} = 1 - \frac{\sum_i \left(y_i - \widehat{y}_i\right)^2}{\sum_i \left(y_i - \overline{y}\right)^2}, \tag{22}$$

where $\overline{y} = \sum_{i=1}^{n} y_i/n$ and $\widehat{y}_i$ is the estimation (prediction) of y_i calculated by the regression model.

Note that we calculated R^2 using test dataset. Thus, the results were more reliable, because we trained our model on one piece of data (training set) but checked the accuracy of the model on the other one (test set).

5.1.3. Scalability Experiments. Next, we tested the proposed methods on big data. We analyzed how increasing the size of the dataset influences the time consumption of the algorithm. First, we discussed how the execution time changed with the increasing of the size of the dataset. Then, we analyzed how methods accuracy depends on the execution (training) time under different conditions. Scalability was measured with the fixed number of cores.

5.1.4. Speed-Up Experiments. Finally, we analyzed the relationship between time consumption and the degree of parallelism (number of cores) in the distributed realization. We varied the various parallelization degrees and compared the speed of work of various methods. The speed was measured with the fixed accuracy and scale.

5.2. Hardware and Software Used. The experiments were carried out on a cluster of 16 computing nodes. Each one of these computing nodes had the following features: processors: 2x Intel Xeon CPU E5-2620, cores: 4 per node (8 threads), clock speed: 2.00 GHz, cache: 15 MB, network: QDR InfiniBand (40 Gbps), hard drive: 2TB, RAM: 16 GB per node.

Both Hadoop master processes the NameNode and the JobTracker were hosted in the master node. The former controlled the HDFS, coordinating the slave machines by means of their respective DataNode processes, while the latter was in charge of the TaskTrackers of each computing node, which executed the MapReduce framework. We used a standalone SPARK cluster, which followed a similar configuration, as the master process was located on the master node, and the worker processes were executed on the slave machines. Both frameworks shared the underlying HDFS file system.

These are the details of the software used for the experiments: MapReduce implementation: Hadoop version 2.6.0, SPARK version 1.5.2, operating system: CentOS 6.6.

5.3. Datasets. We used three datasets: synthetic data, airlines data, and Hanover traffic data. The main characteristics of these datasets are summarized in Table 1. Table 2 presents sample records for each dataset.

Table 1: Characteristics of the datasets.

Dataset	Number of records	Number of factors
Synthetic data	10,000	2
Traffic data	6,500	7
Airlines delays	120,000,000 (13,000)	29 + 22 (10)

Synthetic Data. We started with synthetic data in order to check how the partially linear regression performs on partially linear data and to compute the basic performance characteristics that depended on training and test set sizes. We took the following model:

$$y = 0.5x^1 + x^2 \sin\left(x^2\right) + \varepsilon, \qquad (23)$$

where x^1 and x^2 were independently uniformly distributed in the interval $[0; 1000]$ and $\varepsilon \sim N(0; 50)$. The dependence on x^2 was clearly nonlinear and strictly fluctuating.

Hanover Traffic Data. In [25], we simulated a traffic network in the southern part of Hanover, Germany, on the base of real traffic data. In this study, we used this simulation data as a dataset. The network contained three parallel and five perpendicular streets, which formed 15 intersections with a flow of approximately 5000 vehicles per hour. For our experiments, 6 variables were used: y travel time (min), x^1 length of the route (km), x^2 average speed in the system (km/h), x^3 average number of stops in the system (units/min), x^4 congestion level of the flow (veh/h), x^5 traffic lights in the route (units), and x^6 left turns in the route (units).

For the Hanover traffic data, we solved the problem of travel time predictions. We did not filter the dataset and instead used the data in the original form. Principally, the analysis of outliers allows deleting suspicious observations to obtain more accurate results; however, some important information can be lost [26].

