COMP 551 – Applied Machine Learning **Fall 2021**

Mini Project 2 – Optimization and Text Classification Group 68

Junjian Chen 260909101

Yutian Fu 260789775

Zhekai Jiang 260906711

Abstract

In this project, we first implemented the regular gradient descent (GD) based logistic regression to find an appropriate learning rate and number of iterations. Then, we developed three variants of GD, with minibatch, momentum, and the two combined, to compare their performance and analyze the impacts of the values of parameters on accuracy and speed of convergence. For the minibatch stochastic gradient descent (SGD), we found that under a fixed number of learning rate and training epochs, the smaller the batch size, the higher the accuracy but the lower the convergence speed. For gradient descent with momentum, the value of momentum does not have a large impact on the convergence speed and accuracy in our case. Minibatch SGD with momentum generates the highest accuracy overall. For the second part, we used logistic regression based on term frequency–inverse document frequency (TF-IDF) to classify whether an article is written by a human or generated by a computer, and we reached an overall accuracy over 80.1% on the testing dataset.

1 Introduction

In Part 1, our main purpose is to compare the performance of four gradient descent methods and the effects of several hyperparameters on the convergence speed and accuracy, using the Diabetes Dataset. We decided on 0.0001 as the learning rate throughout Part 1, which leads to a validation accuracy of 67.16% using the regular gradient descent. With this baseline, we observed that the minibatch SGD converges much slower but yields a higher accuracy 71.20% with batch size 16. Momentum alone does not have a significant impact compared to the baseline. But, after adding momentum to minibatch SGD, we achieved the highest overall accuracy 73.38% in one configuration.

In Part 2, the objective is to train a model to classify whether an article is written by a human or generated by a computer, based on the Fake News Dataset. We vectorized the textual data based on TF-IDF, leveraged logistic regression, and finally reached an accuracy of 80.1%.

2 Datasets

2.1 Diabetes Dataset

The diabetes dataset has 8 features, including BMI, glucose, age, etc., of each instance. The dataset has two labels indicating whether one has diabete or not. There are 600 instances for training, 100 instances for validation and 68 instances for testing. As we are instructed to use the dataset as-is, we did not apply preprocessing to the dataset and directly loaded the data.

2.2 Fake News Dataset

The fake news dataset contains textual news either written by a human or generated by a computer. The dataset has two labels indicating whether the new is fake. There are 20000 instances for training, 2000 instances for validation, and 3000 instances for testing. Then, we used the `Pipeline` class in `sklearn` to preprocess the textual data. A `Pipeline` consists of `CountVectorizer()`, `TfidfTransformer()` and `LogisticRegression()`. The former two are used to preprocess textual data and convert them into numerical features; the latter one is the prediction model. We then tuned the hyperparameters of the three components to optimize the performance.

3 Results

3.1 Part 1

3.1.1 Regular Gradient Descent

We experimented with several values of learning rate and a maximum number of iteration of 100000. The model with learning rate 0.0001 converges in less than 20000 iterations and it gives a validation accuracy of 67.16%. Figure 1 shows the change of training and validation accuracy as the iterations proceed. Figures 9 to 11 in Appendix A similarly show the performance resulting from a few other parameters we tried. For the remainder of Part 1, we will use 0.0001 as the learning rate and 100000 as the maximum number of iterations.

3.1.2 Stochastic Minibatch Gradient Descent

We experimented with batch sizes 16, 64, and 128. From Figures 2 to 4, we can observe that small batch sizes make convergence slower—none of the three configurations generated a stable accuracy after 100000 iterations. Among the three we tried, the model with the smallest batch size 16 has the highest validation accuracy 71.20%, which is higher than the regular version.

