**Project Report**

**Parallelization of Linear Regression**



**Prepared For:**

Dr. Manish Kumar Bajpai

Assistant Professor, Computer Sc.

PDPM IIITDM, Jabalpur

**Prepared By:**

K Avinash Reddy (2011251)

Vellanki Anish (2011264)

Table of Contents

[1. Objective 3](#_Toc416792957)

[2. Introduction 3](#_Toc416792958)

[3. Methodology 3](#_Toc416792959)

[4. Serial Algorithm for Linear Regression using Gradient Descent 5](#_Toc416792960)

[4.1 Terminology 5](#_Toc416792961)

[4.2 Algorithm 5](#_Toc416792962)

[4.3 Pseudo Code 6](#_Toc416792963)

[5. Parallel Methodology 6](#_Toc416792964)

[5.1 PRAM Algorithm 7](#_Toc416792965)

[5.2 BSP Methodology 8](#_Toc416792966)

[5.3 BSP Algorithm 9](#_Toc416792967)

[6. Results 10](#_Toc416792968)

[7. Conclusion 11](#_Toc416792969)

# 

# 1. Objective

Parallelization of Linear Regression

# 2. Introduction

In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variable) denoted by X. The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression.

In linear regression, data are modeled using linear predictor functions, and unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, linear regression refers to a model in which the conditional mean of y given the value of X is an affine function of X. Less commonly, linear regression could refer to a model in which the median, or some other quantile of the conditional distribution of y given X is expressed as a linear function of X. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of y given X, rather than on the joint probability distribution of y and X, which is the domain of multivariate analysis.

# 3. Methodology

We are using Gradient Descent for parameter estimation and inference in linear regression.

Gradient descent is based on the observation that if the multivariable function F(\mathbf{x}) is defined and differentiable in a neighbourhood of a point \mathbf{a}, then F(\mathbf{x}) decreases *fastest* if one goes from \mathbf{a} in the direction of the negative gradient of F at \mathbf{a}, -\nabla F(\mathbf{a}). It follows that, if

 \mathbf{b} = \mathbf{a}-\gamma\nabla F(\mathbf{a})

for \gamma small enough, then F(\mathbf{a})\geq F(\mathbf{b}). With this observation in mind, one starts with a guess \mathbf{x}_0 for a local minimum of F, and considers the sequence \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots such that

\mathbf{x}_{n+1}=\mathbf{x}_n-\gamma_n \nabla F(\mathbf{x}_n),\ n \ge 0.

We have

F(\mathbf{x}_0)\ge F(\mathbf{x}_1)\ge F(\mathbf{x}_2)\ge \cdots,

so hopefully the sequence (\mathbf{x}_n) converges to the desired local minimum. Note that the value of the *step size* \gamma is allowed to change at every iteration. With certain assumptions on the function F

(for example, F convex and \nabla F Lipschitz) and particular choices of \gamma(e.g., chosen via a line search that satisfies the Wolfe conditions), convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

This process is illustrated in the picture to the right. Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient *descent* leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

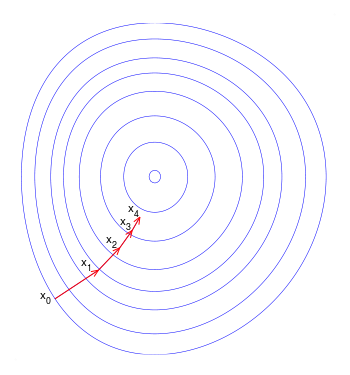


Fig.1. Illustration of gradient descent

# 4. Serial Algorithm for Linear Regression using Gradient Descent

## 4.1 Terminology

**Hypothesis**:

**Loss Function**: It’s a function that maps an event or values of one or more variables onto a real number intuitively representing some cost associated with the event.

**Update Rule**: It’s a gradient descent learning rule for updating the weights of the inputs.

Where is the learning rate

**Learning Rate**: Learning rate is a constant value that controls the size of the adjustments made during the training process. If the learning rate is too high, then the algorithm learns quickly but the error rate is high. If it is low, error is less but algorithm takes longer time to learn.

## 4.2 Algorithm

Loop {

} (for every i ) (Update must be done simultaneously)

## 

## 

## 4.3 Pseudo Code

LinearRegression(data[],gradient)

INPUT: data[]

OUTPUT: gradient

BEGIN

FOR every a and b loop until converge

errors = 0

FOR i = 1 to data.length

fx = a \* data[i] + b

errors += (fx - labelData[i]) \* data[i]

END FOR

gradient = gradient - learningRate \* 1/data.length \* errors

END FOR

END

# 5. Parallel Methodology

* + Machine learning problems become computationally expensive when the complexity (dimensions and polynomial degree) increases and/or when the amount of data increases.
  + Especially on big data sources with hundreds of millions of samples, the time to run optimization algorithms increases dramatically.
  + That’s why we are looking for parallelization opportunities in the algorithms.
  + The error summation of gradient descent algorithm is a perfect candidate for parallelization.
  + We could split the data into multiple parts and run gradient descent on these parts in parallel.
  + We are using PRAM model with the help of OpenMP framework in C language.
  + With the help of work sharing constructs used to split up loop iterations and other OpenMP clauses like Synchronization, scheduling, reduction, we will try to parallelize the gradient descent algorithm for linear regression.
  + This could decrease the run time of the algorithm for large data sets.

