MiniProject 2: Optimization and Text Classification

Abstract

In the first section of this project, we optimized a logistic regression model on the diabetes dataset. We then compared the test accuracy on the baseline set up of the model (optimized with cross validation and by testing different hyper parameters) with mini batch stochastic gradient descent (SGD) implementation and adding momentum to the gradient descent (GD) implementation. On the second part of this project, we worked on a text classification algorithm based on logistic regression to help distinguish between articles written by humans and ones generated by computer. We found that optimizing certain parameters when mapping raw text to features can get us to an accuracy of 76.9% on the test dataset.

1 Introduction

In the first part of this project, we optimized a logistic regression model on the diabetes dataset. By testing different combinations of learning rate (LR) and number of maximum iterations (max_iter), we found that model accuracy increased as number of iterations increased for fully batched GD logistic regression. The model performed the best with a LR = 0.002 and max_iter = 1e6, achieving the test accuracy of 72.06%. However, because of the random manner of mini batch SGD, a different batch size has the best accuracy on each time we run the code. in the next step, the logistic regression model integrated with the mini-batch SGD algorithm. we set the model with hyper parameters obtained from Part 1.1's best model (LR = 0.002, max_iter = 1e6). With growing batch sizes, we found out that highest validation accuracy is achieved with batch size = 256, resulting in the validation accuracy of 76.0%. The accuracy is increased in comparison with the fully batched model, but more time needed for convergence. In part 1.3, by adding momentum to the fully batched GD (the baseline model), we obtained a same validation accuracy of the baseline model $\frac{1e6}{4e4} = 25$ times faster. By doing part 1.4, we concluded that the momentum is very effective for fully batched GD and not to much helpful for the small mini batch. For the second part, we used a basic logistic regression model form SkitLearn, then experimented with stop words from NLTK library and using TFIDF features, yielding different accuracies. Lastly, by only using occurences instead of frequencies and run ngrams, max.df and min.df variations, we are able to yield test accuracy of 76.9% compared to the initial 71.1% accuracy.

2 Datasets

The diabetes dataset comes with 8 features and a single binary target variable (whether the patient has diabetes or not). The dataset is divided into 3 categories: training data (601 samples), test data (69 samples) and validation data (101 samples). We initially did not do any processing on the data. However, efforts were made to better visualize the correlation between different features (Appendix Figure 1) and we noticed some features which are not highly correlated with the outcome. An example is the 'Insulin' feature, which we removed and tested our model with (See Table 1). In addition, we see that no specific feature seems to have a very strong correlation to another.
In part 2, The unprocessed fake news dataset only includes a text corpus and a label target variable indicating whether an article is computer-generated or not. Again, the dataset is divided into 3 categories: training data (20000 samples), test data (3000 samples) and validation data (3000 samples).

Feature vectors were extracted from the documents to proceed with training. We experimented by using term frequency-inverse document frequency or TF-IDF to evaluate how relevant a word is in the entire set of texts as well as tweaking parameters of the sklearn CountVectorizer package such as the n-grams range, frequency thresholds etc. Other pre-processing methods such as the removal of samples with null values, as well as the addition of a stop words list, were experimented with.

3 Results

3.1 Optimization

We initially trained the logistic regression model using pairs of LR and max_iter stopping conditions. This approach allowed us to filter through pairs of parameters that achieved at least 70% accuracy on the validation data. Plotting the accuracy on the validation data against the number of iterations for different rates, showed us that regardless of the rates we had chosen, the model correctness generally improved with a larger number of iterations for fully batched GD setups (Appendix Figure 2 & 3). It total, we ended up with 5 candidate pairs and proceeded to run these configurations on the test data. However, all candidates yielded the same accuracy of about 72.06%. It should be noted that we also got this accuracy by training a model with the maximum number of iterations set to 1e7. This tells us that there is no need to further increase the number of iterations the weights seemed to have already converged. As a final test, 5-fold cross validation was performed. It was determined that, for the baseline implementation provided, the model performed best with a LR = 0.002 and a stopping condition set to 1e6 iterations (67.23% average accuracy) (See Table 1) These results are from a pre-processed dataset that does not include the 'Insulin' feature. We tested initially the fully loaded dataset which gave us parameters max_iters = 1e6 and LR = 0.1 but the results were not as good (Again, please see Table 1). We therefore decided to continue the rest of the experiments with a dataset which does not include the 'Insulin' feature.