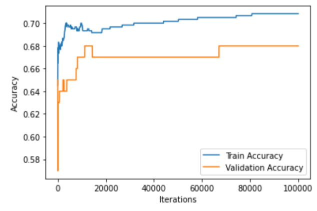

Figure 1: GD, learning rate = 0.0001

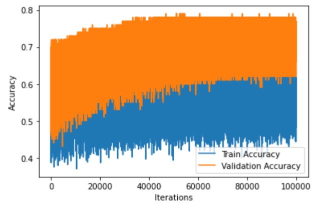

Figure 2: SGD, batch size = 16

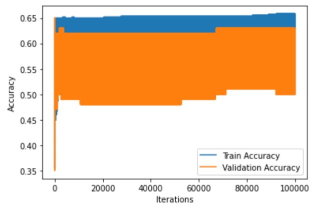

Figure 3: SGD, batch size = 64

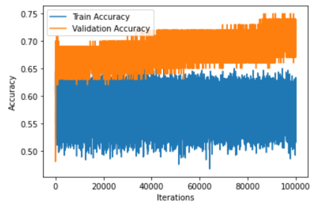

Figure 4: SGD, batch size = 128

3.1.3 Gradient Descent with Momentum

During our experiments with momentum, we found that variations of momentum do not significantly affect the accuracy and convergence speed compared to the baseline—all three configurations we tried generated a validation accuracy of approximately 67.16% and converged in approximately 17500 iterations, as shown in Figures 12 to 14 in Appendix B.

3.1.4 Minibatch Stochastic Gradient Descent with Momentum

From our data, we observed that adding momentum to Minibatch SGD could accelerate convergence compared to SGD without momentum. Figures 5 to 8, as well as Figures 15 to 19 in Appendix C, show some of the results we obtained, among which a batch size of 16 and momentum of 0.9 gives the best model overall with an accuracy of 73.38% after 100000 iterations.

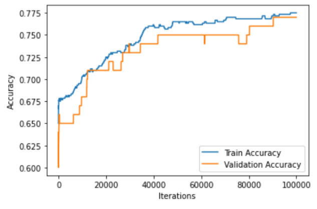

Figure 5: SGDM, batch size = 16, momentum = 0.9

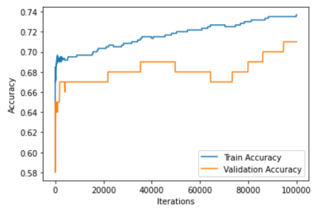

Figure 6: SGDM, batch size = 128, momentum = 0.9

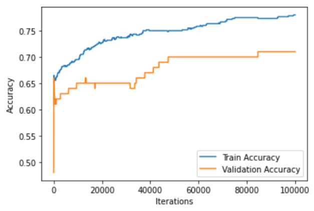

Figure 7: SGDM, batch size = 16, momentum = 0.3

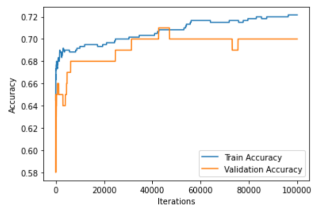

Figure 8: SGDM, batch size = 128, momentum = 0.3

3.2 Part 2

With the default parameters, the accuracy of prediction on validation set and testing set was 73.1% and 71% respectively. We made the following configurations that contributed remarkably to the accuracy, along with a few others that contributed relatively little. When optimizing with `CountVectorizer`, setting `binary` to `true`, `lowercase` to `false`, and `ngram_range` to `(1,2)` will boost the accuracy on prediction in validation set by about 3%. Moreover, setting `class_weight` to `balanced` and increasing `C` of `LogisticRegression`, which stands for inverse of regularization strength, will also improve accuracy. Finally, among all trials, the best testing accuracy we reached was 80.1%. Detailed results of all configurations we tested are listed in Appendix D.

4 Discussion and Conclusion

4.1 Part 1

For gradient descent, we first appreciated the impact of the learning rate. Our experiments showed that a learning rate that is too large or too small could lead to slow convergence and low accuracy. Minibatch SGD could lead to a higher accuracy with a small batch size, but it converges significantly more slowly. This could be explained by oscillations caused by the noise because of the smaller dataset involved in each step. A potential way to improve the accuracy is to add an implicit regularizer to the original loss function, which penalizes the norms of gradients, as shown in [1].

We discovered that the introduction of momentum alone does not have a significant impact compared to the baseline. We hypothesize that it is because gradient descent will undergo similar paths due to the smoothness of our dataset, the small learning rate, and the inclusion of historical values. When momentum is applied with minibatch, however, we obtained the highest accuracy among all configurations while also faster convergence compared to the minibatch SGD alone. A probable reason is that momentum will always tend to add common portions in the same direction and can overcome the oscillations of noisy gradients across flat spots of the search space.

