Parallel Implementation of Multiple Linear Regression Algorithm Based on MapReduce

Moufida Rehab Adjout
Laboratory LIPN-UMR 7030-CNRS,
Paris 13University
Novedia (Groupe VISEO)
Av. J. B. Clement 93430 Villetaneuse, France
Moufida.adjout@gmail.com

Faouzi Boufares
Laboratory LIPN-UMR 7030-CNRS,
Paris 13University
Av. J. B. Clement 93430 Villetaneuse, France
boufares @ lipn.univ-paris13.fr

***Abstract*— The amount of data generated by traditional business activities, creating data repositories ranging from terabytes to petabytes in size. However, this information cannot be practically analyzed on a single commodity computer because the data is too large to fit in memory. For this purpose, the large size of data to be processed requires the use of high-performance analytical systems running on distributed environments. Because the data is so big it affects the types of algorithms we are willing to consider. Then standard analytics algorithms need to be adapted to take advantage of cloud computing models which provide scalability and flexibility.This paper introduces a new distributed training method, which combines the widely used framework for bigdata, MapReduce, with the traditional structure of multiple linear regression.Parallel processing of multiple linear regression will be based on the QR decomposition and the ordinary least squares method adapted to MapReduce. Our platform is deployed on Cloud Amazon EMR service.Experimental results demonstrate that the our parallel version of the multiple linear regression can efficiently handle very large datasets on commodity hardware with a good performance on different evaluation criterions, including number, size and structure of machines in the cluster.**

Keywords: Data mining, Predictive analysis, Multiple linear regression, Big Data, Hadoop, MapReduce, Cloud Computing

I. INTRODUCTION

Processing large volumes of data is at the heart of the emerging issues in research. This issue is especially important for learning supervised systems. They must build their models on large datasets collected automatically with high speed which requires a large capacity evolutionary computation. Thus, analyzing big data becomes quite popular and necessary in many knowledge discovery related areas [2].

On the other side, statistical software provides rich functionality for data analysis and modeling, but can handle only limited amounts of data [16].

The continued collection of such data is meaningful only if the analysis and the decision can be made at short notice. The objective is no longer only to understand phenomena through the data, but to optimize the treatment to get results in the time window where the analyzed information remains relevant. Although many supervised learning algorithms seem to be effective [8] (For example: the multiple linear regression, neural networks, naive Bayesian), which offer rich functionality for data analysis, but on limited amounts of data and operate entirely in main memory [2].Thus, the massive data require two challenges in supervised learning. First, the massive dataset commit memory and computational load for the most complex learning systems. It is therefore impossible to load this mass of data in memory. Second, the response time becomes unacceptable for analysis and learning of this volume of data due to, for example, competitive factors. Thus, the adaptation of these algorithms is essential to meet these challenges [8] [12]. In this work, our contribution is to show that the adaptation of classical learning algorithms of data generally and predictive algorithms practically is possible to provide a response to the phenomenon of BIG DATA.

predictive algorithms is generally serially trained by one machine. But massive training data makes the process slow, costing too much system resources. For these problems, one effective solution is to use the MapReduce framework to do the distributed training.

We focused particularly on the adaptation of the multiple linear regression with massive data processing. Multiple linear regression, is the most powerful and mathematically mature method in data mining [1]. Regression analysis is not only a statistical process for estimating the relationships between variables, but it is also widely used for prediction and forecasting.

Its principle treatment operates on a single server and entirely in main memory. It simply fails when the data becomes too large. Unfortunately, this means that data analysts are unable to work with these packages on massive datasets. Thus, the transition to the scalability of the algorithm is indispensable. Multiple linear regression proves

unsuited to the scalability of the data processed [4]. The use of parallel distributed computing with MapReduce paradigm seems like a natural solution to this problem.

Recently the MapReduce technique has gained popularity in the scientific community because of its effectiveness in parallel processing. MapReduce is a programming model that enables distributed algorithms in parallel on clusters of machines with varied [5] features. The structure of the paper is organized as follows: the next section presents the basic concepts related to multiple linear regression. Section 3 shows experimental limits of the multiple linear regression. We will present in section 4 subsequently our scalable approach, based on the QR decomposition and the ordinary least squares method in an environment MapReduce. Section 5 shows experimental results and evaluations. Finally, the conclusions and future work are presented in Section 6.

