An Extensive study of Linear Regression Algorithms in Large-Scale Machine Learning

Research Report

Sai Srilakshmi Kanumuri
950808-3061
Sakc16@student.bth.se

Swetha Penumetsa
950510-0868
Swpe16@student.bth.se

I. Group Members' Participation

The group members participated in idea creation and in report writing with the following amount of involvement.

Group Member	Idea Creation	Report Writing
Sai Srilakshmi Kanumuri	75 %	75%
Swetha Penumetsa	25%	25%

Abstract: **Linear regression algorithms are the most widely used statistical techniques in data analytics. But the tremendous increase in data generation limited the performance of these algorithms by their processing times. Previous research described the expertise of several regression algorithms in cost optimization However, there is no precise analysis that uncovers the tradeoffs in their processing times in large scale. This paper addresses this challenge by performing a comparative study on three linear regression algorithms namely, Gradient descent (GD), Stochastic gradient descent (SGD) and Mini batch gradient descent (MGD). These algorithms are trained and tested on two large datasets. Their corresponding processing times and iterations number are noted and then compared. The datasets used in this comparative study are found in sklearn database. Experimental results on test dataset showed that Stochastic gradient descent works desirable in large scale machine learning. This algorithm worked 93.34% and 92.44% faster than Gradient descent and Mini batch gradient descent.**

II. Introduction

Machine learning has grown as the state-of-the-art in recent times. The fundamental task in machine learning is "data prediction". In the last two decades, there has been a tremendous increase in the generation of online content. So there is a need to efficiently dig meaningful information from this huge data[7]. Several techniques are used for predicting data. But the most popular machine learning technique that is used for data prediction is "Regression".

There are two basic operations that are performed by regression algorithms for data prediction. Initially, these algorithms are made to learn from a training dataset. Then they are expected to predict the value of variables when a new test input is given. In the case of statistical data analysis, linear regression algorithms are proven to give the best results with their optimal cost functions. Some of these algorithms are Gradient descent (GD), Stochastic gradient descent (SGD) and Mini batch gradient descent (MGD).

Previous research showed that the above mentioned algorithms learn data efficiently from small and mean sized datasets. But their computational capabilities are not analyzed on large data sets. Since data is being generated with faster rates and new datasets have to be learned within less time, there is a need to suggest an algorithm that works quick on large data sets. This is because, when one has to predict data from a large data set, implementing the best-fit algorithm on a single shot is always better than trying all the available algorithms in a trial and error fashion.

So, the main objective of our research is to find the rapid algorithm among GD, SGD and MGD on large datasets. A comparative study is performed on these algorithms to carry out this work. Expected outcomes like iteration number and processing times are analyzed and conclusions are made.

Since we have to work on quantitative data, we decided to perform an experiment[8]. The dataset used for training these algorithms is taken from sklearn database. This dataset contains 20,000 samples with 14 informatives. The experimental results, i.e processing time and number of iterations performed by these algorithms are obtained by implementing them on test dataset. These implementations are done in python framework. These results are then analyzed using descriptive statistics[9]. Other observations from the experiment are represented using scatter plots and frequency distribution graphs.

Our paper, thus helps the practitioners in choosing a quick algorithm (SGD over GD and MGD) for predicting data in large scale. Also, other novelties in computational complexities like distribution of training examples, variations in cost function, effect of batch size in MGD are discussed in this paper. Suggestable measures to some of the threats to our study are also summarized in this paper.

This structure of this paper is as follows: Section III gives a concrete description of previous work done in the area of study, section IV discusses the aims, objectives of research and research questions. It also gives short summary of research method chosen. Section V gives full description of how exactly the research was performed and the correctness of data collected. Section VI discusses the results and analyses them. All the research questions are answered in this section. Section VII describes the contributions of study, threats to validity and its limitations. The conclusions drawn and future work are discussed in section VIII.

III. Background and Motivation

Processing time of an algorithm is important for many data analytics tasks. In the context of large scale machine learning, there are several challenges. First, there are huge number of sample examples available. Second, it requires large number of iterations to converge. Therefore computing data from large datasets becomes very difficult. Many algorithms are being developed in order to compute large data sets within less time. With the development of online data generation, we have to analyze huge datasets for various projects. Therefore, it is necessary to choose fast processing algorithm among the existing ones.