## 5.1 PRAM Algorithm

ParallelLinearRegression(data[],gradient)

INPUT: data[]

OUTPUT:gradient

BEGIN

FOR i=0 to p-1

Psum[i] = 0;

END FOR

DO IN PARALLEL FOR ALL Processors Pj 0jp-1

FOR every a and b loop until converge

errors = 0

FOR i = 0 to

fx = a \* data[j\* + i] + b

Psum[j] += (fx - labeldata[j\* + i] \* data[j\* + i]

END FOR

FOR i=0 to p-1

errors = errors + Psum[i]

END FOR

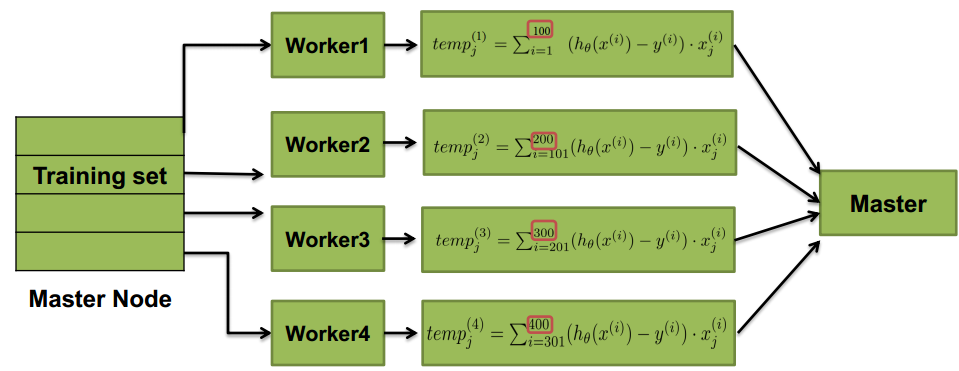
gradient = gradient - learningRate \* (\* errors

END FOR

END

## 5.2 BSP Methodology

* Master node divides the iterations equally to slave nodes
* Each slave calculates its temporary gradient back to the master node.



* Update rule in this case is changed to:
* Weights() are sent to the slave node by the master node where local gradient is calculated.
* All the temporary gradients are sent to the master node.
* Master node sums up the local gradients to find the total gradient .

## 5.3 BSP Algorithm

Linear Regression(data[],gradient)

INPUT: data[]

OUTPUT: gradient

BEGIN

MASTER NODE DO

FOR i=0 to p

temp[i] = 0

END FOR

FOR i=0 to n/p

Pj[i] <= data[j\*n/p +i]

tempj <= 0

END FOR

DO IN PARALLEL FOR PROCESSORS Pj 0jp-1

FOR every a and b loop until converge

errors = 0

FOR i = 1 to Pj.length

fx = a \* Pj[i] + b

errors += (fx - labelData[i]) \* Pj[i]

END FOR

tempj = tempj - learningRate \* 1/Pj.length \* errors

END FOR

temp[j] tempj

END PARALLEL

MASTER NODE DO

FOR i=0 to p-1

gradient += temp[i]

END FOR

END

# 6. Results

We performed our program on Intel(R) Core i7-2670QM CPU Quad Core (No. of threads=8) with 2.20 GHz clock speed and 6GB main memory.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.no** | **Data Set Name** | **Data Set Size**  **(No. of rows, No. of Attributes)** | **Runtime for Serial Algorithm** | **Runtime for Parallel Algorithm** | **Speedup** |
| 1 | Seeds | 210, 7 | 1.3 sec | 2.2 sec | 0.59 |
| 2 | Yacht Hydrodynamics | 308, 7 | 2.1 sec | 2.8 sec | 0.75 |
| 3 | Car Evaluation | 1728, 6 | 8.3 sec | 4.73 sec | 1.75 |
| 4 | Abalone | 4177, 8 | 12.56 sec | 5.89 sec | 2.13 |
| 5 | Page Blocks Classification | 5473, 10 | 19.92 sec | 7.12 sec | 2.79 |

Table.1. Results for different datasets

Fig.2. Comparing Runtimes for different datasets

# 7. Conclusion

In this project we have formulated parallel algorithm for linear regression in PRAM model which reduced runtime for huge data sets. We achieved speedup upto 2.79 on a core i7 computer.

**8. References**

1. Linear Regression wiki link: <http://en.wikipedia.org/wiki/Linear_regression#Other_estimation_techniques>
2. Gradient Descent wiki link:

<http://en.wikipedia.org/wiki/Gradient_descent>

1. Machine Learning by T. Mitchell
2. Data Set Link:

<http://archive.ics.uci.edu/ml/datasets.html>