Includes 'Insulin'	Train Accuracy	Test Accuracy	Validation Accuracy
Yes	63.50 %	57.35 %	63.00 %
No	69.67 %	72.06 %	74.00 %

Table 1: Baseline logistic regression model results

We then implemented the mini batch SGD. We set LR = 0.002 and max_iter = 1e6 as the models hyper-parameters, which are extracted from the best model from part 1.1. By sampling growing mini batch sizes to train the model, we are able to see that larger batch size produces higher accuracy at the cost of much slower convergence/computational speed (Appendix Figures 4 & 5), compared to the fully batched baseline implementation from part 1.1. Although mini batch SGD needs less iteratoins to converge, it takes more time than GD because the gradient calculation is more sequential (we have a for loop over data batches). We saw that the validation accuracy of 76.0% achieved with batch size = 256. In part 1.3, by adding momentum to the fully batched GD (the baseline model), we obtained a better training and validation accuracy (77.83% and 75.0%, respectively) for LR = 0.002 and max_iter=1e6 (Appendix Figure 6) by choosing $\beta = 0.95$. Moreover, we achieved the same validation accuracy of the baseline model $\frac{1e6\ iterations}{4e4\ iterations} = 25$ times faster (Appendix Figure 8). The computation time for GD with momentum is lower than the GD without momentum ($\beta = 0$) based on the Appendix Figures 7 and 9. On part 1.4, the smallest batch size is 8 and the largest is 256. Considering figures 6 to 11, we conclude that the momentum is very effective for fully batched GD and not too much helpful for the small mini batch.

3.2 Text Classification

We began by training a basic logistic regression model from sklearn (i.e., default parameters) to find out an initial test accuracy on the fakenews dataset. The model yielded an accuracy of 71.1% on the test data but only 43.15% on the validation data (See Table 2). With these values in mind, we ran different experiments.

Train Accuracy	Test Accuracy	Validation Accuracy
85.52 %	71.10 %	43.15 %

Table 2: Baseline results for text classification

The first experiment conducted involved using a list of english stop-words from the NLTK library. Stop words are not very informative so we wish to ignore them before we build feature vectors. The results are presented in Table 3. We see that, while the accuracy on the validation dataset was improved, test accuracy was lowered. In addition, we were lead to believe that using stop-words might not be the best approach as computer-generated text could have patterns that are recognizable thanks to these "basic" words. We therefore wish to keep them in the analysis. Next, we tested

Train Accuracy	Test Accuracy	Validation Accuracy
86.21 %	69.83 %	59.25 %

Table 3: NLTK stop-words implementation for text classification

the max_df and min_df parameters from the CountVectorizer object which allows us to remove terms that are in more or less than a certain percentage of the documents. We found the best combination of max_df and min_df parameters to be 0.6 and 0 respectively yielding a test accuracy of 71.17% (barely improving upon the baseline model) and a validation accuracy of 65.85%. We then looked at the ngrams parameter which allows us to specify the minimum and maximum length of a sequence of words to extract from the texts. We noticed from that experiment that only extracting unigrams (i.e., set ngrams = (1, 1)) gives us the best accuracy of 71.17% out of 2 other possible set of min/max values. So far, our experiments were conducted by using TFIDF features. So, We decided to only use occurences instead of frequencies and ran similar experiments (ngrams, max_df and min_df variations). We were able to get a test accuracy of 76.9% which greatly improves upon the previous results.

4 Discussion and Conclusion

Through training and testing various logistic regression implementations with different hyper parameters, we found the best model with highest accuracies and saw the correlation between increase in test accuracy and convergence speed and decrease in mini batch size. By adding momentum to gradient descent, it helps accelerate gradients vectors in the right directions. Through usage of libraries and different implementations of hyperparameters and features for text classification, we are able to discover a best model and improve test accuracy to 76.9%.