Gradient descent is the most common algorithm that is used for predictive modeling in linear regression. It is also known as cost optimization algorithm [1]. Although it gives optimal cost function, gradient descent needs human intervention in deciding upon number of iterations and the value of learning rate [1].Furthermore, it involves a lot of pre-work that has to be initialized before the algorithm was actually trained on the chosen data set [1]. These properties of gradient descent may impose certain limitations on its processing time when it is trained on large data set.

In contrast, Stochastic gradient descent has got desirable features in the area of large scale machine learning [2]. Many new algorithms are being developed by applying its features to solve complex tasks. The theoretical guarantees of this method made it highly efficient by pro-viding sharp bounds on the convergence rates [2]. Also, the update rule of this algorithm has been the inspiration in developing an active learning algorithm which minimizes the cost of data annotation [3].

On the other hand, the problems with SGD in minimizing the computational cost of DML are solved by mini-batch sampling[4].Also, the problem with prohibitive computational cost in preference learning is solved by using Mini-batch shrinkage strategy. Using MGD has resulted in minimal computational cost which enforces sparsity [5].

Thus, it is clear that major amount of research is done in cost optimization using these algorithms. But these algorithms are never tested for fast data processing. The properties of these algorithms like iteration factor and processing time play a key role in large scale data processing [6].

Since large scale machine learning has gained huge importance in data analytics, there is an urgent need to analyze the disparities in capabilities of existing algorithms. Thus broader analysis on processing capabilities of chosen algorithms would definitely help practitioners in solving complex problems.

Linear Regression, Gradient descent, Stochastic gradient descent and Mini batch gradient descent:

A. Linear Regression:

Linear regression is a statistical technique used for modeling the relationship between a dependent variable and one or more independent variables. The case of only one independent variable is called simple linear regression. This technique makes use of linear predictor functions whose unknown parameters are estimated from the previous data. It was the most basic kind of regression analysis that is employed in many applications. Some of them are prediction, error reduction, forecasting etc.

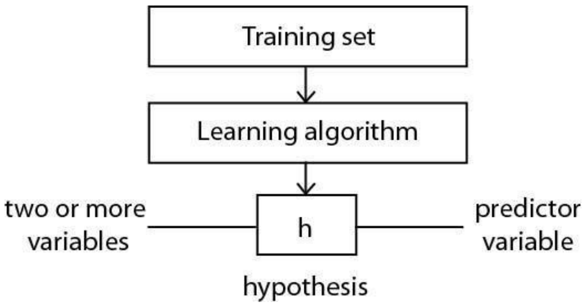


Fig 1: Simple overview Model

The general representation of simple linear regression equation is

$$h_\theta\ (x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

Where theta0 and theta1 are constant values and x is the independent variable.

There are several linear regression algorithms which can be used for predicting continuous data values. Some of them are GD, SGD and MGD. Data prediction using these algorithms is easy only in the case of small datasets. The reason is that each of these algorithms performs iterations on the sample data in order to reach the local minima. In all these algorithms, the theta valued obtained in each iteration directs to the next step towards the local minima until it is converged.

B. Gradient descent:

Gradient descent is most popularly known as optimization algorithm. The cost function is calculated in such a way that, for finding each successive step towards local minima, gradient descent makes use of all the training examples. The values of theta is updated until cost function with optimal value is achieved.

Cost function:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Theta update rule: Say m = Sample size

Repeat {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

(For every j = 0,…, n)

}

where alpha is the learning rate.

C. Stochastic Gradient Descent:

Unlike gradient descent, SGD makes use of a single training example for each iteration in reaching local minima. That means, the theta value gets updated after each iteration on a sample example.

Cost function:

$$\text{cost}\left(\theta,\left(x^{(i)},y^{(i)}\right)\right)=\frac{1}{2}\ \left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

$$J_{train}(\theta)=\frac{1}{m}\sum_{i=1}^{m}\text{cost}\left(\theta,\left(x^{(i)},y^{(i)}\right)\right)$$

Theta update rule: Say m = Sample size

1) Randomly shuffle dataset

```
2) Repeat {
        For i = 1,…, m{
       $\theta_j := \theta_j - d\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j{}^{(i)}$
       (For j = 0,…, n )}
    }
```

Where alpha is the learning rate.

