Parallelizing Logistic Regression for Biological Experimental Design

Wansang Lim and Valerie Angulo

Abstract— **Logistic regression is a statistical analysis tool used as a predictive analytic in a variety of disciplines. In this paper, we focus on parallelizing binary logistic regression analysis for predictive analytics in data science for environmental science datasets. Parallelizing logistic regression would improve computation time and decrease memory latency when analyzing large data sets, allowing for more data to be processed faster. In this paper, we compare a sequential implementation of binary logistic regression with a CUDA implementation, a parallelized R implementation and a multiprocessor OpenCV version, looking at the relationship between dataset sizes and time taken to process the data. Our findings point to improvements in computation time and less memory latency for CUDA versions of logistic regression.**

I. INTRODUCTION

Logistic Regression is used as a predictive analytic in many disciplines ranging from biology and conservation to business. It models a binary dependent and one or more binary or nonbinary independent variables. This is useful in cases of observing phenomena that may occur due to a specific event. The purpose of logistic regression is to predict the occurrence of phenomena based on acquired current data. Mathematically, the binary dependent variable is either 0 or 1 and indicates the presence or absence of a certain condition, such as alive/dead or win/lose, that may be related to the independent conditions. In this paper, we are primarily concerned with the applications of binary logistic regression analysis for predictive analytics in data science, particularly for environmental science datasets. For our purpose, logistic regression is used to classify dependent variables into different groups.

Currently, there is an abundance of large datasets that are open sourced and easily accessible to the public. This is especially beneficial for scientific research. However, processing large data sets is time consuming and resource intensive for CPU in terms of memory and computation time. Sequential implementations of logistic regression require a lot of time to process smaller amounts of data and have a high memory latency. Implementation of a parallelized binary logistic regression would allow for an increased amount of data to be processed in less time and with less resource intensive computations. Logistic regression is an excellent analytic to parallelize because it primarily utilizes matrix multiplication, which has some testable characteristics to be converted to parallel code. The key point of logistic regression is to calculate the Y parameter. The procedure to calculate the parameter Y involves matrix inversion after calculating the logit, which converts binary variables to continuous variables. In this paper, we compare a sequential implementation of binary logistic regression with a CUDA implementation, a parallelized R implementation and a multiprocessor OpenML version, looking at the relationship between dataset sizes and time taken to process the data. Our findings point to improvements in computation time and less memory latency for CUDA versions of logistic regression, as well as parallelized R implementations.

II. BACKGROUND

Logistic regression is a predictive analytic that is used to categorize data into different groups. It can be binomial, ordinal or multinomial and is based on linear regression; a logistic regression estimates a multiple linear regression function. It is the correct regression to use if there is a binary dependent variable, such as time spent studying versus whether a student passes or fails a course, or other situations depending on independent variables such as win/lose, dead/alive, yes/no, present/absent etc. Binary logistic regression is used to describe the relationship between a binary dependent variable and one or many independent variables. This predictive analytic tool is useful in data science and many other fields for predicting how likely an event will occur when certain independent factors are present. Following are the steps needed to implement a logistic regression.

A. Linear Regression

In order to do a logistic regression, we must first start with a linear regression.

$$Y = b_0 + b_1X_1 + b_2X_2 + b_nX_n$$

In this linear model, the xs are the predictors/independent variables and the Bns are the parameters of the model or the coefficients of the independent variables, with b_0 being a constant term. The Y value is the outcome /dependent variable and can vary from negative to positive infinity for a linear regression.

B. Logistic Function

The next step is to turn the linear regression into a sigmoid function, also known as a logistic function.

$$p(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = \frac{e^{(b_0+b_1X_1+\ldots+b_nX_n)}}{1+e^{(b_0+b_1X_1+\ldots+b_nX_n)}}$$

P(x) is the probability of the dependent variable equaling a success. E(Y) is the expected value of the binary dependent variable Y. X is the linear regression formula. The function is as such because the range of p(x) should be between 0 to 1, rather than -infinity to +infinity. This function provides a limited range of values for probability, so the odds of the probability must be taken next to fully convert this logistic function into a logistic regression.

