Comp551 Mini Project 2

Report

Group18

Haochen Liu(260917834)

Yiran Fu(260904135)

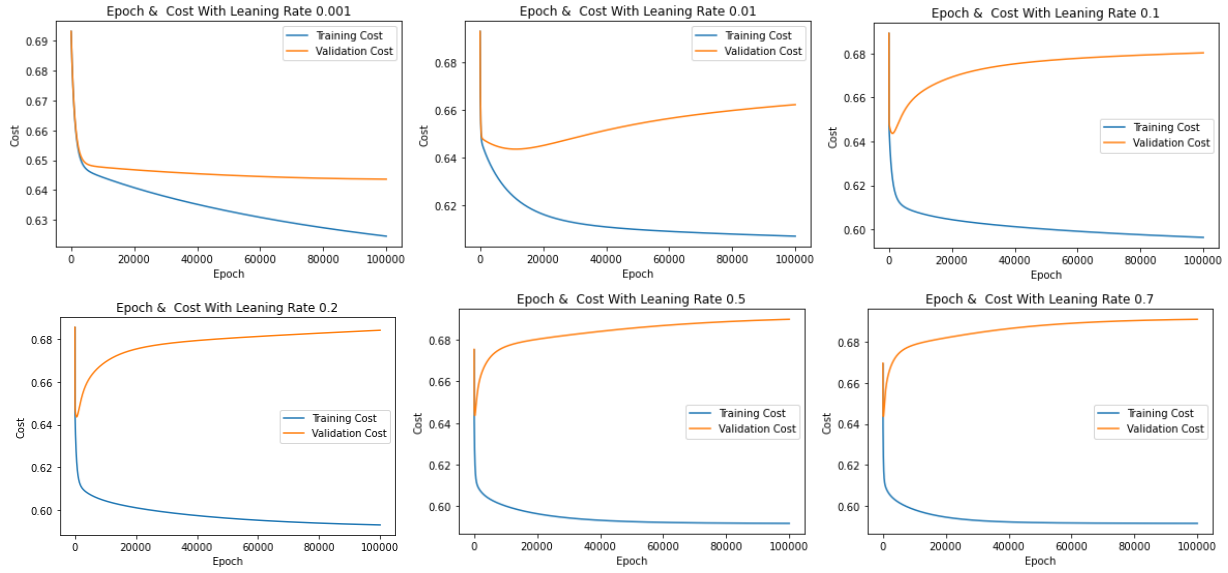
Ye Yuan(260921269)

Abstract
In this project, we did two separate tasks: one is the optimization process of gradient-based methods, and the other is about binary text classification. In the first part, by using the diabetes dataset, we utilized the logistic regression model to investigate the convergence speed and the quality of final results of regular gradient descent, mini-batch stochastic gradient descent, and gradient descent with momentum. As a result, a faster convergence speed will be achieved by using a higher learning rate or a smaller batch size. The bigger the momentum coefficient, the smaller oscillations the training and validation cost curve will have. In the second part, we did binary text classification in the faked news dataset to detect whether the news is generated by computers or not. We tried different combinations of text preprocessing methods and different machine learning models to determine the most suitable configuration. The mixture of unigram and bigram performs best in this case.