D. Mini batch gradient descent:

Minibatch gradient descent stands in the middle of both Gradient descent and Stochastic gradient descent. It performs iterations on a batch of training examples. The typical range of this batch size is 2-100.

Cost function:

$$J_{train}(\theta)=\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

Theta update rule: Say b = Batch size , m = Sample size

```
Repeat {
       For i = 1, b+1, 2b+1,…, m-b{
   $O_j :- O_j - \alpha\frac{1}{b}\sum_{K=i}^{i+9}\left(h_\theta\left(x^{(k)}\right)-y^{(k)}\right)\ x_j{}^{(k)}$
       (For every j = 0,…, n )
                                                  }
   }
```

where alpha is the learning rate.

Selecting alpha value:

Learning rate or step size, alpha is used to determine the size of the step in gradient descent algorithms. The typical range of alpha is 0.001 and 10.A good pick of learning rate would help in minimizing the cost function as theta approaches to local minima[1].

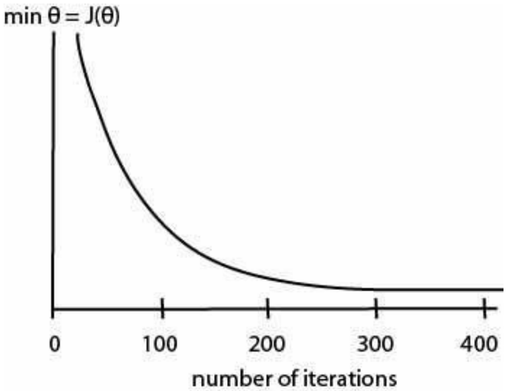


Fig 2: Good Learning Rate

Smaller alpha value: In this case, the step size is small and the algorithm performs large number of iterations to get converged. But this results us in achieving optimal cost function[1].

Large alpha value: In this case, the step size is large and the algorithm performs less number of iterations and converges in less amount of time. But this may skip the optimal cost function[1].

IV. Research Definition and Plan

The main objective of this research is to find the fastest algorithm among GD, SGD and MGD which computes the chosen large data set within less number of iterations and optimal processing time. This can be achieved by answering the three research questions that are listed below:

RQ1: What are the number of iterations and processing time taken by GD for computing the test dataset?

RQ2: What are the number of iterations and processing time taken by SGD for computing the test dataset?

RQ3: What are the number of iterations and processing time taken by MGD for computing the test dataset?

RQ4: Which algorithm among the three, GD, SGD and MGD works fast on test dataset?

The research method we have chosen to answer these questions is “Experiment”. We have to work on quantitative data and conducting an experiment on this kind of data would always help in obtaining desirable results.

Before conducting the real experiment, we have conducted a pilot study in order to identify the minor issues that would potentially destroy our experiment. This helped us in making sure that everything is set up right.

The units of analysis in this research are “Processing Time” and “Iterations number”. Whereas, the subjects involved in this research are the training examples in chosen datasets. The sampling technique we have chosen to train the raw data is “Simple probability sampling”. All the training examples in the datasets are included in the study and they are randomly selected for testing the algorithm. All the data collection in carrying out this study is done by using an "Experimental approach".

The final experiment is then conducted by first training the algorithm on chosen large training set. The algorithm is then tested on chosen test dataset by executing it in python framework. The outputs of this executed python program provide answers to three (RQ1, RQ2, RQ3) of the above mentioned research questions. These results are analyzed using statistical methodologies in order to answer the fourth research question (RQ4).All the observations and conclusions are clearly discussed using the descriptive statistical representations like tables, graphs etc.

V. Research Operation

We trained and tested the three linear regression algorithms on two public datasets. These are found in scikit-learn library. The results obtained by each algorithm are noted down and then they are compared against each other.

A. Datasets:

The training data set chosen for this study is the corpus of random samples. It was built by importing the regression generator from "sklearn.datasets" package. The most common generator used for solving regression problems is make_regression. Both, training and test datasets are built using this generator. The training dataset contains 20,000 sample examples, all different from each other. These samples are labeled by 14 informatives. A sampling algorithm is used to generate these datasets.

The performance of chosen algorithms is evaluated on a test dataset. Like the training dataset, this was also generated by make_regression generator. This dataset also contains 20,000 sample examples but with a single informative. It has to be noted that both the training and test dataset are built with same number of sample examples. This homogenous feature of datasets helps in enforcing sparsity to the study.