C. Logistic Regression

The logistic regression is the natural log of the inverse of the logistic function. The core of logistic regression is estimating the log odds of an event, also known as the logit of the probability, where the log odds is a prediction of the odds of a dependent variable based on one or more binary or real valued independent predictors/x-values.

The odds are determined by the inverse of the logistic function, where the probability of an outcomes success (Y) is divided by the probability that it will not occur (1-Y). To determine the log-odds of an event occurring, we must take the natural log of this inverse. The purpose of this is to fit the logit of the probability of success with the predictors/x-values. We are then able to obtain a continuous predictor for the odds of an event happening [6].

$$logitp(x) = ln[\frac{p(x)}{1-p(x)}]$$

The dependent variable turns into a logit variable (the natural log of the odds of the dependent variable occurring or not). We then estimate the probability of the occurrence of a certain event based on the independent variables. The logit serves as a link between the linear regression and logistic function. The probability of success of obtaining a particular value from a binary dependent is equivalent to the odds of the dependent variable Y equaling a particular case.

$$ln[\frac{Y}{1-Y}] = b_0 + bX$$

Logistic regression seeks to find the equation that best predicts the value of Y for each value of X. The Y variable in logistic regression is the probability of obtaining a particular value of a binary variable, whereas the Y variable in linear regression is measured directly.

III. LITERATURE SURVEY

We've looked at a variety of papers regarding parallelizing logistic regression, as well as the applications of parallelized and sequential logistic regressions. The following related works are presented because they either implement a biostatistic(?) model of logistic regression or they compare sequential and parallel versions of logistic regression.

A. Biostatistic Logistic Regression

The algorithm that we have coded sequential and parallel versions of is applicable to biology and environmental studies. The work done in ”Deforestation Modelling Using Logistic Regression and GIS” [1] uses logistic regression in the manner that we implement and parallelize. In their work, Bavaghar looks at binary logistic regression, determining whether deforestation is present or not in an area based on slope as an independent variable. The setup of the experiment is similar to ours, and the work is what drove us to parallelize logistic regression for scientific fields. While a sequential version of the logistic regression is analyzed, there is no attempt to parallelize it. The data input is also far different than ours, where the focus is to use GIS with logistic regression, making comparison with this work difficult. Bavaghar analyzes digital thematic-topographic maps with 63 different data layers such as forests, ranges and gardens. Our tests read data from a file containing random input values that fall within a certain range to represent our x and b values, rather than topographic maps or open source data. Bavaghar

B. Parallelizing Logistic Regression

Others, such as Liang, Choi et al. [2], have parallelized the logistic regression for machine learning. They have looked at parallelized regressions in distributed platforms, parallel algorithms and sub-linear approximations. They worked with Hadoop and Spark, two distributed systems used in data science and analytics, to run sequential and parallel versions of code and compare platform/algorithm combinations with sub-linear algorithms specific to machine learning. This is clearly useful for machine learning but is not applicable to other fields that utilize logistic regression. Liang, Choi et al. focus on parallelizing machine learning algorithms, while we are looking to parallelize logistic regression itself. They used five open datasets related to machine learning to make sure that they could accurately compare their methods whereas our data is randomly generated and doesnt come from open source. Using open datasets are beneficial because the results allow for comparability with the performance of sequential machine learning algorithms that already exist.

The work done by Peretti, Alessandro, and Francesco Amenta [3] focuses on parallelizing logistic regression for the purposes of machine learning, bioinformatics and data analysis. While it is primarily a machine learning approach and our work aims to be more generalized, the parallelization is reached by using CUDA parallel programming support, which we use in our experimental design as well. However, not much detail is gone into about how CUDA is implemented, only that different CUDA software versions are compared in determining the loss function for the machine learning logistic regression. Nevertheless, this work is useful in that it this approach is applicable to bioinformatics and data analysis. The dataset used is a real dataset containing anonymized patient information obtained from the Wisconsin Diagnostic Breast Cancer database. It contains 569 instances, with each instance corresponding to a patient. This is beneficial in that the results obtained can be tested in real life, and they are. From the estimated Y values obtained per patient, there was a better chance at predicting whether a patient had breast cancer or not.