Taking different subsets of variables, we found that the following variable assignment to linear and kernel parts of PLM is optimal:

(i) Linear part **U**: x^4, x^6

(ii) Kernel part **T**: x^1, x^2, x^3, x^5

Airlines Data. The data consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008 [27]. This is a large dataset: there are nearly 120 million records in total, taking up 12 gigabytes. Every row in the dataset includes 29 variables. This data was used in various studies for analyzing the efficiency of methods proposed for big data processing, including regression [28]. In order to complement this data and to obtain a better prediction, we added weather average daily information, including daily temperatures (min/max), wind speed, snow conditions, and precipitation, which is freely available from the site https://www.wunderground.com/. This provided an additional 22 factors for each of the 731 days. The airlines and weather datasets were joined on the corresponding date.

For this data, we aimed to solve the problem of the departure delay prediction, which corresponded to the DepDelay column of data.

For test purposes, we selected the Salt Lake City International Airport (SLC). Our experiments showed that construction of a model for several airports resulted in poor results; this required special clustering [29] and could be the subject of future research.

Therefore, we selected two years (1995 and 1996) for our analysis. This yielded approximately 170,000 records.

An additional factor, which was added to this data, was the number of flights 30 minutes before and 30 minutes after the specific flight.

The initial analysis showed the heterogeneous structure of this data. Clustering showed that two variables are the most important for separating the data: departure time and travel distance. A very promising cluster with good prediction potential was short late flights (DepTime ≥ 21:45 and Distance ≤ 1000 Km). They could provide relatively good predictions; however, the distribution of delays did not significantly differ from the original dataset. This cluster contained approximately 13,000 records.

In subsequent experiments, we used subsets from this data in order to demonstrate the influence of dataset sizes on the prediction quality.

For the PLM algorithm, the first important issue was how to divide the variables for the linear and kernel parts of the regression. Taking different subsets of variables and including/excluding variables one by one, we found 10 most significant variables and the following optimal subsets:

(i) Linear part **U**: distance of the flight (Distance), average visibility distance (MeanVis), average wind speed (MeanWind), average precipitation (Precipitationmm), wind direction (WindDirDegrees), number of flights in 30-minute interval (Num), and departure time (DepTime)

(ii) Kernel part **T**: day of the week (DayOfWeek), thunderstorm (0 or 1), and destination (Dest)

Table 2 presents data fragments of 10 random records from each training dataset.

6. Experimental Framework and Analysis

6.1. Accuracy Experiments. We compared the goodness-of-fit metric (R^2) of the partially linear, linear, and kernel regression for the datasets by varying the size of the training dataset. Note that the accuracy did not depend on the size of the test set (which was taken equal to 1000). The results are presented in Figures 4, 5, and 6.

We observed very different results. For synthetic data, one variable was linear by definition and another one was nonlinear; the partially linear model was preferable as it produced R^2 with a value of 0.95 against 0.12 for both linear and kernel models. For the airlines dataset, partially linear model was also preferable, but the difference was not so significant. In contrast, for the Hanover data, the kernel model was preferable compared with the partially linear one

Table 2: Fragments of datasets.

(a) Synthetic dataset

y	x^1	x^2
2.31	0.87	1.26
1.45	1.27	−0.36
−0.5	0.47	−0.06
−1.9	−0.94	1.51
−1.51	0.33	1.72
−0.09	−0.1	−1.71
0.17	0.24	1.64
1.8	0.07	0.77
−0.5	0.45	−0.22
0.76	0.94	1.32

(b) Hanover dataset

y: travel time	x^1: length	x^2: speed	x^3: stops	x^4: congestion	x^5: tr. lights	x^6: left turns
256	2107.51	30.30	2.10	42.43	225	0
284	2349.74	22.36	4.89	85.56	289	4
162	1248.51	19.33	9.27	85.91	81	1
448	2346.80	20.58	8.39	86.60	289	1
248	352.67	19.33	9.27	85.91	25	1
327	907.30	23.54	3.96	86.95	100	0
443.5	1093.29	22.01	5.44	88.66	169	0
294	348.35	23.68	3.81	89.33	25	0
125.5	1236.62	18.97	10.65	85.21	81	1
511.5	357.23	19.96	7.66	84.85	25	1