5 Statement of Contributions

Mehdi Ammar: Part 1.1 (optimize logistic regression) and Part 2 (text classification)
Hamed Jafarzadeh Asl: Part 1.3, 1.4 (adding momentum to gradient descent, analysis of impact
Ying Zhou: Part 1.2 (implementing mini-batch stochastic gradient descent and comparison)

Appendices

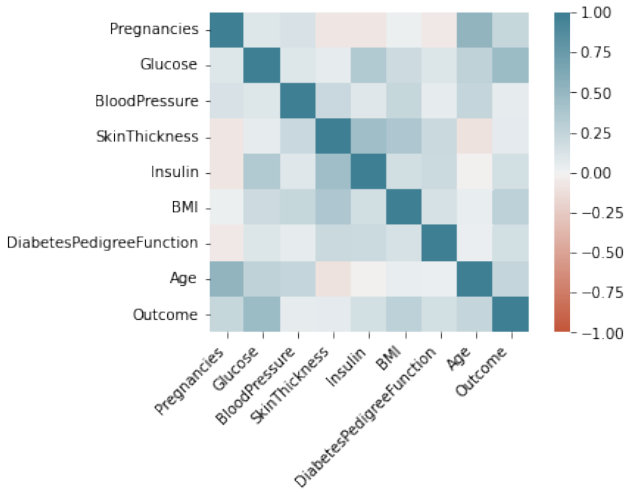

Figure 1: Correlation matrix for the diabetes dataset

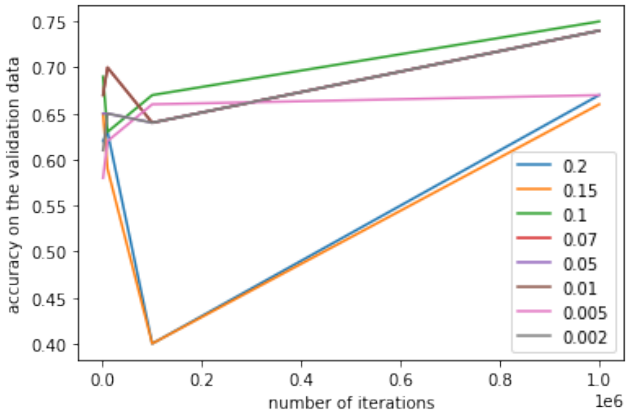

Figure 2: Validation data accuracy against the maximum number of iterations for various learning rates

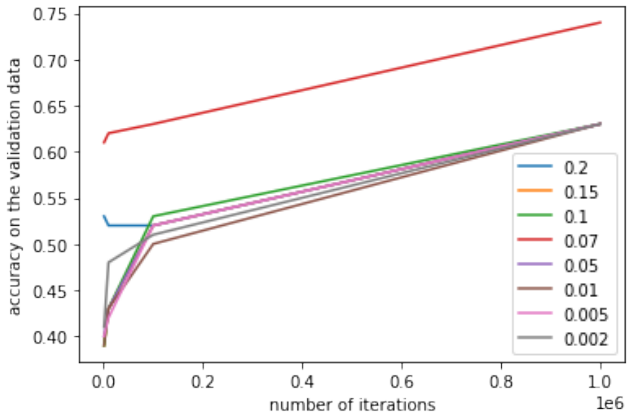

Figure 3: Validation data accuracy against the maximum number of iterations for various learning rates (Including the 'Insulin' feature)