C. Implementations Used or Referenced

In Evaluating Parallel Logistic Regression Models [2], Liang, Choi et al. implement a binary logistic regression, and we reference the parallel algorithm approach, but without the distributed platform or sub-linear approximation. They analyze their experimental results in accuracy, efficiency, scalability, robustness and running time. We analyze our results in terms of accuracy and running time and use this work as a reference on how to proceed with such an analysis.

From Bavaghar's work in Deforestation modeling using logistic regression and GIS [1], we use a similar logistic regression and proceed with the sequential code and then the parallelized code in a step-by-step approach as well. We break down our implementation into linear regression, then logistic function and finally into logistic regression in order to code logistic regression. Bavaghar uses R^2 chi square test to determine the fitness of the model, which is something we consider in our work but did not have enough time to do.

IV. PROPOSED SOLUTION

We aim to parallelize the logistic regression as applied to biology and data science by using CUDA. There has been a lot of research on parallelizing logistic regression for machine learning techniques and algorithms, but nothing that weve seen that is general enough to cover parallelized logistic regression outside of machine learning purposes. The biostatistic research that we have come across uses sequential logistic regression with other new technologies but does not attempt to parallelize the logistic regression itself. From our research, we havent found others who attempt to parallelize a general logistic regression using CUDA. Therefore, we propose to parallelize the logistic regression in order to improve predictive behavior in a number of fields. The purpose of this is to show that logistic regression can be parallelized and used to analyze more data in less time.

V. EXPERIMENTAL SETUP

A. Assumptions

For our experiment we are implementing a binary logistic regression where our dependent variable is either 0 or 1 while our independent x variables are real numbers ranging from -3 to 3. To make sure our generated data set is valid, we followed the assumptions of data used for logistic regression applications. This involves having a binary dependent variable, not having outliers present in the data or strong correlations between independent data sets, and making sure no x values are below -3.29 or above 3.29 [8]. Another consideration we took into account is the model fit. Having more independent x variables increases the amount of variance (R^2) but adding too many x variables will decrease the generalizability of the predictive analytic, thus decreasing its accuracy. During our analysis we made sure not to have a very large number of columns, which represents our x variables, so that we could model prime data sets used for logistic regression. To analyze the accuracy of our logistic regression, we computed its goodness-of-fit based on the Chi square test, as well as the amount of variance R^2.

B. Methods

We decided to use a matrix of random values for our input data, where the user chooses the number of rows and columns. The rows represent the number of samples that data is acquired for while the columns within each row represent the number of independent x variables. For example, rows could represent a patient and the columns could represent independent factors that may determine whether the patient has a certain illness or not. We chose this presentation of data so that we could calculate matrix multiplication and inversion for sequential and CUDA code.

To code our sequential version of a logistic regression, we broke down the logistic regression as described in the background section. An array of all the x values are read in from the file and transposed into a new array. X and X are then multiplied together (XX). We then calculate the inverse of the transposed X which is (-1*X') and then we perform matrix multiplication on (XX) and (-1*X) . The value obtained is then multiplied with Y values to produce B values. From these B values that are produced, we populate an array of new X values and combine them with the B values to give us a predicted Y value.

For our CUDA code, we used the same logic as the sequential code but we parallelized matrix multiplication, inverse etc

Our random data is composed of values between -3 and 3 where we perform a logistic regression on the first values of each row, which then represent our Y values. Our rows therefore represent individuals (such as students, mice, etc) with a dependent Y and row-1 number of independent xs. For our experiment, we imagine that the Y values are dependent on the x values within each row, for experimental purposes. The rows actually represent groups of individuals which is why our Y values are not binary. Toxicologists use this for logistic regression, we want our project to match biological and environmental research.

The random data has logistic regression as described in the background performed on it for the Y, the outputs are B values that are the averages of each individual/group.

Our code follows the multiple regression model because this model is used in biology for multiple variable.