II. MULTIPLE LINEAR REGRESSION

A. Definition of multiple linear regression

"Multiple linear regression [1] is a statistical model used to describe a linear relationship between a dependent variable called "explain" or "endogenous", and a set of independent or predictor variables called "explanatory" or "exogenous" reflecting observable phenomena. It is possible to estimate this relationship statistically, from a series of observations.
The simplest form of regression, linear regression, uses the formula of a straight line ($y_i = \beta_i x_i + \varepsilon$) and determines the appropriate value for β and ε to predict the value of y based on the inputs parameters, x".

B. General form of the multiple linear regression

This equation specifies how the dependent variable "y_k" is connected to the explanatory variables x_{ki}:

$$y_k = \beta_0 + \beta_1 x_{k1} + \beta_2 x_{k2} + \ldots + \beta_n x_{kn} = \varepsilon, k := 1, \ldots, m$$

Where m: number of observations and n: number of variables, Where:

- ✓ y_k (k = 1,2, .., m) is the dependent variable;
- ✓ x_{ki} (i = 1,2, .., n) are the independent variables measured without error (not random);
- ✓ β_0, β_1, β_2, .., β_n are the unknown parameters of the model;

A primary goal of a regression analysis is to estimate the relationship between the predictor and the target variables or equivalently, to estimate the unknown parameter β

C. Estimation of model parameters

1) The model

The matrix formulation of the multiple linear regression is described as [3]:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nm} \end{pmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}$$

We note:

$$\boldsymbol{y = \beta X} \quad \text{Where} \quad : \quad \boldsymbol{\beta = (X^T X)^{-1} X^T y} \qquad (\mathbf{1})$$

Where **m:** number of observations, **n:** number of variables, $\mathbf{X^T}$: Transposed matrix **X.**

D. QR decomposition

QR decomposition [2] (also called QR factorization) is one of the most common decomposition matrices in scientific computing to solve the problems of ordinary least squares.
The QR decomposition of a matrix X is a decomposition of the latter into an orthogonal matrix Q and an upper triangular matrix R. QR decomposition is a decomposition of X such that:

$$X = QR$$

With the orthogonal matrix Q (such as Q^T Q = I). If X is invertible, then this factorization is unique [6].

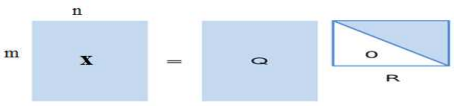


Fig. 1: The QR decomposition

1) Model estimation

To determine the model coefficients, the method is to simplify the calculation by decomposing the data matrix X into two matrices Q and R obtained with the QR decomposition, and thus:

In $\boldsymbol{\beta = (X^T X)^{-1} X^T y}$ $(\mathbf{1})$, Replacing **X** by **QR**:

$$\boldsymbol{\beta = (R^T Q^T Q R)^{-1} R^T Q^T Y =}$$
$$\boldsymbol{(R^T R)^{-1} R^T Q^T Y = R^{-1} (R^T)^{-1} R^T Q^T Y = R^{-1} Q^T Y}$$

Are obtained:

$$\boldsymbol{\beta = R^{-1} Q^T Y} \qquad (\mathbf{2})$$

III. THE LIMITS OF THE MULTIPLE LINEAR REGRESSION

Multiple linear regression have been proved to be successfully used in a variety of data mining applications. But, it will face problems when being applied to deal with

large scale data due to its intensive computation from many levels of training process against large scale data.

As a result, training Multiple linear regression on a single PC is usually very time-consuming, to finish or even cannot be done.

Therefore, we tested the speedup of training process of multiple linear regression against large scale data to discover its limits.

Experiments are carried out on ordinary computer, with 4 GB memory and a processor of 2.67 GHz by varying the number of observations (m) and the number of explanatory variables (n).

The results show that training speed of large scale Multiple linear regression has limited on 500,000 observations and 100 variables which reached quickly at this size caused by memory heap space which can't load all of this volume.

Even if we try to avoid this shortcoming either by exploiting vertical scalability that is, using the most powerful machine available or by working on only subsets or samples of the data. Both approaches have severe limitations. First, vertical scalability is inherently limited and expensive: using a more powerful server will certainly analyze a large volume of data but it will always be limited by its memory size, due of the total load of the datasets. Second, use sampling methods may lose relevant values of this mass of data.