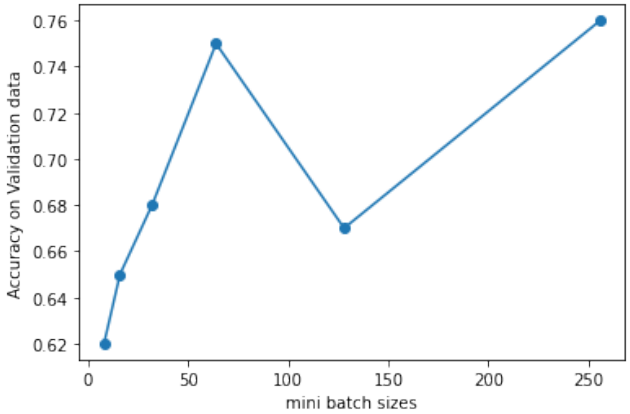

Figure 4: Accuracy on Validation data versus mini batch sizes - mini batch SGD

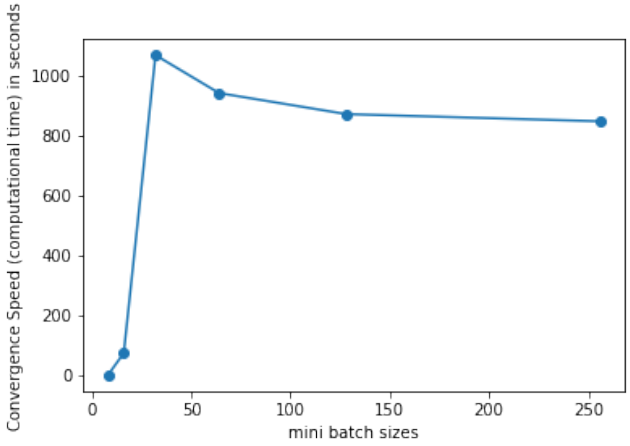

Figure 5: Convergence speed (computational time) of mini batch SGD model for various mini batch sizes

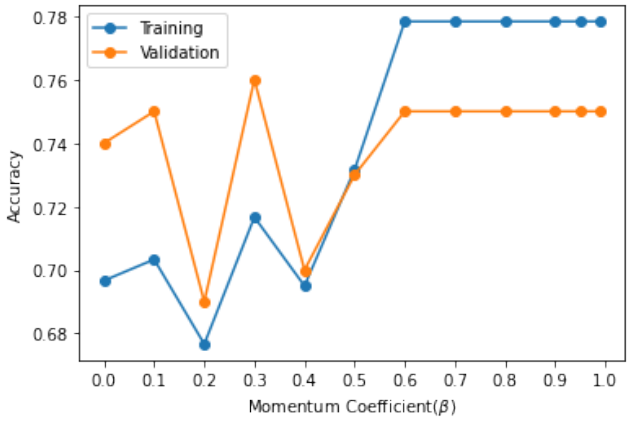

Figure 6: Accuracy on Training and Validation data versus Momentum Coefficient (β) - GD with Momentum [max_iter=1e6]

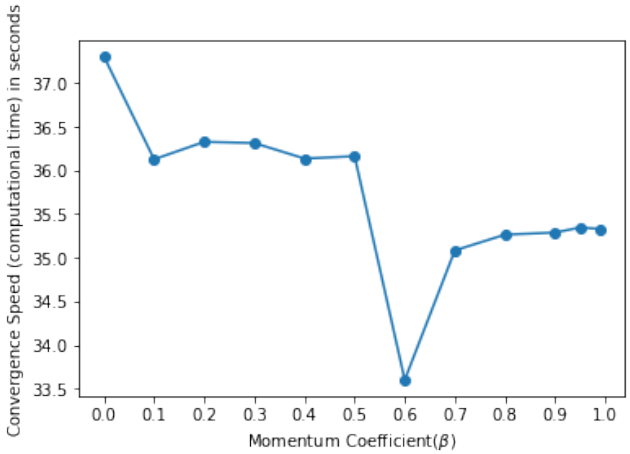

Figure 7: Convergence speed (computational time) of GD with Momentum model for various Momentum Coefficients (β) [max_iter=1e6]