(c) Airlines dataset

DepDelay	DayOfWeek	Distance	MeanVis	MeanWind	Thunderstorm	Precipitationmm	WindDirDegrees	Num	Dest	DepTime
0	3	588	24	14	0	12.7	333	18	SNA	2150
63	7	546	22	13	0	1.78	153	2	GEG	2256
143	5	919	24	18	0	9.14	308	27	MCI	2203
−4	6	599	22	16	0	11.68	161	23	SFO	2147
4	6	368	24	14	0	7.62	151	22	LAS	2159
19	5	188	20	35	0	1.02	170	23	IDA	2204
25	7	291	23	19	0	0	128	22	BOI	2200
1	6	585	17	10	0	6.6	144	25	SJC	2151
0	3	507	28	13	0	9.91	353	24	PHX	2150
38	2	590	28	23	0	0	176	7	LAX	2243

starting from some point. One possible explanation is that the dimension of traffic data was less than airlines data, so the kernel model worked better when it had sufficient data.

6.2. Scalability Experiments. We compared the speed (execution time) of the partially linear, linear, and kernel regression for the datasets by varying the size of training and test datasets. Note that both sizes of training and test datasets influenced the execution time. The results for airlines data are presented in Figures 7 and 8. Other datasets produced similar results.

For the partially linear regression, we could see quadratic relationship between execution time and the size of the training set and a linear dependence of execution time from the size of the test set. These results could be easily interpreted, because the computation of $\widetilde{\mathbf{U}}$ and $\widetilde{\mathbf{Y}}$ (steps (9) and (10) of PLM training, Figure 2) required a smoother matrix of size $n \times n$, where n is the size of the training dataset. On the other hand, the test set participated in the forecasting step only, and the complexity had a linear relationship with its size. For kernel and linear regressions, the relationship with execution time was linear for both training and test set sizes.

Next, we demonstrated how much resources (execution time) should be spent to reach some quality level (R^2) for the partially linear regression model. The results for synthetic data are presented in Figure 9. This graph was constructed

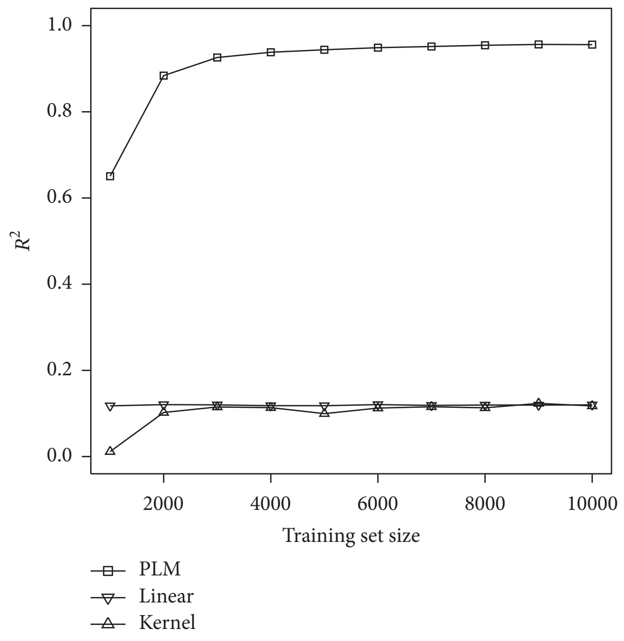

Figure 4: Forecast quality dependence on training set size for synthetic data.