C. Machine Setup

We used NYU linux CIMS accounts to run our sequential and parallelized logistic regression code. Before running the sequential and parallelized code, we made sure to generate the random data files that we needed to run our code with. To do this we compiled the random data program and then ran it with our specified number of rows and columns. From here, to run the parallelized code we used the cuda5 machine on the NYU prince cluster and loaded the cuda version 9.1 module. For the sequential code, we used the snappy machine on NYUs prince cluster to obtain faster results for sequential code. We used the time flag to record time taken to compute the logistic regression of our sequential code and CUDA code, and we used nvprof to profile our CUDA code.

D. Problem Size

We tested various problem sizes ranging from an input of 16 rows up to 1,000 rows for the sequential code, with column numbers increasing in multiples of 2, up until 12.

For CUDA, we tested much larger row and column numbers as well as smaller numbers to try to relate it with our sequential test sizes.

our code does matrix multiplication.. the column numbers are the independent x variables that lie on the x-axis of a matrix and our row numbers represent the individuals being tested, the data points that we are trying to obtain Y values for and these values lie on the y-axis of the matrix.

VI. Results and Discussion

For our results, we used time to obtain the real time our programs took to compute the B values. We took an average of 5 time values per column per row to obtain a more accurate time.

A. Sequential Results

The results for our sequential code show a very small difference per change in row value (number of individuals). However, there is an increase in time with each increment of the column number, the number of independent variables, that appears to increase exponentially (Fig 1). Beyond column 12, we were unable to obtain results because time took longer than 1 hour, even when we use crunchy3 and snappy3 machines for quicker sequential computing. This limited our ability to properly compare our sequential results with our CUDA results. Differences in CUDA for column values as small as 2-12 did not provide much insight. We still obtained results for CUDA values that match sequential values.

Figure 1 shows that the difference between row number is very small, but the difference between column numbers with a constant row number vary exponentially. In the table CITE/MAKE THIS TABLE!!! the differences of the time between columns for a constant row number are seen. Because the values of the columns varied greatly, but the values between rows stayed close, we decided not to test more rows out. The column numbers are the independent variables, the values along the x-axis of a matrix. Through our results, the more variables that are present per row, the exponentially greater time it takes to do matrix multiplication and inverse on the data matrix. We believe this is due to the recursion present in the sequential code, particularly in calculating the matrix inversion. We believe that there is little variation among row values because in calculating logistic regression multivariable logistic regression, the data samples/individuals/rows are averaged together to find a general Y value for all samples, but the inversions and high overhead operations are being performed on the independent variables, or column values. Therefore, sequential code is able to handle a large value of rows.

We tested with row values of 10,000, 50,000 and 100,000 but received segmentation fault errors (TABLE X). However, a slow increase of time is noticed, where at row = 100,000 the time it takes to compute 2 variables per row increases by 10, compared to where row = 100.

(Make a table like sequential but for CUDA?. Scatter? with Rows = 20 and plot the columns, showing how it exponentially increases. the average of 5 times per column value. This can be a log graph?)

B. CUDA Results

We checked if exponential performance difference depends on column number for our parallel code, as it does with our sequential code and found that the CUDA performance does not depend on column number up to 100 (Fig 2). So we focused on the number of rows for our CUDA code. Fig 2 shows that the column number of CUDA code up to 100 has no effect on performance. For our purposes, the number of independent variables, or column number in our project, is less than around 20 in papers for biology and environmental modeling.CITE THIS. Because of this, we checked the effect of sample numbers while the column number is limited to 20. In both column number 10 and 20, the performance of CUDA code is unstable up to 1000 and 2000. After that, the performance is almost proportional to sample or row numbers (Fig 3 and Fig 4).

The nvprof for row number 1,000, 10,000, 100,000 and 200,000 is checked with column number 10 (Table 1). The time for cuda matrix multiplication reduced proportionally with increase of row numbers from 1,000 to 200,000 and occupies over 85%. Even in small number of rows such as 1,000, the time percentage for matrix multiplication is 99.89%. This result indicates that our CUDA code is compute bound before the row number 200,000.