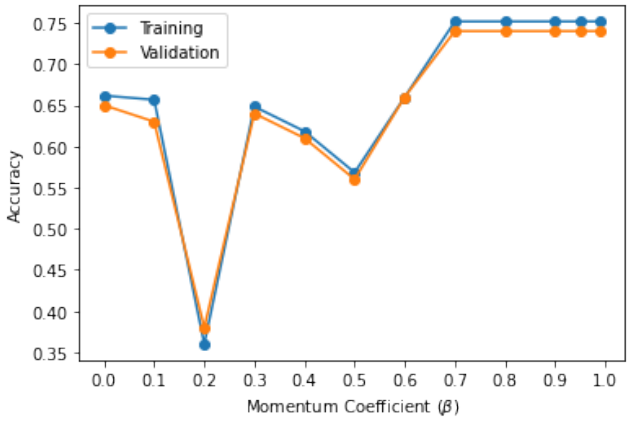

Figure 8: Accuracy on Training and Validation data versus Momentum Coefficient (β) - GD with Momentum [max_iter=5e4]

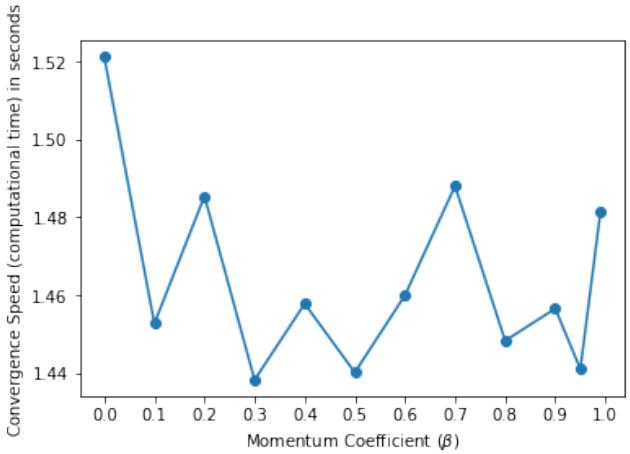

Figure 9: Convergence speed (computational time) of GD with Momentum model for various Momentum Coefficients (β) [max_iter=5e4]

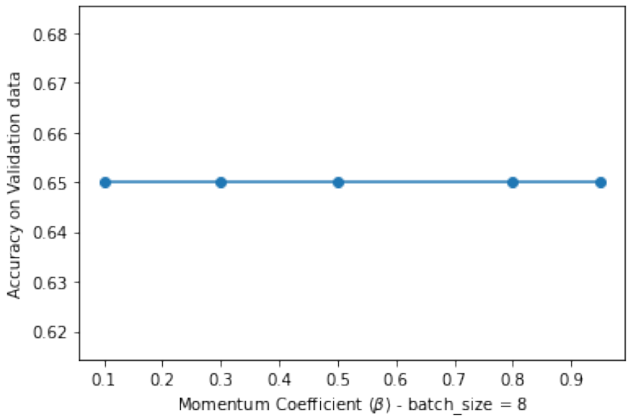

Figure 10: Accuracy on Validation data versus Momentum Coefficient (β) - mini batch SGD with Momentum [batch size = 8 & max_iter=1e6]

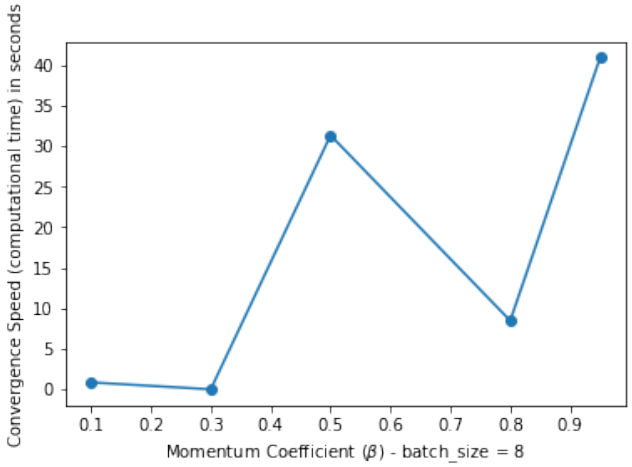

Figure 11: Convergence speed (computational time) of mini batch SGD with Momentum model for various Momentum Coefficients (β) [batch size = 8 & max_iter=1e6]

s