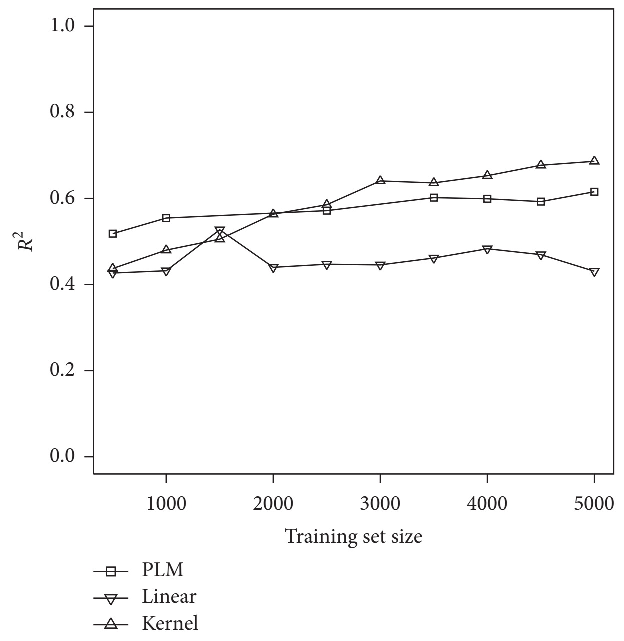

Figure 5: Forecast quality dependence on training set size for traffic data.

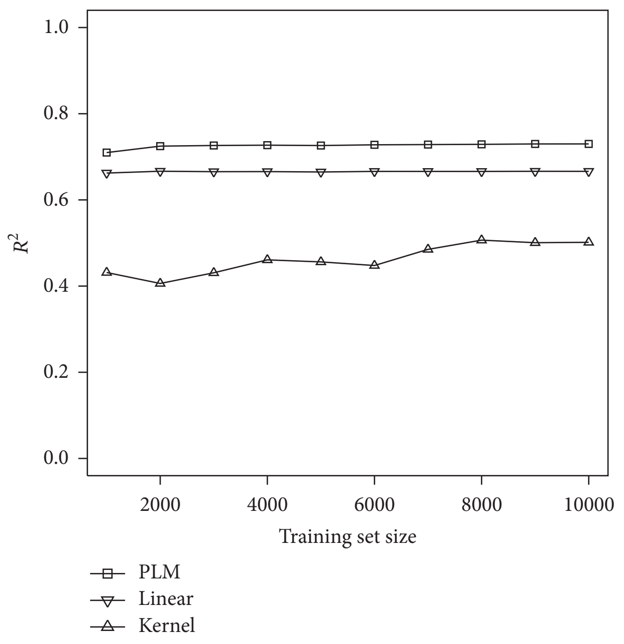

Figure 6: Forecast quality dependence on training set size for airlines data.

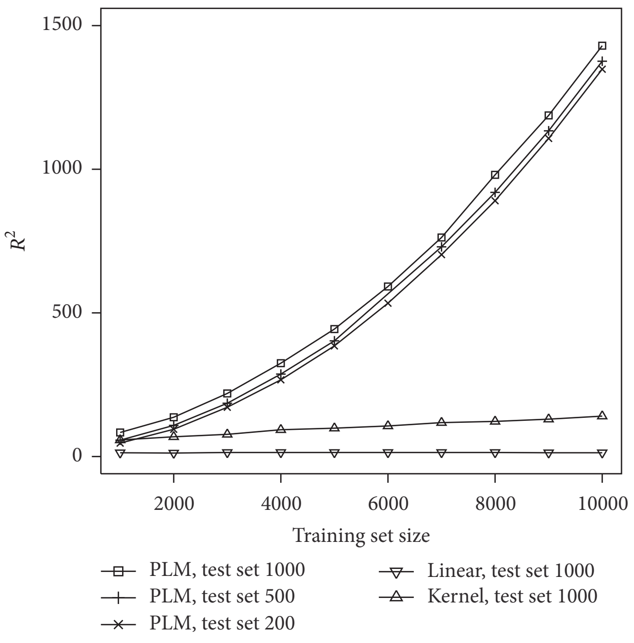

Figure 7: Execution time dependence on training set size for airlines data.

by fixing the test set size to 900, executing the algorithm for different training set sizes, obtaining the corresponding execution time and accuracy, and plotting them on the graph.