The figure below can give an idea about the performance of basic multiple linear regression's train

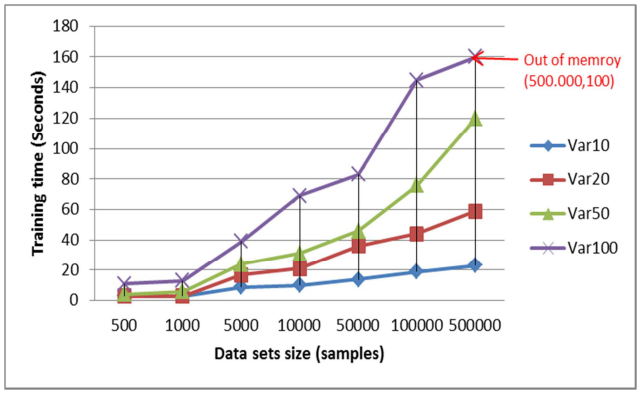


Fig. 2: Measurements and performance tests of the rise in the scale for the multiple linear regression

To overcome these limitations and adapt the multiple linear regression to this huge amount of data, we will present a new computational approach. It will be massively parallel to absorb this mass and increase the performance.

IV. MR-RLM: MapReduce with Regression Linear Multiple

A MapReduce-based Multiple Linear Regression is proposed by this paper to make conventional Multiple Linear Regression work effectively in distributed environment like the cloud computing platform in Amazon Web Services. Our method has three steps.

The following part describes the detailed introduction of the three steps of our method.

A. Implementation of MR-RLM

The algorithm takes as parameters "Block size" used to divide the big training data correspond to independent variables x_i datasets when uploaded to HDFS, into many small blocks. Each block will be delivered to a Mapper:

$$\mathbf{X}_{(\mathbf{m},\mathbf{n})} = \begin{bmatrix} \mathbf{X_1} \\ \mathbf{X_2} \\ \mathbf{X_3} \\ \mathbf{X_4} \end{bmatrix} \qquad where : X_{i\,(m_i,n_i)}\ m_i = block\ size,\ n_i = n;$$

The y_i datasets correspond to explain (dependent) y vector are decomposed also to the same small size blocks and delivered to a Mapper with x_i blocs:

$$\mathbf{y}_{(\mathbf{m})} = \begin{bmatrix} \boldsymbol{y_1} \\ \boldsymbol{y_2} \\ \boldsymbol{y_3} \\ \boldsymbol{y_4} \end{bmatrix} \quad where : y_{i\,(m_i)}\ ;\ m_i = block\ size,\ n_i = 1$$

Main function :

Input : Dataset X
Vector y

Begin

nbBlocks= Size (X) / BlockSize

Xi :=BlocDivide(X , BlockSize)

End

Output : X_i ; *With i:=1...nbBlocks*
y_i ; *With i:=1...nbBlocks*

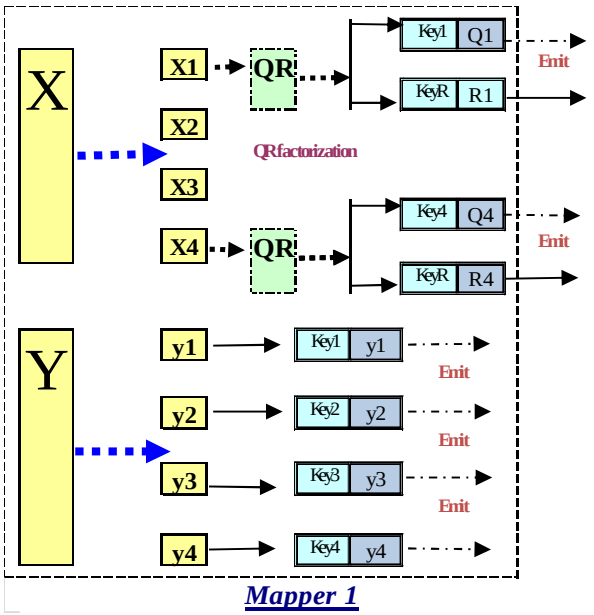
✓ **First step**

1) Map task

Every map task has the same integral QR factorization function. Each map task receives two sub-datasets as input: **Matrix $\mathbf{x_i}$** and **vector $\mathbf{y_i}$**. The input vector y_i (m_i) is sent directly with the key «Key_i» to reduce without any treatment. Each map task computes a local QR factorization for each block x_i received [4]. The results matrices Q_i (m_i,n), R_i (n, n) are associated respectively with the key «Key_i » «Key_R» and

sent them to "reduce". The factorization of **X** looks as follows [4]:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X_1} \\ \mathbf{X_2} \\ \mathbf{X_3} \\ \mathbf{X_4} \end{pmatrix} = \begin{pmatrix} Q_1 R_1 \\ Q_2 R_2 \\ Q_3 R_3 \\ Q_4 R_4 \end{pmatrix} \quad \text{Where: } X_i = Q_i R_i;$$



Mapper 1

2) Reduce Task

In this step, there is a single reduce task. The inputs are: the set of R_i, Q_i matrices and y_i vector from the map Tasks. The Q_i matrices and y_i vectors are sent to the second step.