C. Comparing Both

For the sequential code, at column 12, the time is around 400 seconds up to 1000 rows (Fig 1). We checked cuda performance for the same condition. The time for CUDA code with a row of 1000 dramatically reduced to 0.3 seconds, making it 1333 times faster than its other values(CUDA values or sequential values?).

Currently, we are planning to compare algorithm for matrix inversion. For sequential code, we use adjoint and cofactor for matrix inversion. It has recursion which is not good for cuda code. We used Gaussian elimination to get matrix inversion. With this comparison, we can remove algorithm effect. Our row number limitation is about 100,000 20,000. We will try tiling for matrix multiplication to expand row numbers.

VII. Conclusions

Summary of what we found, why its useful, future implementations or things we'd do or change.

- recursive nature of logistic regression in sequential code greatly limits the time it takes to compute an increasing number of independent variables.
- CUDA code gracefully handles a large input of independent variables but could be improved in number of sample size.
- In practical terms, biology and environmental science use around 20 independent variables at maximum.
- We would like to improve our CUDA code to make it more efficient in processing a larger sample size, such as a size of 1,000,000 individuals or more.
- We find that our CUDA code is adequate at processing large numbers of independent variables.

- We would like to balance CUDA codes processing large amounts of independent variables for the ability to process very large sample sizes.

TABLE I

SEQUENTIAL CODE TIME RESULTS PER ROW AND COLUMN VALUE. TIME IS IN SECONDS, THESE RESULTS WERE OBTAINED THROUGH SNAPPY3 CIMS MACHINE.

Col	Row = 16	Row = 20	Row = 50	Row = 100	Row = 500				
2	0.0168	0.0104	0.0134	0.0116	0.0106				
4	0.016	0.01	0.0046	0.0112	0.0104				
6	0.0148	0.0112	0.0062	0.0134	0.081				
8	0.031	0.035	0.0276	0.0364	0.1254	[illegible]	[illegible]	NAN	NAN
10	2.5764	2.602	2.615	2.5568	2.8394	2.5564	NAN	NAN	NAN
12	397.905	395.137	396.281	396.541	395.702	395.609	NAN	NAN	NAN

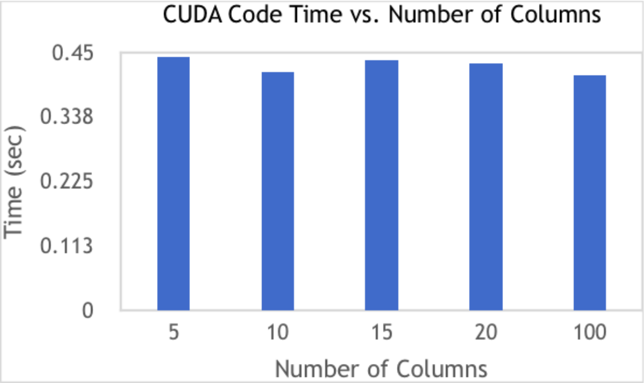

Fig. 2. Performance difference of CUDA code between column number (5 20). The row number is set to 20 for all column numbers except for column = 100. The row number = 100 for column = 100.

TABLE II

TIME(PERCENTAGE) OF PROFILING RESULT FROM NVPROF. THE COLUMN NUMBER IS SET TO 10

operation	Row = 1,000	Row = 10,000	Row = 100,000
matrix multiplication	99.89	99.49	90.81
CUDA memcpy DtoD	0.04	0.33	5.32
CUDA memcpy HtoD	0.03	0.08	3.78

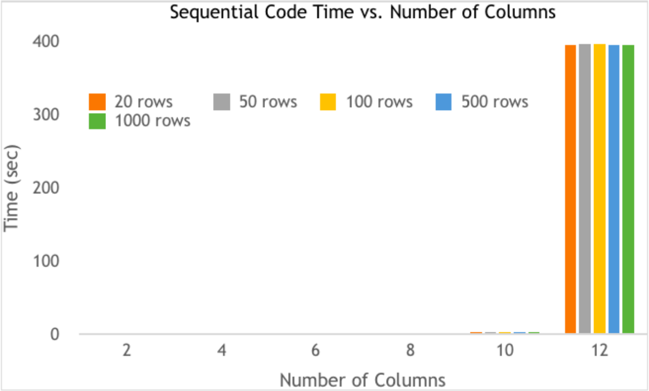

Fig. 1. Sequential code performance for logistic regression with varying number of columns and rows. Each bar represents a different value for the row input, and all row values obtained are compared with column time values.