B. Experimental setup:

The experimental setup in this study makes use of same parameters in all the three algorithms. All the three algorithms are trained using the same training set chosen. While doing this, the learning rate for these algorithms was initially set to 0.001 which is very small value. Due to this all the three algorithms iterated for a large number of times in order to reach the local minima. Because of large size of dataset, using this small value of alpha took a large amount of time to converge. So, we checked for other values of alpha in a trial and error fashion. We then analyzed the costvalues in all these cases. Finally, the value of 0.01 gave the optimal cost function possible for all the three algorithms.

Also, the convergence rate of all the three algorithms is set to 0.0001. The absolute difference between the cost function and mean squared error is compared with this value inorder to check whether the algorithm converges or not. There is also a limitation on maximum number of iterations performed by each algorithm. We set this value to 40,000. If an algorithm exceeds 40,000 iterations, we conclude that it does not converge for the given set of data values. Even though it gives optimal cost, chosen algorithms involve huge complications in computing data.

Using these parameters, the three algorithms are finally tested on the test dataset. Expected outcomes like number of iterations and processing time taken by each algorithm in computing this test data are collected and compared.

Parameters	Gradient descent	Stochastic gradient descent	Mini batch gradient descent
No. of samples in training data	20,000	20,000	20,000
No. of samples in test data	20,000	20,000	20,000
No. of informatives in training data	14	14	14
No. of informatives in test data	1	1	1
Convergence Criterion	0.001	0.001	0.001
Learning rate(α)	0.01	0.01	0.01

Table 1: Parameters used in Conducting Experimental Study

C. Quality assurance:

As mentioned earlier, a pilot study is conducted prior to the real experiment to identify the errors in the experimental setup. Also, the experimental results are checked with the theta values from scipy linear regression to ensure the validity of experiment.

Algorithm	Experimental Values		Scipy Regression Check	
	θ_0	θ_1	θ_0	θ_1
Gradient Descent	0.0986	49.737	0.0986	49.737
Stochastic Gradient Descent	9.266	13.525	9.265	13.525
Mini Batch Gradient Descent	-28.130	50.419	-28.131	50.419

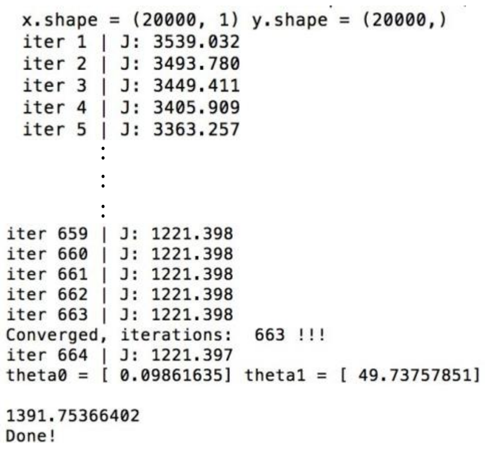
Table 2: Representation of Experimental and scipy Regression Check values to ensure validity of outcomes (theta$_0$, theta$_1$)

VI. Data Analysis and Interpretation

All the experimental observations are analyzed, aggregated and summarized in answering the research questions.

RQ1: What are the number of iterations and processing time taken by GD for computing the test dataset?

Answer: Gradient descent performs 664 iterations on the chosen data inorder to converge at the local minima. It took 1391.753 seconds to reach the minima .Thus the processing time taken by GD for computing the test dataset is 1391.753 seconds.



```
x.shape = (20000, 1) y.shape = (20000,)
iter 1 | J: 3539.032
iter 2 | J: 3493.780
iter 3 | J: 3449.411
iter 4 | J: 3405.909
iter 5 | J: 3363.257
         :
         :
         :
iter 659 | J: 1221.398
iter 660 | J: 1221.398
iter 661 | J: 1221.398
iter 662 | J: 1221.398
iter 663 | J: 1221.398
Converged, iterations:  663 !!!
iter 664 | J: 1221.397
theta0 = [ 0.09861635] theta1 = [ 49.73757851]

1391.75366402
Done!
```

Fig 3: Processing Time and No. of Iterations Performed by GD on Test DataSet

The value of cost function in this case is 1221.397. It is also noted that the value of cost function gradually decreased with every successive step towards local minima. This scenario states that the actual behavior of algorithm serves as an evidence to its theoretical description.