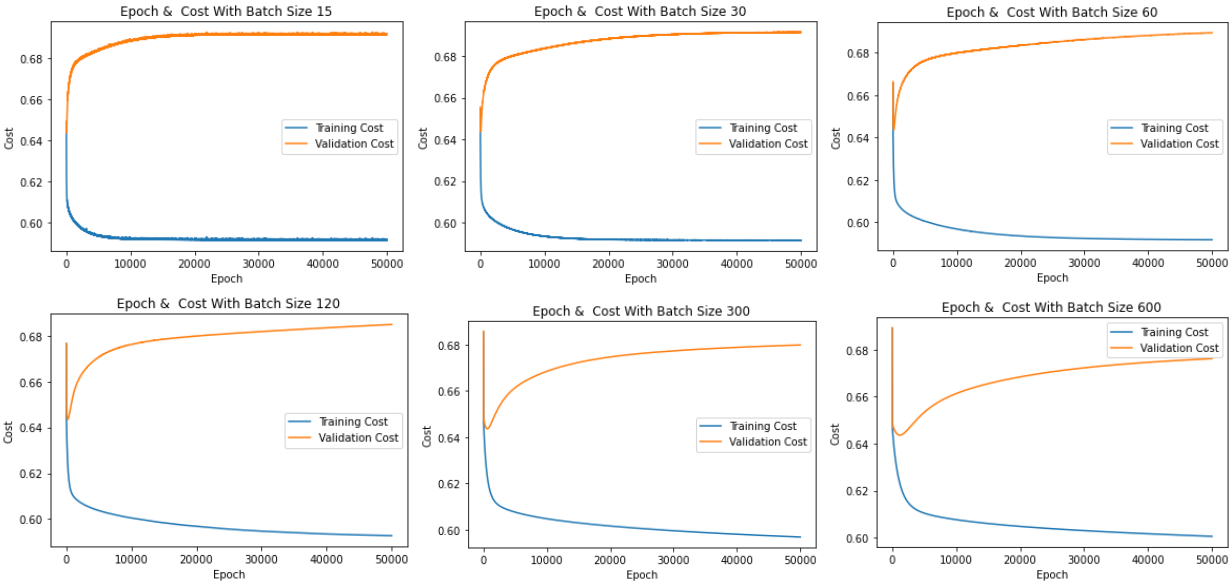
Introduction
We employed the logistic regression model to predict if a person has diabetes in the first portion of this project. We compared the convergence speed and quality of different types of gradient descent, such as regular gradient descent, mini-batch stochastic gradient descent, and gradient descent with momentum. When it comes to the definition of convergence, a model is said to be converged when its cost curve has nearly flattened out and further training will not significantly increase performance. Meanwhile, we exploited the cost of the validation set as the quality of the final solution.
In the second part of this project, we also applied the logistic regression model to do binary text classification. Text-specific preprocessing is an essential step in this part, and various preprocessing functions from the Natural Language Toolkit (NLTK) package played an important role here. We attempted numerous typical NLP preprocessing approaches, such as stemming, lemmatization, n-grams, digits removal, and punctuation removal. The combination of unigram and bigrams improved the performance the most in this faked news dataset.

Datasets
For part 1 of this project, we used the diabetes datasets with 8 features including pregnancies, glucose, blood pressure, etc. In the beginning, we used the data as-is to train the model with the given gradient descent method but found that the norm of gradient was still very high(up to 60) even after 100000 iterations. Thus, we decided to preprocess data by using the 'l2' normalizer and got considerably better results.
In the second part, we used the faked news dataset, which contains two sorts of articles: those generated by computers and those written by people. In the preprocessing steps, we incorporated a variety of regularly used text-specific preprocessing techniques, including Stemming, Lemmatization, n-grams, digits removal, and punctuations removal, to enhance the performance of our model.

Results
In part 1, we first ran the given code of logistic regression model using the normalized data to find a learning rate and the number of training iterations which the model has fully converged. Here we tuned the learning rates 0.001,0.01,0.1-0.9 with interval 0.1 and used the default max iteration 100000 as in the given code. We found that when we used a very small learning rate of 0.001, the training cost was still decreasing evidently after 100000 iterations. But for a large learning rate such as 0.7, the training cost remained almost the same only after 20000 iterations. This shows that as we increased the learning rate, the time for the model to converge became shorter. Considering both the convergence speed and the quality of the final result, we decided to choose the configuration of learning rate 0.1 and 50000 iterations in the rest of our project, which can guarantee model convergence with an acceptable quality of final results.



In the second step, we fixed the learning rate at 0.1 and tuned the minibatch sizes(15, 30, 60, 120, 300). We found that smaller batch sizes lead to faster convergence speed, but they also caused bigger oscillations. The model with a larger batch size would take more epochs to converge, but the minimal validation costs of all minibatch sizes were almost same. We thought the model worked best when minibatch size=60 because of the relatively rapid convergence speed and small oscillations that do not contribute to overshoot.



