Multi Class Classification Logistic Regression With SGD Using OpenMp

Nilabja Bhattacharya
2018201036

Ajay Jadhav
2018201068

Ravi Hooda
2018201041

***Abstract*— Logistic regression is one of the most fundamental and widely used Machine Learning Algorithms. Logistic regression is not a regression algorithm but a probabilistic classification model.This paper proposes a novel method to optimally classify a instance to one of the many classes in the dataset. The proposed implementation is implemented in C++ using openmp for parallelising the code and validated for different datasets. The results verify the applicability of the proposed approach for the accurate prediction of the different classes in the datasets.**

***Index Terms*— SGD, OpenMp, MultiClass, Logistic Regression, One vs All logistic regression**

I. INTRODUCTION

Multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes. It is a model that is used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables (which may be real-valued, binary-valued, categorical-valued, etc.).

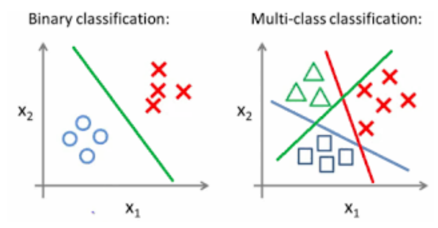

Fig. 1. Logistic Regression

II. MULTICLASS CLASSIFICATION TECHNIQUE

A. One Vs All

Now we will approach the classification of data when we have more than two categories. Instead of y = 0,1 we will expand our definition so that y = 0,1...n.

Since y = 0,1...n, we divide our problem into n+1 (+1 because the index starts at 0) binary classification problems; in each one, we predict the probability that 'y' is a member of one of our classes.

We are basically choosing one class and then lumping all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction.

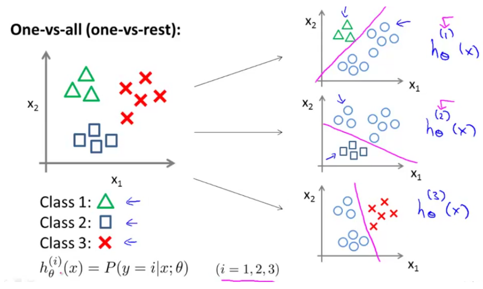

Fig. 2. One Vs All

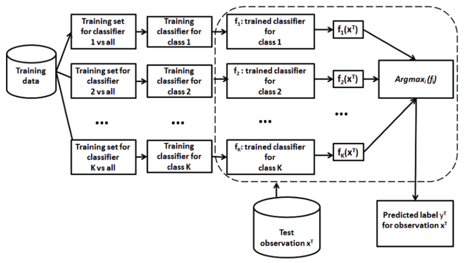
Training data
Training set for classifier 1 vs all
Training set for classifier 2 vs all
Training set for classifier K vs all
Training classifier for class 1
Training classifier for class 2
Training classifier for class K
f1: trained classifier for class 1
f2: trained classifier for class 2
fK: trained classifier for class K
f1(xT)
f2(xT)
fK(xT)
Argmaxi(fi)
Test observation xT
Predicted label yT for observation xT

Fig. 3. One Vs All Illustration

B. One Vs One

- The one vs one classification technique is about picking a pair of classes from a set of n classes and develop a binary classifier for each pair. So given n classes we can pick all possible combinations of pairs of classes from n and then for each pair we develop a binary classifier.

- For combinations we have:

$^{m}C_k = \frac{m!}{k!(m-k)!}$

and for k=2 corresponding to two pairs we have

$^{m}C_2 = \frac{m!}{2!(m-2)!}$

$^{m}C_2 = \frac{m!}{k!(m-k)!}$

$^{m}C_2 = \frac{n(n-1)}{2}$

- Among the binary classified value,one vector will go for further computation.The value which is most frequent in all values is selected. - For this data we have 10 classes ,so there are total 45 pairs of binary classifier .Among these most frequent one will be selected.