We could see relatively fast growth at the beginning, but it slowed towards the end, and successive execution time investments did not increase the goodness of fit, R^2.

6.3. Speed-Up Experiments. Finally, we examined how the execution time changes with the number of available cores. The results for the Hanover traffic data are presented in Figure 10 and the remaining datasets produced similar results.

We could see that the execution time decreased until it reached a threshold of 5–10 cores and then slightly increased (this is true for the PLM and the kernel regression; for the linear regression, the minimum was reached with 2 cores). This is explained by the fact that data transfer among a large number of cores takes significantly more time than computations. This meant that SPARK still needed a better

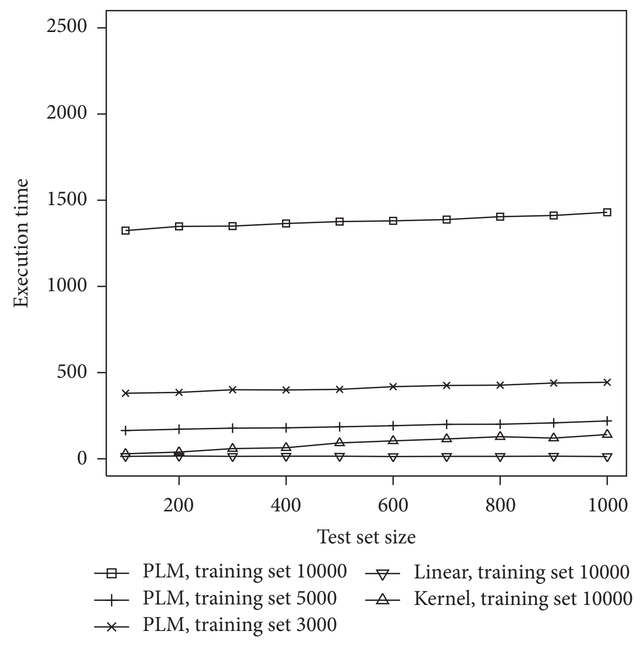

Figure 8: Execution time dependence on test set size for airlines data.

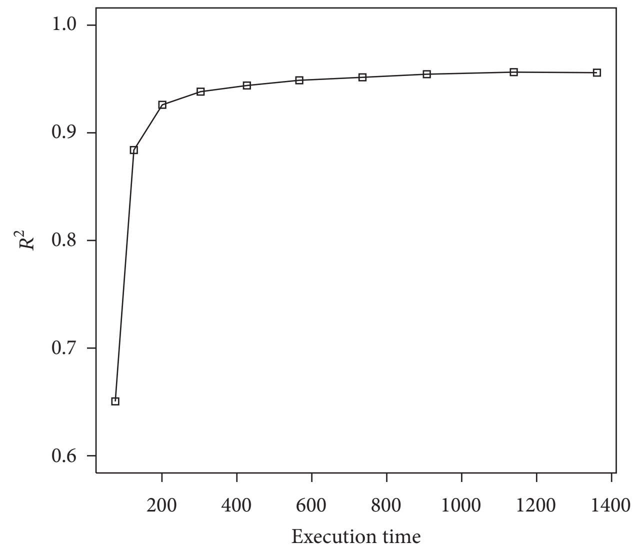

Figure 9: Forecast quality dependence on execution time of PLM algorithm for synthetic data.

optimization mechanism for parallel task execution. A similar issue has been reported by other researchers in [30] for clustering techniques.