4.2 Part 2

In Part 2, we tried to optimize the performance by modifying the parameters of the data processor (`CountVectorizer`) and the model (`LogitsicRegression`).

For the model, we found that increasing the inverse of regularization strength (`C`) from 1 to 5 improves accuracy significantly. However, when `C` is above 5, the accuracy almost stops to increase, likely due to overfitting caused by a low regularization.

For the data processor, we tuned `binary`, `lowercase`, and `ngram_range`, as well as a few other parameters. We found it useful to set `binary` to `true`. Also, setting `lowercase` to `false`, which means keeping the capitalization pattern of the orignal text, is shown to be helpful as well, likely because of the different behaviour of humans and machines in terms of capitalization. Furthermore, setting `ngram_range` to `(1, 2)` applies combination of unigrams and bigrams so that each sentence will be divided into parts of one word and parts of two words.

For further improvements, due to the relevance of the experiments to human language, Natural Language Processing (NLP) may be useful. As shown in [2], the NLP and deep learning algorithms could be beneficial for detecting fake news. The future work could be done in COMP 550.

Statement of Contributions

Junjian Chen: Implementation of Part 1 and Part 2, hyperparameter tuning, testing, write-up
Yutian Fu: Data visualization, testing, write-up
Zhekai Jiang: Implementation of Part 1, hyperparameter tuning, testing, write-up

References

[1] S. L. Smith, B. Dherin, D. G. T. Barrett, and S. De. "On the origin of implicit regularization in stochastic gradient descent," in *International Conference on Learning Representations*, 2021.

[2] S. Girgis, E. Amer, and M. Gadallah, "Deep learning algorithms for detecting fake news in online text," in *13th International Conference on Computer Engineering and Systems (ICCES)*, 2018, pp. 93-97, doi: 10.1109/ICCES.2018.8639198.

Appendices

Appendix A: Performance of Regular Gradient Descent with Other Values of Learning Rate

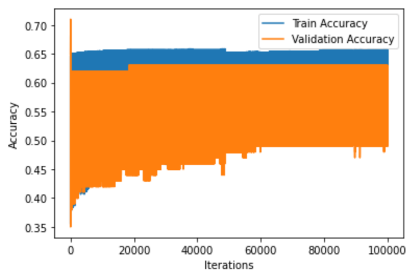

Figure 9: GD, learning rate = 0.1

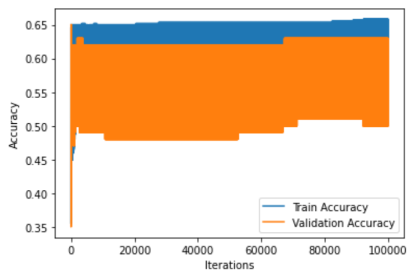

Figure 10: GD, learning rate = 0.001

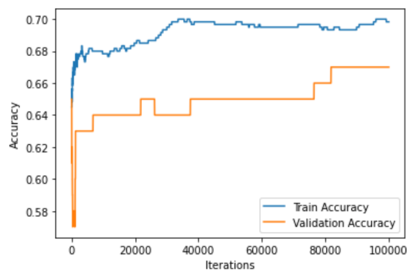

Figure 11: GD, learning rate = 0.00001

Appendix B: Performance of Gradient Descent with Momentum with Different Values of Momentum

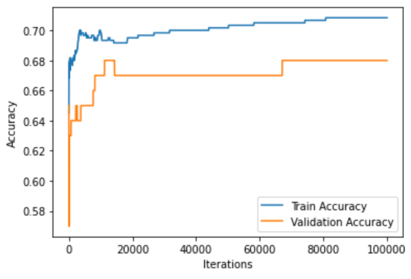

Figure 12: GDM, momentum = 0.9

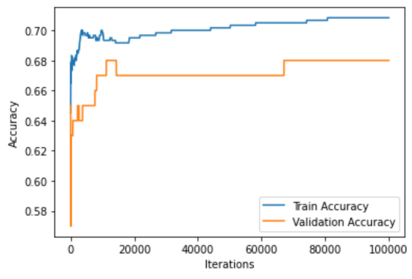

Figure 13: GDM, momentum = 0.6

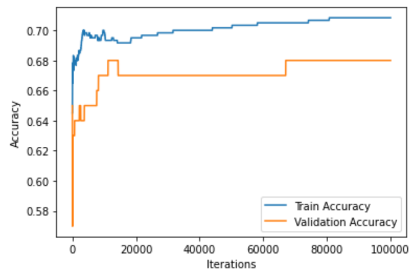

Figure 14: GDM, momentum = 0.3

Appendix C: Performance of Minibatch Stochastic Gradient Descent with Momentum with Other Values of Batch Size and Momentum