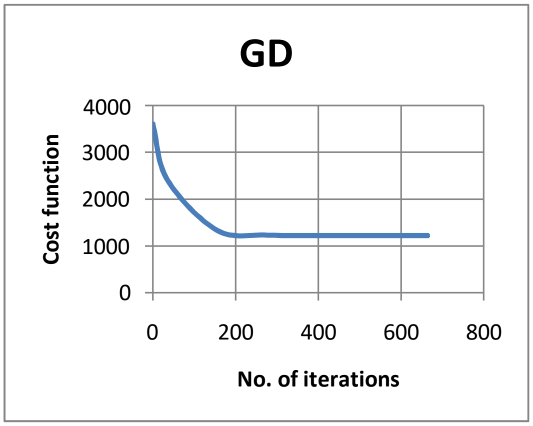


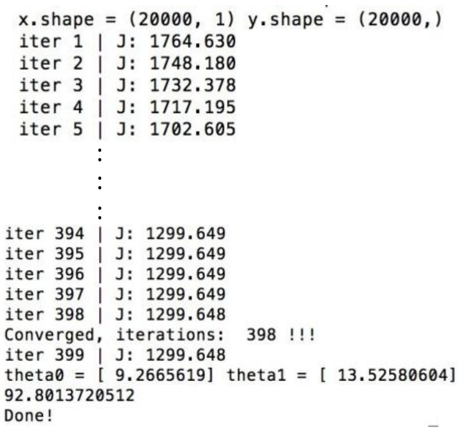
Fig 4: Variation of Cost Function with No. of Iterations in GD

Training data	Test data	No. of iterations	Processing time(sec)	Cost function	θ_0	θ_1
20,000	20,000	664	1391.753	1221.397	0.0986	49.737

Table 3: Performance Measures for GD

RQ2: What are the number of iterations and processing time taken by SGD for computing the test dataset?

Answer: Stochastic gradient descent performs 399 iterations on the test data set inorder to converge at the local minima. It took 92.801 seconds to reach the local minima. Thus the processing time taken by SGD for computing the test dataset is 92.801seconds.



```
x.shape = (20000, 1) y.shape = (20000,)
iter 1 | J: 1764.630
iter 2 | J: 1748.180
iter 3 | J: 1732.378
iter 4 | J: 1717.195
iter 5 | J: 1702.605
          :
          :
          :
iter 394 | J: 1299.649
iter 395 | J: 1299.649
iter 396 | J: 1299.649
iter 397 | J: 1299.649
iter 398 | J: 1299.648
Converged, iterations:  398 !!!
iter 399 | J: 1299.648
theta0 = [ 9.2665619] theta1 = [ 13.52580604]
92.8013720512
Done!
```

Fig 5: Processing Time and No. of Iterations Performed by SGD on Test DataSet

The value of cost function in this case is 1299.648. It is also noted that the value of cost function gradually decreased with every successive step towards local minima. This proves that the algorithm works properly on chosen dataset.

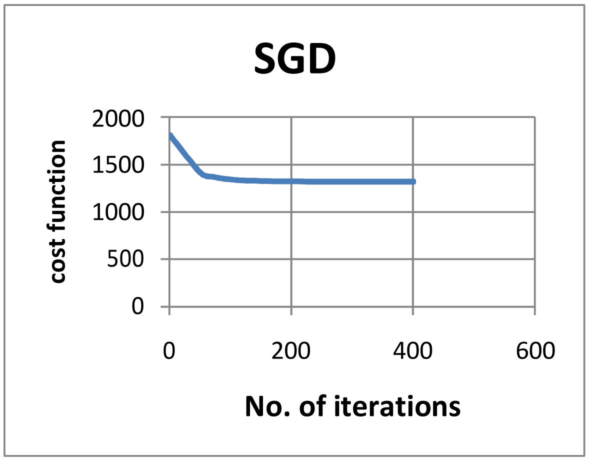


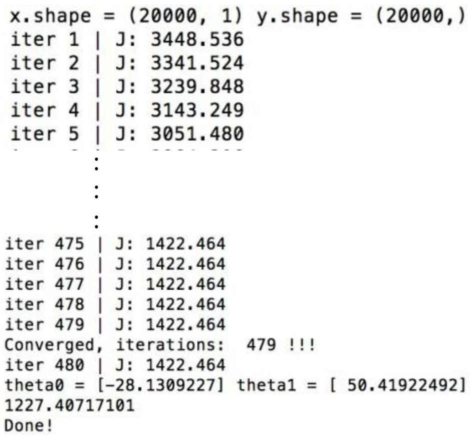
Fig 6: Variation of Cost Function with No. of Iterations in SGD