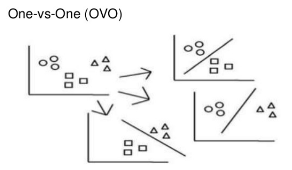

Fig. 4. One Vs One

III. Gradient Descent Algorithm

A. *Gradient Descent algorithm and its variants*

Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is basically used for updating the parameters of the learning model.

Types of gradient Descent:

- *Batch Gradient Descent:* This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.
- *Stochastic Gradient Descent:* This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large. $\theta_j = \theta_j - \alpha(\hat{y^i} - y^i)x^i_j$
- *Mini Batch gradient descent:* This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here b examples where b¡m are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

IV. The Proposed Approach

This section presents the proposed approach for classifying instances of dataset into multiple classes of the dataset.Various phases were illustrated as follows:

A. ***The phases of proposed approach are:***

- *Data Collection Phase:* This phase includes collection of different multiclass dataset on which classification is to be done.In total 5 multiclass dataset were collected and further classification is done on them.
- *Data Filtration Phase:* This phase filters data to remove all empty/redundant values.
- *Train-Test Split Phase:* This phase splits all data into training and testing data subsets. For example, data are divided into two parts per a ratio of 80% training data and 20% test data.
- *Data-Scaling Phase:* Before data are passed to the model, the data are scaled according to model requirements. In this way, this phase reshapes data to make them more suitable for the model.
- *Model-Building Phase:* The proposed approach is implemented in C++. Model is build for classifying various instances of the dataset.Probabilties are calculated for a instance for each classes.Those class which is having highest probability for a particular instance will be classified accordingly.
- *Model Learning and Evaluation Phase:* Data are trained using gradient descent algorithm.On the defined number of iteration,alpha parameter is learned over a number of iterations.
- *Parallelizing the model:* Gradient descent algorithm is parallelized.Scaling function of the instance variable is also parallelized.Various minor function are also parallelized.
- *Prediction Phase:* Prediction is made using the saved model. Input values are passed to the model to classify instance in one of the classes. Then that output is compared with testing data to calculate accuracy and losses over different number of threads.Analysis is done on the different threads.

V. Portion of code that was parallelized

A. *For stochastic gradient descent*

- Splitting the train and test data into feature and label vector using *pragma omp parallel for collapse(2)*
- Data scaling part using *pragma omp parallel for*
- Fitting n models where n is number of classes using *pragma omp parallel for*
- Stochastic Gradient Descent part using *pragma omp parallel for reduction(+:* $cross_entropy_loss$)
- Vector multiplication part using *pragma omp parallel for reduction(+:r)*
- Predicting the label for train and test set using *pragma omp parallel for*

B. *For batch gradient descent*

- Splitting the train and test data into feature and label vector using *pragma omp parallel for collapse(2)*
- Data scaling part using *pragma omp parallel for*
- Fitting n models where n is number of classes using *pragma omp parallel for*
- Batch Gradient Descent part to calculate cross_entropy_loss, update weights and update bias part using pragma omp parallel for reduction(+:r)
- Vector multiplication part using *pragma omp parallel for reduction(+:r)*
- Predicting the label for train and test set using *pragma omp parallel for*

VI. Observations

A. Dataset Details

TABLE I
DATASET DETAILS

Dataset	Train Size	Test Size	Features	Classes
Acostic	63058	15765	50	3
DNA	1600	400	180	3
E.coli	268	68	7	8
Glass	171	43	9	6
Iris	120	30	4	3
Letter	12000	3000	16	26
satimage	3548	887	36	6
seismac	63058	15765	50	32
vowel	422	106	10	11
wine	3526	882	11	7