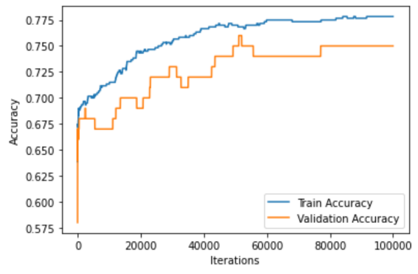

Figure 15: SGDM,
batch size = 16, momentum = 0.6

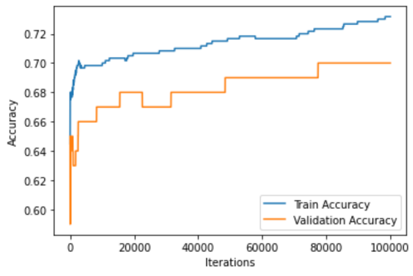

Figure 16: SGDM,
batch size = 128, momentum = 0.6

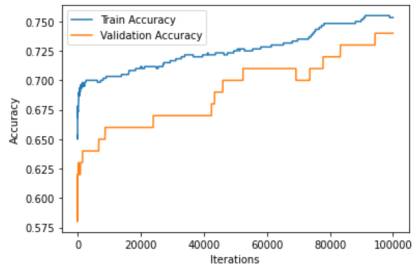

Figure 17: SGDM,
batch size = 64, momentum = 0.3

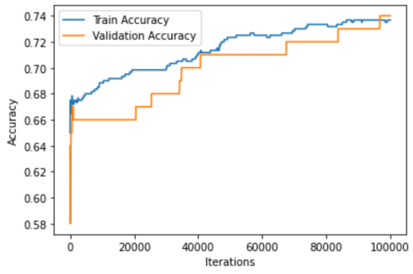

Figure 18: SGDM,
batch size = 64, momentum = 0.6

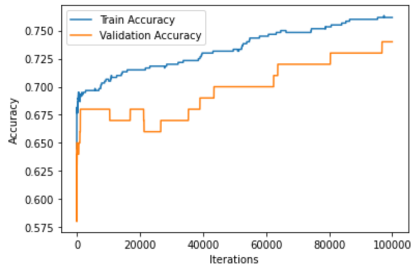

Figure 19: SGDM,
batch size = 64, momentum = 0.9

Appendix D: Performance of Text Classification with Different Configurations

Note: The unmentioned parameters or components of Pipeline are set as default.

Parameter	Validation Accuracy	Testing Accuracy
Default	0.731	0.71
`CountVectorizer(binary=True)`	0.763	0.742
`CountVectorizer(lowercase=False)`	0.729	0.724
`CountVectorizer(binary=True,lowercase=False)`	0.765	0.755
`LogisticRegression(C=3)`	0.751	0.735
`LogisticRegression(C=5)`	0.758	0.739
`LogisticRegression(C=7)`	0.751	0.735
`LogisticRegression(C=10)`	0.751	0.735
`LogisticRegression(C=3)` `CountVectorizer(binary=True,lowercase=False)`	0.781	0.771
`LogisticRegression(C=3)` `CountVectorizer(binary=True,lowercase=False, ngram_range=(1,2))`	0.783	0.772
`LogisticRegression(C=5)` `CountVectorizer(binary=True,lowercase=False, ngram_range=(1,2))`	0.782	0.781
`LogisticRegression(C=10)` `CountVectorizer(binary=True,lowercase=False, ngram_range=(1,2))`	0.787	0.791
`LogisticRegression(C=3,class_weight='balanced')` `CountVectorizer(binary=True,lowercase=False)`	0.782	0.769
`LogisticRegression(C=3,multi_class='multinomial')` `CountVectorizer(binary=True,lowercase=False)`	0.782	0.769
`LogisticRegression(C=80,class_weight='balanced', multi_class='multinomial')` `CountVectorizer(binary=True,lowercase=False, ngram_range=(1,2))`	0.795	0.801