Training data	Test data	No. of iterations	Processing time(sec)	Cost function	θ_0	θ_1
20,000	20,000	399	92.801	1299.648	9.266	13.525

Table 4: Performance Measures for SGD

RQ3: What are the number of iterations and processing time taken by MGD for computing the test dataset?

Answer: Mini batch gradient descent performs 480 iterations on the test data set inorder to converge at the local minima. It took 1227.407 seconds to reach the local minima. Thus the processing time taken by MGD for computing the test dataset is 1227.407 seconds.



```
x.shape = (20000, 1) y.shape = (20000,)
iter 1 | J: 3448.536
iter 2 | J: 3341.524
iter 3 | J: 3239.848
iter 4 | J: 3143.249
iter 5 | J: 3051.480
              :
              :
              :
iter 475 | J: 1422.464
iter 476 | J: 1422.464
iter 477 | J: 1422.464
iter 478 | J: 1422.464
iter 479 | J: 1422.464
Converged, iterations:  479 !!!
iter 480 | J: 1422.464
theta0 = [-28.1309227] theta1 = [ 50.41922492]
1227.40717101
Done!
```

Fig 7: Processing Time and No. of Iterations Performed by MGD on Test DataSet

The value of this cost function is found to be 1422.464. It is also noted that the value of cost function decreased with every successive step in the beginning and its value was consequently increased until the local minima was reached. This property is in relevance to the theoretical description of Mini batch gradient descent.

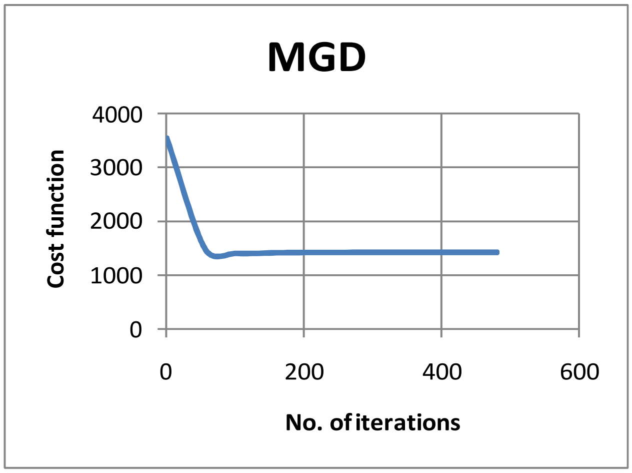


Fig 8: Variation of Cost Function with No. of Iterations in MGD

Training data	Test data	No. of iterations	Processing time(sec)	Cost function	θ_0	θ_1
20,000	20,000	480	1227.407	1422.464	-28.13	50.419

Table 5: Performance Measures for MGD

RQ4: Which algorithm among the three, GD, SGD and MGD works fast on test dataset?

Answer: SGD works much faster than GD and MGD on test dataset. The processing time of SGD is 92.801sec. Thus it worked 93.34% faster than GD and 92.44% faster than MGD.

This algorithm performed an optimal number of iterations (399) to converge at the local minima.

Algorithm	Processing time	No. of iterations
Gradient Descent	1391.753	664
Stochastic Gradient Descent	92.801	399
Mini Batch Gradient Descent	1227.407	480

Table 6: No. of iterations and Processing Times of three Algorithms Involved in Comparative study

VII. Discussion

The three regression algorithms GD, SGD and MGD are trained and tested on chosen data sets. Training and testing of these algorithms is performed in python framework.

Gradient descent took large amount of time to converge at the local minima. This is because of its property of performing summation on all the samples in each iteration. So it took 1391.735 seconds and 664 iterations to compute the test dataset. But the sample distribution using GD on this large dataset is not so good as when it is used with small datasets.

Mini batch gradient descent stood quiet forward to Gradient descent with a processing time of 1227.40 and 480 iterations on test dataset. Depending on the batch size, this algorithm performed different number of iterations on the test data set. A batch size of 10 is used in this experiment which resulted the algorithm in converging with optimal number of iterations. Unlike Gradient descent, data distribution is slightly better in this case.