7. Concluding Remarks

This paper presents distributed parallel versions of kernel-density-based and partially linear regression models, intended to process big datasets in a reasonable time. The algorithms deploy Apache SPARK, which is a well-known distributed computation framework. The algorithms were tested over three different datasets of approximately 10,000

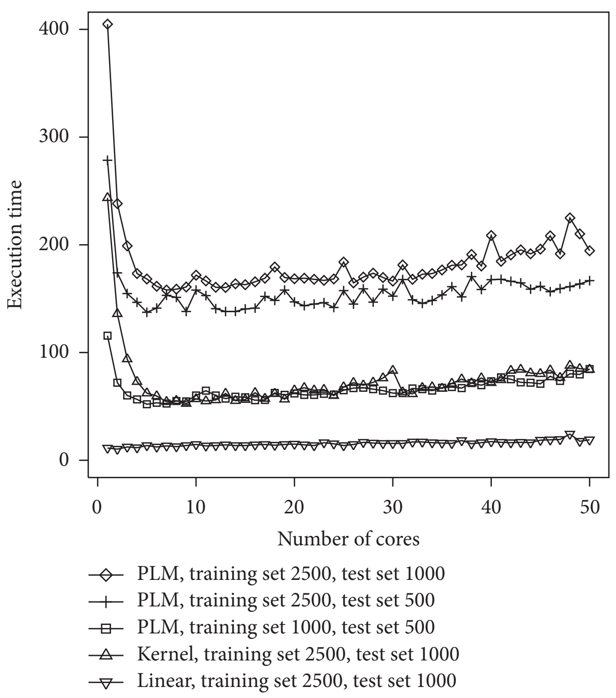

Figure 10: PLM algorithm execution time depending on the number of processing cores for traffic data.

records, which cannot be processed by a single computer in a reasonable time. Thus, it was tested over a cluster of 16 nodes. We conducted three kinds of experiments. First, in the *Accuracy Experiments,* the results indicated that they could not be well forecasted using the linear regression model. We conducted the experiments with a more common situation, when the structure of the data is not linear or partially linear. For all the datasets, (non)semiparametric models (kernel and PLM) showed better results, taking as an efficiency criterion coefficient of determination, R^2. As discussed, kernel regression experiences problem with increasing the dimensionality, because it is difficult to find the points in the neighborhood of the specified point in big dimensions. In our experiments, benchmark data and real-world data had many variables and we showed that semiparametric models gave more accurate results. We also conducted experiments by increasing the training set to show that it resulted in increased accuracy. Next, in the *Scalability Experiments*, we changed the sizes of training and test sets with the aim of analyzing the algorithms' computation time with the same fixed level of parallelism (number of working nodes/cores). All the experiments showed that the training set influenced the time nonlinearly (at most quadratically), but the test set influenced the time linearly. Finally, in the *Speed-Up Experiments*, our purpose was to show the importance of parallel realization of the algorithms to work with big datasets, taking into account the fact that nonparametric and semiparametric estimation methods are very computationally intensive. We demonstrated the feasibility of processing datasets of varying sizes that were otherwise not feasible to process with a single machine. An interesting aspect was that for each combination (dataset, algorithm) we could find the optimal amount of resources (number of cores) to minimize the algorithms execution time. For example, for the PLM regression of the airlines data, with the training set size equal to 2500 and test set size equal to 500, the optimal number of cores was 5 and with the same training set size and the test set size equal to 1000 the optimal number of cores was 9. After this optimal point, the execution time of the algorithm starts to increase. We could explain this phenomenon, as the distribution expenses in this case were more than the award from the parallel execution. Thus, we could conclude that it is important to find the optimal amount of the resources for each experiment.

Competing Interests

The authors declare that there are no competing interests regarding the publication of this paper.

References

[1] C. Long, Ed., *Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data*, John Wiley and Sons, Inc, New York, NY, USA, 2015.

[2] J. Dean and S. Ghemawat, "Map reduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[3] Spark. spark cluster computing framework, http://spark.apache.org/.

[4] D. Peralta, S. del Río, S. Ramírez-Gallego, I. Triguero, J. M. Benitez, and F. Herrera, "Evolutionary feature selection for big data classification: a MapReduce approach," *Mathematical Problems in Engineering*, vol. 2015, Article ID 246139, 11 pages, 2015.

[5] "CRAN task view: High-performance and parallel computing with R," https://cran.r-project.org/web/views/HighPerformanceComputing.html.

[6] SparkR(Ronspark), https://spark.apache.org/docs/latest/sparkr.html.