In the third step, the momentum coefficient we tried are 0.5,0.67, 0.75, 0.8,0.9,0.99. Compared to regular gradient descent, theoretically, adding momentum can boost the convergence speed while maintaining the quality of the final solution as good as before. However, as illustrated in the appendix, the benefit of using momentum with full batch size was not apparent in our experiment. Different values of momentum didn't make noticeable changes in convergence speed. Moreover, the shapes of the curves of training and validation cost were comparable.

In the last step, we repeated step 3 by using the smallest batch size 15 and the largest batch size 300. Comparing the plots of batch size 15(illustrated in the appendix), as we increased the value of

momentum, the amplitudes of oscillations are getting smaller. Nonetheless, as in the previous step, the convergence speed after adding momentum is not improved a lot. Because the oscillations are still visible for batch size=15, we believe that minibatch size 300 with momentum 0.9 is the most effective in our settings with the learning rate of 0.1 due to its smoother curve with relative fast convergence time.

In part 2, we first built a CountVectorizer => TfidTransformer => classifier pipeline class in the basic version. We used the LogisticRegressionCV（CV for automatic cross validation）in sklearn as our classifier and experimented with various text-preprocessing approaches in order to increase accuracy. The techniques we used include porter-stemmer, n-grams, word net lemmatizer, and digits and punctuations removal. The following table shows our results.

Techniques	None	Remove punctuations	Remove digits	Stemmer	lemmatizer	Ngram_range =(1,2)
Validation Accuracy	0.7505	0.7520	0.7420	0.6525	0.7315	0.787

Apart from these text preprocessing techniques, by the no-free-lunch theorem, we assume that other models may perform better than the logistic regression model. We used MLPClassifier, which is a neural network model from sklearn, but got similar results as in logistic regression. Among all settings we tried, the combination of unigram and bigram gave the best accuracy of 0.787 in the validation set. After applying to the test set, it remained a good performance with test accuracy of 0.777.

Conclusion and Discussion

In part 1 first step, we did regular gradient descent and found that as we increased the learning rate, the number of epochs for the model to converge decreased. In the second step, we did minibatch stochastic gradient descent with different batch sizes. We explored that choosing a small batch size is a good way to speed up convergence, but smaller sizes lead to bigger oscillations. In the third and fourth steps, adding momentum to gradient descent can assist to reduce the oscillations. In general, the larger values of momentum will be more helpful than the smaller ones, so we always pick the value of momentum around 0.9.

In the second part, we tried different text preprocessing techniques. However, most of these strategies are ineffective in improving our accuracy. Among all these methods, using the mix of unigram and bigram had the best performance. We also explored the performance of another machine learning model, the MLPClassifier, but the result was identical to logistic regression. In general, it is difficult to say which preprocessing technique or model would always perform better. The techniques with lower performance in this dataset may be useful in other cases. As a result, a case-to-case experiment is needed to find the most fitted preprocessing techniques and model for a specific dataset.