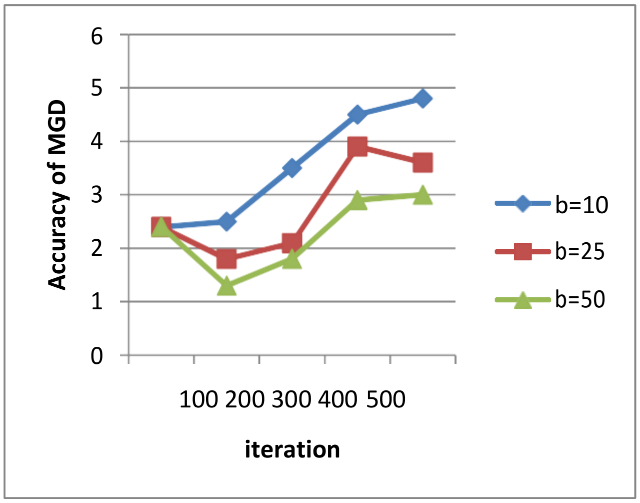


Fig 9: Effect of Batch Size on Accuracy of MGD

Whereas, Stochastic gradient descent gave a contrary performance to both gradient descent and Mini batch gradient descent. The algorithm with its widely applied theta update rule took only 92.801 seconds and 399 iterations to compute the test dataset. These optimal values of processing

time and iteration number revealed the desirable features of SGD over GD and MGD in large scale machine learning.
Besides achieving these optimalities, Stochastic gradient descent worked equally well in data distribution of training examples.

All the sample examples in the test data set are uniformly distributed by SGD. This feature of this algorithm helps in mitigating the problem of overfitting in data.

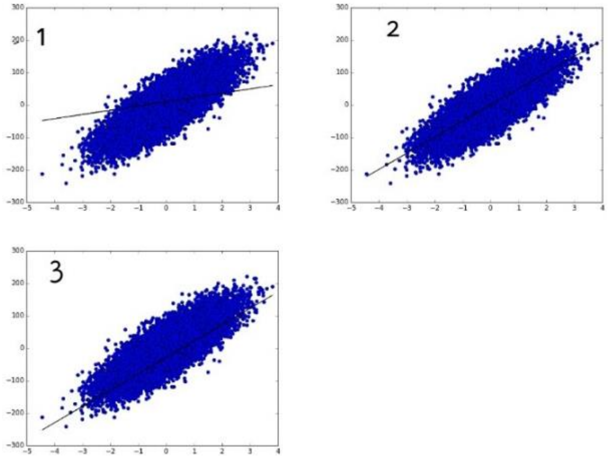


Fig 10: Distribution of Sample Examples by three Algorithms 1. Gradient Descent; 2. Stochastic Gradient Descent; 3. Mini Batch Gradient Descent.

A. Contributions:

The main contribution of this comparative study is that it evaluated the processing capabilities of GD, SGD and MGD in large scale machine learning. As discussed above SGD with its optimal processing time and iteration factor is proved to be the best fit over GD and MGD. Along with other novelties like variations in cost function with each iteration and effect of batch size in MGD, this comparative study minimized the gap in previous research.

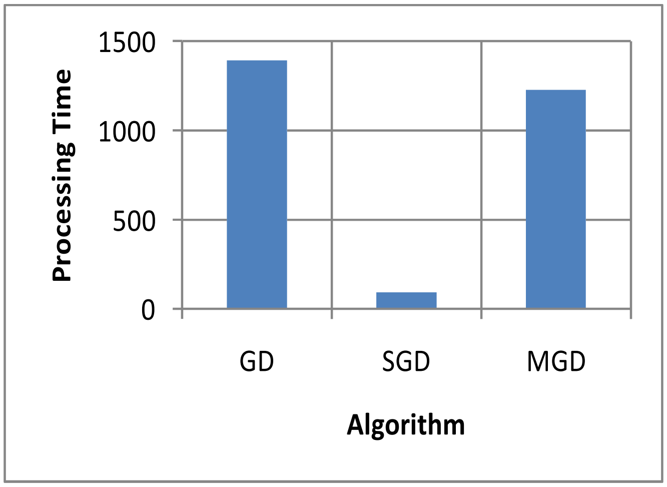


Fig 11: Processing Times of Algorithms Involved in the Study

B. Threats to Validity:

The following are some of the threats to validity to our comparative study.