The matrix R_{temp} is constructed with the intermediate matrix R_i collected from map function [4].

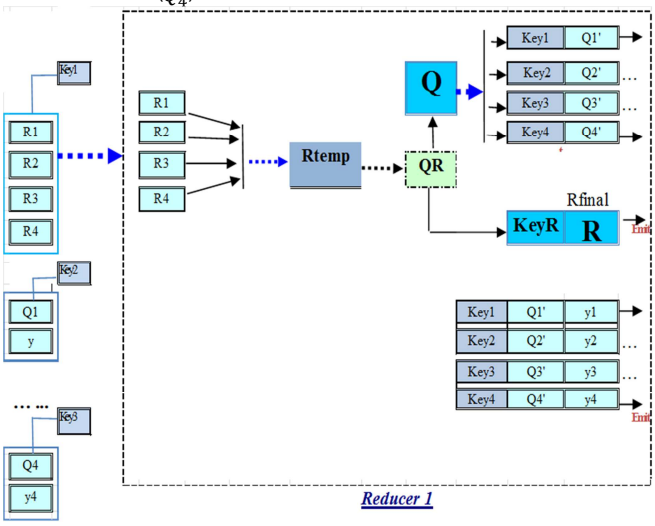
$$R_{temp} = \begin{pmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{pmatrix}$$

A single QR factorization is calculated for this time as follow:

$$R_{temp} = Q'R$$

R is considered to R final delivered with the key «Key_R».
Q' is decomposed to the same small blocks where $size(Q_i') = size(Q_i)$ and delivered with the key «Key_i» to the step 2:

$$Q' = \begin{pmatrix} Q_1' \\ Q_2' \\ Q_3' \\ Q_4' \end{pmatrix} \quad \text{Where size } (Q_i') = size(Q_i)$$



Reducer 1

Step1 :
Mapper function

Input : X_i ; *With i:=1...nbBlocks*
y_i ; *With i:=1...nbBlocks*

Begin

(Q_i, R_i) = ***Map1(X_i)***

Emit(key_i, Q_i)
Emit(key_R, R_i)
Emit(key_i, yi)

End

Output : Q_i ; *With i:=1...nbBlocks*
R_i ; *With i:=1...nbBlocks*
y_i ; *With i:=1...nbBlocks*

Function Map1(X_i)
Input : block X_i
Begin
(Q_i, R_i):= *QRfactorize(X_i)* ;
End
Output : Q_i , R_i ;

Step1:
Reducer function

Function Reducer()
Begin

Function Reduce1(Key$_i$, [y_i, Q_1,Q_2,..,Q_i])
Input : Q_i ; *With i:=1...nbBlocks*
y_i ; *With i:=1...nbBlocks*
Begin
Emit(key$_i$, Q_i)
Emit(key$_i$, yi)
End

Function Reduce2(Key$_R$, [R_1,R_2,..,R_i])
Input : R_{temp} = Matrix [R_1, R2,.., R_i]
Begin
(Q', R_{final}) := QRfactorize(R_{temp}) ;
Q'_i := BlocDivide (Q', BlockSize);
End
Output : *Q'_i ; With i:=1...nbBlocks*
R_{final}
End

✓ **Second step**

The second step also uses only reduce tasks. The input is the set of Q'_i , Q_i and y_i factors which have the same key «Key$_i$» from the first step, received by the same reducer to make the following processing:

The corresponding (Q'_i , Q_i) factors are multiplied together [4]:

$$Q\_result_i := Multiply\ (Q_i, Q'_i) \quad With\ i: =1...nbBlocks$$

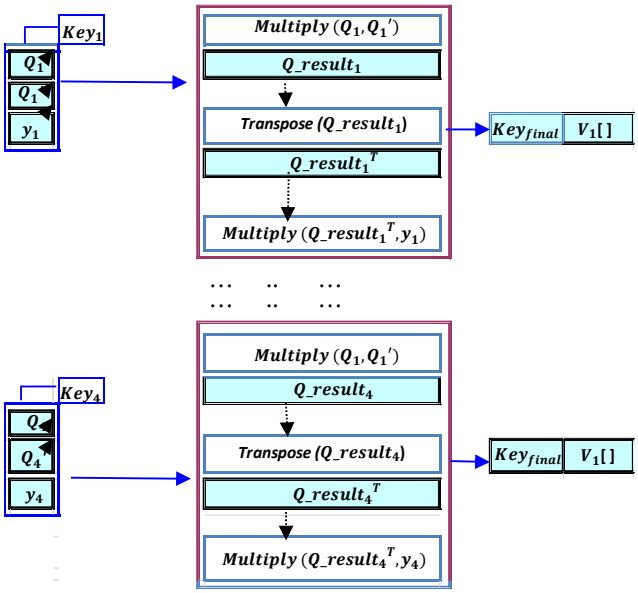
The result of the multiplication Q_result$_i$ is transposed:

$$Q\_result_i^T := Transpose(\ Q\_result_i)$$

Then $Q\_result_i^T$ is multiplied with y_i to emit a intermediate vector V_i :

$$V_i := Multiply\ (\ Q\_result_i^T, yi) \qquad With\ i: =1...nbBlocks$$

The set of V_i factors and R_{final} are emitted with the same key «key$_{final}$ » for the end treatment in the third step



Step 2

Step2:
Reducer function

Function Reducer()
Begin

Function Reduce2 (key$_i$, **List [**Q_i , Q'_i, y_i**])**
Input : List [Q_i , Q'_i, y_i**]**
Begin
Q_result$_i$:= Multiply *(Q_i, Q'_i)*
Q_result$_i^T$:=*Transpose* (Q_result$_i$)
V_i = Multiply *(Q^T, y_i)*
End
Output: V_i[ ] ;

Function Reduce(Key$_R$, R$_{final}$)
Input : R_{final}
Begin
Emit(key$_{final}$, R_{final})
End
Output : R_{final}
End

✓ **Third step**

The third step uses only a one reduce task. The input is the set of vectors of V_i and R $_{final}$ factors from the second step.

The Vi vectors are added together to have the final vector V, Multiplied at the end with R $_{final}$ to have au final the Linear multiple Regression coefficients:

$$\beta := \text{Multiply}(R_{final}, \sum V_i[\,])$$

Step3:
Reducer function

Function Reducer()
Begin

Function Reduce2 (key$_{final}$**, List [**R$_{final}$, V_1,….. Vi**])**
Input: List [R$_{final}$, V_1,….. Vi**]**
Begin

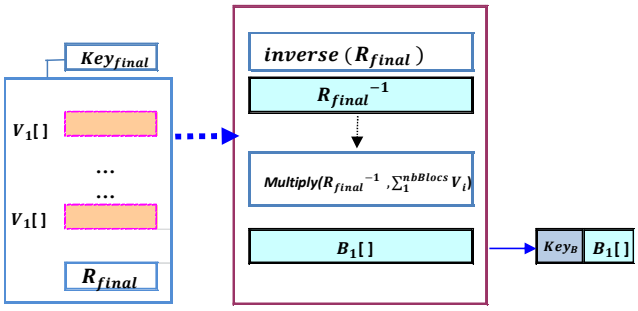
$$R_{final}^{-1} := Inverse\ (R_{final})$$

$$B_i[\,] := Multiply\ (R_{final}^{-1}\ ,\ \sum_{1}^{nbBlocs} V_i)$$

End

Output: B$_i$[ ] ;

End



Step 3

V. EVALUATION

In this section, we evaluate the performance of MR-RLM on large datasets. Thus, we conduct several groups of experiments to compare the training performance of our contribution. The experiments are carried out by varying size of training datasets. We construct the cluster of experimentation by varying also the size and the number of machine for more precision.

A. Experiment Setup

For our experiments, we used Hadoop (version 2.4.2), which includes MapReduce and the Hadoop distributed file system (HDFS). All experiments were run on cluster build in Amazon Web Services (AWS) using Elastic Map Reduce (EMR) service. The dataset used in these experiments is randomly generated,

Our cluster is build up with 2,4,6,8 nodes. Dataset used with 10 Million samples. Each node is equipped with the Intel (R) CPU 2.2 GHZ and 8G memory.