Statement of contributions

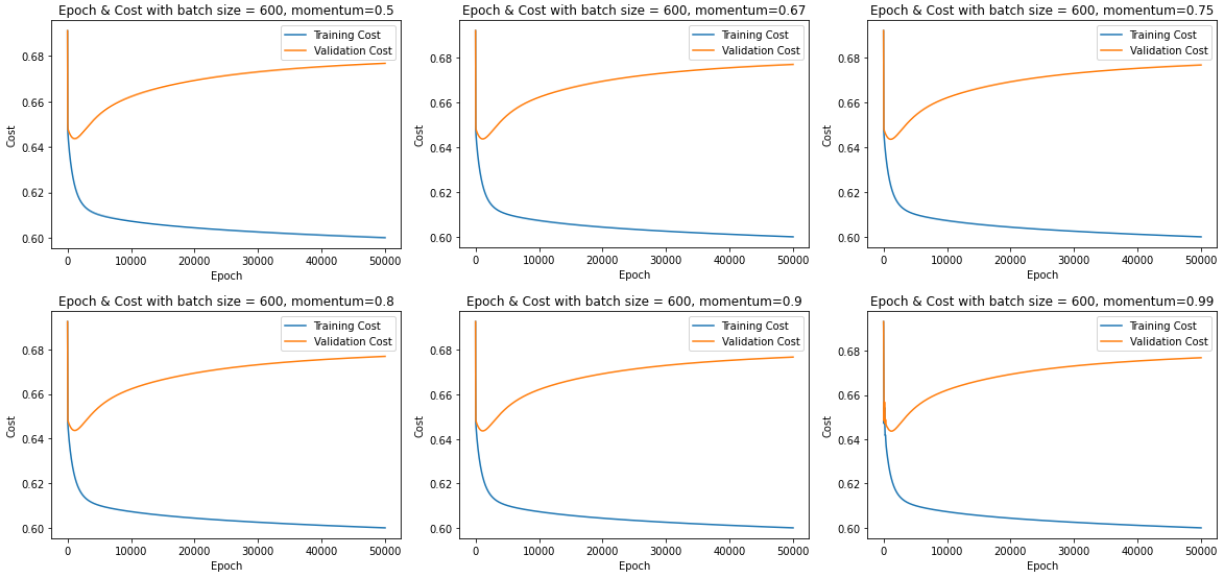
Ye Yuan: Optimized the mini batch stochastic gradient descent. Scratched the SGD with momentum.
Haochen Liu: Scratched the mini batch stochastic gradient descent. Optimized the SGD with momentum.
Yiran Fu: Independently completed the text classification part

Citation

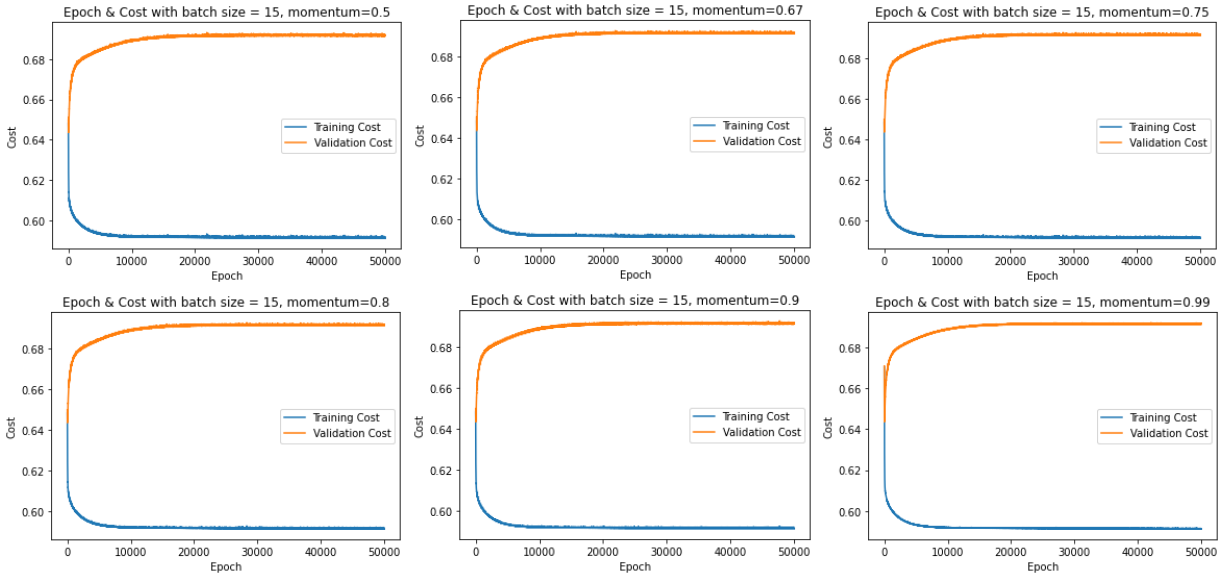
Real Python. “Stochastic Gradient Descent Algorithm with Python and NumPy.” Real Python, Real Python, 19 Jan. 2021, https://realpython.com/gradient-descent-algorithm-python/.

Appendix

Diagrams for Part1 Step3



Diagrams for Part1 Step4



Diagrams for Part1 Step4