- Experimenter's bias: This occurs when the experimenters involved in the study alter the observations by creating expected outcomes by themselves. We used "double-blind" technique to combat this bias.
- Measurement bias: This is because of errors in data collection and data measurement. We mitigated this threat by employing "Triangulation approach" to our study. By using this approach, all the data collected and evaluated in the study is validated through cross verification.
- Statistical regression: This occurs when there is any unreliability in the value of dependent variables. The extreme scores may tend to regress at unexpected rates. We resolved this issue by generating active and regularly ordered sample features in our datasets.
- Population validity: When the sample datasets used in the study are less representative, generalizing observations from the results will become difficult. 14 informatives are used in the experimental datasets to mitigate this problem.
- Instrumentation bias: This threat is due to the change in observers in the midst of experiment. This shift in witnessing persons may result in "missing experimental observations". This issue is solved by employing only one observer until the experiment is finished. For more precise analysis, second round of testing is performed under similar conditions used in the first round.
- Procedural bias: This occurs when one of the observers involved in the study fasten the experimental procedure by violating the specified criterion. This may result in poor outcomes. We overcomed this conflict by employing individual evaluation of experimental status by all the group members from time to time.

C. Limitations:

The main limitation of this study is that this paper did not address the problem of "Overfitting" of data. Since we are building models in large scale machine learning, there is a possibility of occurrence of "overfitting". This is because of the noise present in the datasets.This noise may effect the accuracy of the experimental results.

VIII. Summary and Conclusions

This paper elucidates the processing capabilities of GD, SGD and MGD in large scale machine learning. The processing times and iteration numbers of these algorithms are evaluated in this comparative study. Experimental results proved that SGD works quick on large datasets besides giving good values for cost function. It is observed that SGD works 93.34% and 92.44% faster than GD and

MGD respectively. From the experimental observations, it is found that SGD converges within 399 iterations at the local minima. Whereas GD and MGD took 664 and 480 iterations respectively to converge. These results describe the desirability of SGD over GD and MGD in large scale machine learning. It is also found that the accuracy of MGD decreases with increase in its batch size. Thus our research reduced the gap of analyzing processing capabilities of Linear regression algorithms in large scale machine learning.

For future work, we would like to overcome the problem of Overfitting in our datasets. We would also like to work on finding the memory space used by these algorithms while working on large datasets.

REFERENCES

[1] Lubis, Fetty Fitriyanti, YusepRosmansyah, and SuhonoHarsoSupangkat. "Gradient descent and normal equations on cost function minimization for online predictive using linear regression with multiple variables." *ICT For Smart Society (ICISS), 2014 International Conference on*. IEEE, 2014.

[2] Agarwal, Alekh, SahandNegahban, and Martin J. Wainwright. "Stochastic optimization and sparse statistical recovery: Optimal algorithms for high dimensions." *Advances in Neural Information Processing Systems*. 2012.

[3] Cai, Wenbin, Ya Zhang, and Jun Zhou. "Maximizing expected model change for active learning in regression." *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 2013.

[4] Q. Qian, R. Jin, J. Yi, L. Zhang, and S. Zhu, “Efficient distance metric learning by adaptive sampling and mini-batch stochastic gradient descent (SGD),” *Mach. Learn.*, vol. 99, no. 3, pp. 353–372, 2014.

[5] Chen, Xi, et al. "Learning preferences with millions of parameters by enforcing sparsity." *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010.

[6] C. Ngufor and J. Wojtusiak, “Learning from Large Distributed Data: A Scaling Down Sampling Scheme for Efficient Data Processing,”*Int. J. Mach. Learn. Comput.*, vol. 4, no. 3, pp. 216–224, 2014.

[7] Li, Wenjuan, and WeizhiMeng. "An empirical study on email classification using supervised machine learning in real environments." *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015.

[8] M. V. Zelkowitz and D. R. Wallace, "Experimental models for validating technology," in *Computer*, vol. 31, no. 5, pp. 23-31, May 1998.

[9] W. Mendenhall, R. J. Beaver, and B. M. Beaver, *Introduction to Probability and Statistics*, Cengage Learning, 2012.