B. Accuracy on datasets

TABLE II
ACCURACY ON DATASETS

Dataset	SGD train	SGD test	BGD train	BGD test	Sklearn train	Sklearn test
Acostic	0.68130	0.68727	0.68505	0.68772	0.68353	0.68823
DNA	0.9775	0.9425	0.98125	0.95	0.9893	0.9525
E.coli	0.8694	0.8676	0.88095	0.86764	0.7611	0.75
Glass	0.54386	0.58139	0.55555	0.58139	0.66081	0.55813
Iris	0.89166	0.96667	0.89166	0.96666	0.95	0.96667
Letter	0.66233	0.66755	0.66	0.67	0.71891	0.72433
satimage	0.82694	0.81632	0.8365	0.81172	0.84836	0.82074
seismac	0.70511	0.711098	0.7077	0.7115	0.70814	0.71252
vowel	0.4834	0.4528	0.4786	0.4528	0.6445	0.5566
wine	0.46880	0.4727	0.4671	0.4682	0.5292	0.5328

C. Cross Entropy Analysis On Dataset

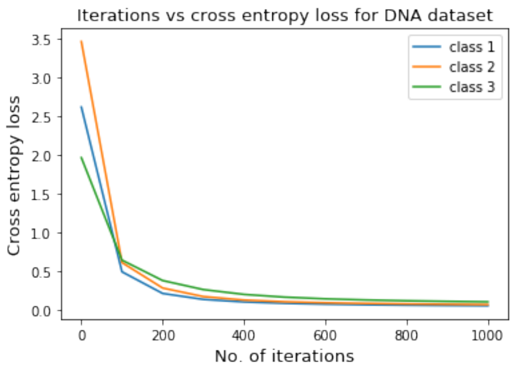

Fig. 5. Cross Entropy Loss vs No. Of Iterations for DNA dataset

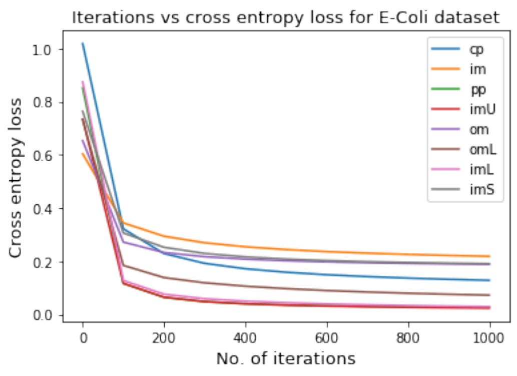

Fig. 6. Cross Entropy Loss vs No. Of Iterations for ecoli dataset

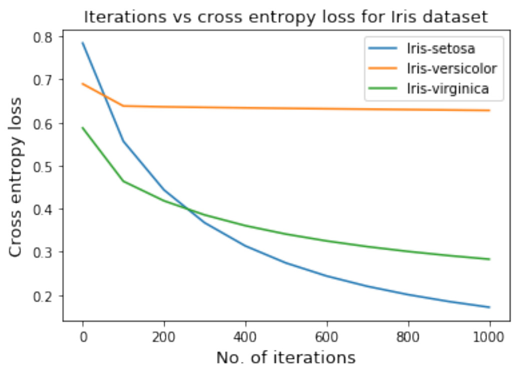

Fig. 7. Cross Entropy Loss vs No. Of Iterations for iris dataset

D. Analysis on dataset for time when run on various number of threads

TABLE III
ANALYSIS ON DATASET FOR TIME WHEN RUN ON VARIOUS NUMBER OF THREADS

Threads / Dataset	1	2	4	8
Acostic	4m 6.145s	2m 4.197s	1m 6.488s	0m 30.561s
DNA	0m 18.316s	0m 9.453s	0m 4.889s	0m 4.862s
E.coli	0m 0.675s	0m 0.343s	0m 0.184s	0m 0.114s
Iris	0m 0.109s	0m 0.068s	0m 0.041s	0m 0.037s
satimage	0m 18.262s	0m 10.580s	0m 5.496s	0m 3.416s
wine	0m 12.882s	0m 6.509s	0m 4.023s	0m 2.292s