[7] P. D. Michailidis and K. G. Margaritis, "Accelerating kernel density estimation on the GPU using the CUDA framework," *Applied Mathematical Sciences*, vol. 7, no. 29-32, pp. 1447–1476, 2013.

[8] A. Fernández, S. del Río, V. López et al., "Big data with cloud computing: an insight on the computing environment, mapreduce, and programming frameworks," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 5, pp. 380–409, 2014.

[9] N. E. Helwig, "Semiparametric regression of big data in R," in *Proceedings of the CSE Big Data Workshop*, May 2014.

[10] N. Draper and H. Smith, *Applied Regression Analysis*, John Wiley & Sons, New York, NY, USA, 1986.

[11] M. S. Srivastava, *Methods of multivariate statistics*, Wiley Series in Probability and Statistics, Wiley-Interscience [John Wiley & Sons], NY, USA, 2002.

[12] L. Bottou, "Stochastic gradient descent tricks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7700, pp. 421–436, 2012.

[13] W. Härdle, *Applied Nonparametric Regression*, Cambridge University Press, Cambridge, UK, 2002.

[14] W. Härdle, M. Müller, S. Sperlich, and A. Werwatz, *Nonparametric and Semiparametric Models*, Springer Series in Statistics, Springer-Verlag, New York, NY, USA, 2004.

[15] E. Nadaraya, "On estimating regression," *Theory of Probability and its Applications*, vol. 9, no. 1, pp. 141–142, 1964.

[16] G. S. Watson, "Smooth regression analysis," *The Indian Journal of Statistics. Series A*, vol. 26, pp. 359–372, 1964.

[17] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman & Hall, 1986.

[18] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, Wiley-Interscience, 1992.

[19] H. Liang, "Estimation in partially linear models and numerical comparisons," *Computational Statistics & Data Analysis*, vol. 50, no. 3, pp. 675–687, 2006.

[20] P. Speckman, "Kernel smoothing in partial linear models," *Journal of the Royal Statistical Society B*, vol. 50, no. 3, pp. 413–436, 1988.

[21] M. Zaharia, M. Chowdhury, T. Das et al., "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI '12)*, USENIX Association, Berkeley, Calif, USA, 2012.

[22] Mllib guide, http://spark.apache.org/docs/latest/mllib-guide.html.

[23] N. Pentreath, *Machine Learning with Spark*, Packt Publishing Ltd., Birmingham, UK, 2015.

[24] R. B. Zadeh, X. Meng, A. Ulanov et al., "Matrix computations and optimisation in apache spark," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, 38, 31 pages, San Francisco, Calif, USA, August 2016.

[25] M. Fiosins, J. Fiosina, J. P. Müller, and J. Görmer, "Agent-based integrated decision making for autonomous vehicles in Urban traffic," *Advances in Intelligent and Soft Computing*, vol. 88, pp. 173–178, 2011.

[26] J. Fiosina and M. Fiosins, "Cooperative regression-based forecasting in distributed traffic networks," in *Distributed Network Intelligence, Security and Applications*, Q. A. Memon, Ed., chapter 1, p. 337, CRC Press, Taylor & Francis Group, 2013.

[27] Airline on-time performance data, asa sections on statistical computing statistical graphics, http://stat-computing.org/dataexpo/2009/.

[28] Data science with hadoop: Predicting airline delays part 2, http://de.hortonworks.com/blog/data-science-hadoop-spark-scala-part-2/.

[29] J. Fiosina, M. Fiosins, and J. P. Müller, "Big data processing and mining for next generation intelligent transportation systems," *Jurnal Teknologi*, vol. 63, no. 3, 2013.

[30] J. Dromard, G. Roudire, and P. Owezarski, "Unsupervised network anomaly detection in real-time on big data," in *Communications in Computer and Information Science: ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8–11, 2015. Proceedings*, vol. 539 of *Communications in Computer and Information Science*, pp. 197–206, Springer, Berlin, Germany, 2015.