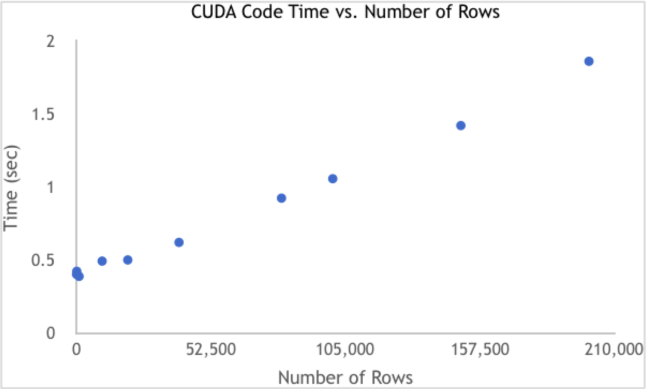

Fig. 3. Performance difference of CUDA code with a constant column value of 10.

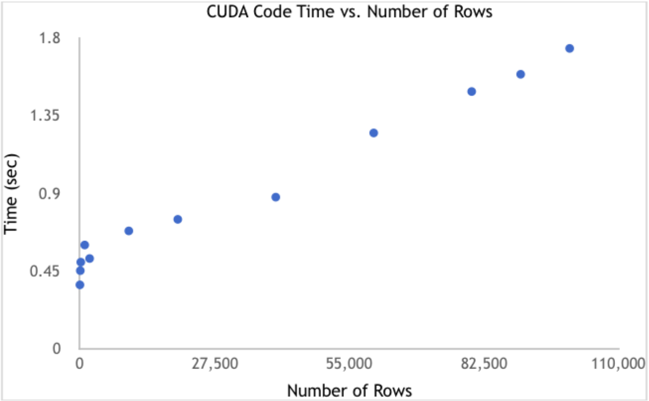

Fig. 4. Performance difference of CUDA code with a constant column value of 20.

ACKNOWLEDGMENT

Thank you Professor Zahran for all your help.

REFERENCES

[1] Bavaghar, M. Pir. ”Deforestation Modelling Using Logistic Regression and GIS.” Journal of Forest Science 61, no. No. 5 (2016): 193-99.

[2] Peng, Haoruo, Ding Liang, and Cyrus Choi. ”Evaluating Parallel Logistic Regression Models.” 2013 IEEE International Conference on Big Data, 2013.

[3] Peretti, Alessandro, and Francesco Amenta. ”Breast Cancer Prediction by Logistic Regression with CUDA Parallel Programming Support.” Breast Cancer: Current Research 01, no. 03 (2016).

[4] Salame, Camil Wadih, Joaquim Carlos Barbosa Queiroz, Gilberto De Miranda Rocha, Mario Miguel Amin, and Edson Paulino Da Rocha. ”Use of Spatial Regression Models in the Analysis of Burnings and Deforestation Occurrences in Forest Region, Amazon, Brazil.” Environmental Earth Sciences 75, no. 3 (2016).

[5] Singh, Sameer, Jeremy Kubica, Scott Larsen, and Daria Sorokina. ”Parallel Large Scale Feature Selection for Logistic Regression.” Proceedings of the 2009 SIAM International Conference on Data Mining, 2009, 1172-183.

[6] Sperandei, Sandro. ”Understanding Logistic Regression Analysis.” Biochemia Medica, 2014, 12-18.

[7] Van, Phu Nguyen, and Thophile Azomahou. ”Nonlinearities and Heterogeneity in Environmental Quality: An Empirical Analysis of Deforestation.” Journal of Development Economics 84, no. 1 (2007): 291-309.

[8] ”What Is Logistic Regression?” Statistics Solutions. Accessed December 05, 2018. https://www.statisticssolutions.com/what-is-logistic-regression/.