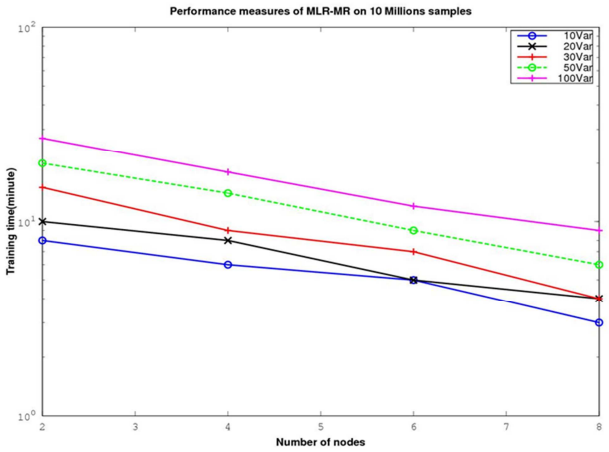


Fig. 4: Performance measures of multiple linear regression with MapReduce.

B. Discussion

Extensive experiments are conducted and the experimental results prove the efficiency and scalability of the method over large scale data on the cloud computing platform of Amazon Web Service (AWS). Fig.4 shows the detailed information. We can see that training Linear Multiple Regression with the scalability of the treatment doesn't any limitation memory and can deal any datasets training size unlike serial method.

VI. CONCLUSION

In this paper, we propose a new method called LMR-MR for linear multiple regression training based on MapReduce paradigm, able to extract its coefficients with massive data. Comparing with the basic training method, LMR-MR effectively solves the problem of the out of memory and accelerates training speed. Our approach provides scalability and therefore need to increase computing capacity, unlike to the Multiple Linear Regression in not distributed environment.
Experimental results show that LMR-MR is able to train over millions of samples in around 20 minutes.

After evaluate the performance of our approach, in the near future, we plan also to evaluate the semantic of the

coefficients β_i calculated, to judge the relevance of LMR-MR when the number of observations becomes large. With this deluge of information, designing and implementing more data mining algorithms in Big Data presents a significant challenge to follow, in order to increase processing performance by avoiding the memory limits without changing their relevance.

REFERENCES

[1] S.Tufféry (2010). Data Mining et statistique décisionnelle, Éditions Technip, nouvelle édition revue et enrichie, pp 10-16.

[2] P.Constantine, D.Gleich (2011). Tall and skinny QR factorizations in mapreduce architectures. Proceedings of the second international workshop on MapReduce and its applications, pp 43-50.

[3] N.Marz, J.Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 2013.

[4] A.Benson, F.Gleich, J.Demmel (2013). Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures pp 44-45.

[5] A.Rajaraman, J.Leskovec, D.Ullman (2010). Mining of Massive Datasets, pp 4-7.

[6] J.Dean, S. Ghemawat. MapReduce: Simplified data processing on large clusters. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI2004), pp 137–150.

[7] L.Breiman, J.Friedma (1985).Estimating Optimal Transformations for Multiple Regression and Correlation. Journal of the American Statistical Association, pp 580-581.

[8] P.Foster, A.Stine (2004), Variable selection in data mining: building a predictive model for bankruptcy," Journal of the American Statistical Association, pp 303-313.

[9] A. Bifet, E. Frank (2010). Sentiment knowledge discovery in Twitter streaming data. In Proc 13th International Conference on Discovery Science, Canberra, Australia, pp 1–15.

[10] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer (2010). MOA: Massive Online Analysis http://moa.cms.waikato.ac.nz/. Journal of Machine Learning Research (JMLR).

[11] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J. M. Hellerstein (2010). Graphlab: A new parallel framework for machine learning. In Conference on Uncertainty in Artificial Intelligence (UAI), Catalina Island, California.

[12] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: a fast and scalable tool for data mining in massive graphs. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada, pp 81–90, 2002.

[13] D.boyd, K.Crawford (2012). Critical Questions for Big Data. Information, Communication and Society, pp 662-679.

[14] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C.Guestrin, J. M.Hellerstein (2010).Graphlab: A new parallel framework for machine learning. In Conference on Uncertainty in Articial Intelligence (UAI), Catalina Island, California.

[15] C. Parker (2012). Unexpected challenges in large scale machine learning. In Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms,

[16] Systems, Programming Models and Applications, BigMine '12, pp 1-6, New York,NY, USA.

[17] S.Das, Y.Sismanis ( ). Ricardo: Integrating R and Hadoop

[18] machine learning. In Conference on Uncertainty in Articial Intelligence (UAI), Catalina Island, California.

[19] C. Parker (2012). Unexpected challenges in large scale machine learning. In Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine '12, pp 1-6, New York,NY, USA