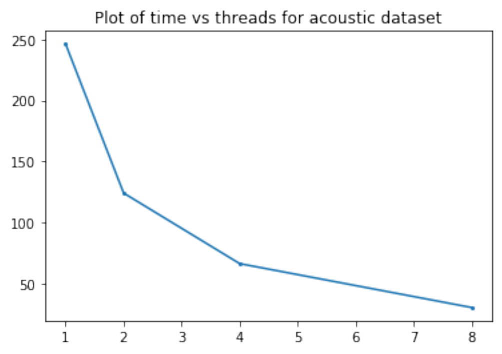

Fig. 8. Plot of time vs threads for acoustic dataset

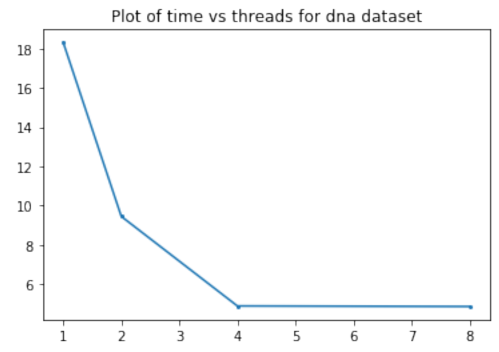

Fig. 9. Plot of time vs threads for dna dataset

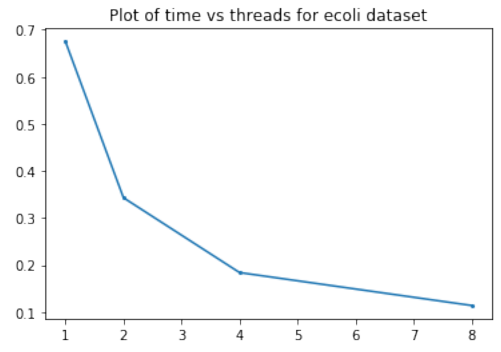

Fig. 10. Plot of time vs threads for ecoli dataset

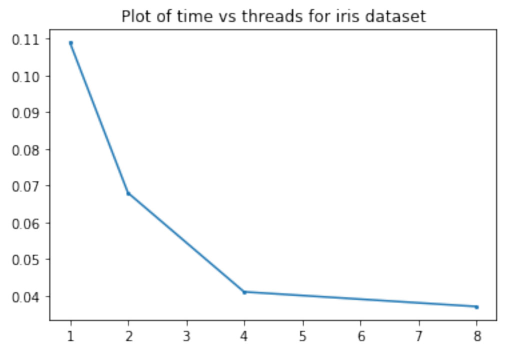

Fig. 11. Plot of time vs threads for iris dataset

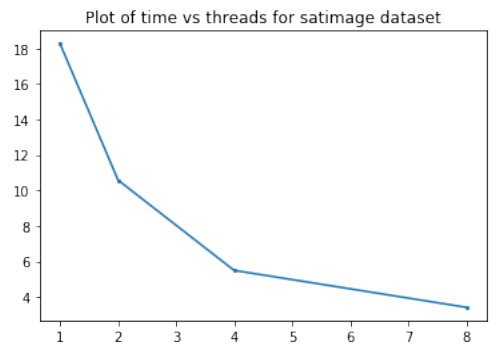

Fig. 12. Plot of time vs threads for satimage dataset

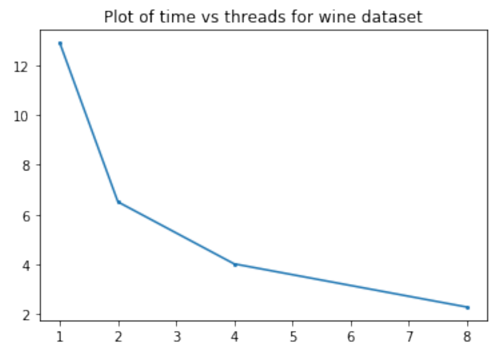

Fig. 13. Plot of time vs threads for wine dataset

VII. Conclusions

The proposed approach is implemented in C++ and is parallelized using openmp. One Vs All version of logistic regression is implemented. Performance is analyzed on different number of threads.On increasing the number of threads,performance is optimized. Also different datasets were used to analyzed the performance of the optimized algorithm. Apart from this we have also used batch gradient descent and sklearn OVR multiclass classification module to compare the performance of our code.

References

[1] LIBSVM Data
[2] SGD and BGD