Optimization and Text Classification

Byron Chen, Jiayang Lyu, Kaiwen Xu

Abstract

In this project we investigate the performance of logistic regression model with gradient descent optimization method. We work on the "Diabetes" dataset, where we predict whether a person has diabetes based on various factors like age and BMI. We find that the different variations of gradient descent (stochastic, mini-batch, and full-batch) can all reach the best accuracy, but their convergence speed differs significantly. In addition, we explore methods to extract features from text data through the "fake news" dataset. We also train a classifier using gradient descent to help detect fake news.

1 Part1: Optimization

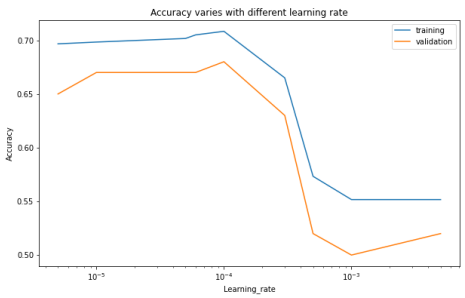
Inspired by Keshav Dhandhania [1], who gives a quick example on training logistic regression model to classify people with and without diabetes, we want to do it ourselves and apply more advanced optimization technologies like mini-batch SGD and momentum to see if we can achieve better accuracy.

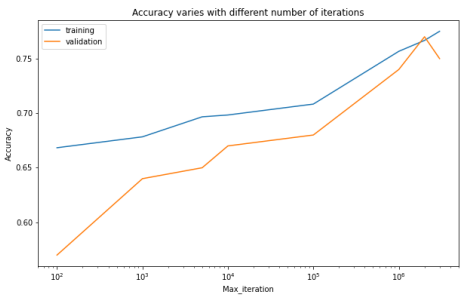
1.1 Dataset: diabetes

"Diabetes" dataset includes 600 instances with features like age, BMI and blood pressure as well as a label "outcome" indicting whether the person has diabetes. The dataset is already preprocessed and split into training, validation and test sets, so there's not much we can do.

1.2 Fully convergence parameter

First, we want to find a learning rate and a number of training iterations such that the model has fully converged to a solution. To do so, we have to formally define convergence. A model is said to converge if further training does not significantly improve its performance. In our context, performance is measured by validation accuracy. After applying grid search, we find accuracy is optimized when learning rates = 0.0001. Fixing the learning rate, we do another grid search over max_iteration, and find that at max_iteration=2e6, we get the best validation accuracy 0.76. Thus when (learning_rate, max_iteration) = (0.0001, 2e6), the model fully converges.





1.3 Implement mini-batch SGD

In this part, we implement and explore the mini-batch SGD. We compare the convergence speed and quality of the final solution on growing batch sizes. Batch size of 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 600 are chosen and our experiment ranges from one single batch to a full batch scale for better comparison. To make the experiment consistent, we reuse the same configurations as we got in part 2.2 with learning rate = 0.0001. In addition, to make sure the final solution converges within the maximum iterations, we first try use epsilon = 0.01 and max iterations = 2e6, but it is found hard for all solutions to converge and the running time is extremely high.

$$J(w) = \sum_n y^{(n)} \log(1 + e^{-xw^\top}) + (1 - y^{(n)}) \log(1 + e^{xw^\top})$$

To fix this, we introduce a new configuration called err_ specified. By calculating the loss function (shown above), we could have the loss value in each iteration and once the loss is lower than the value of err_specified, we would terminate and return the model.

Finally, with err_specified = 0.5, we find the result is acceptable. We observe from Figure B that the convergence time has a tremendous raise after batch size = 1 and then it increases slowly as the batch size grows. From Figure C, We also find that the loss is huge at batch size = 1 and then it decreases as batch size increases, this result also responses to Figure D, the accuracy generally gets better with larger batch size. It is clearly that batch size = 1(SGD) is a special case in our experiment, the reason for this could be the solution based on one data point, it's very noisy and may go off in a direction far from the batch gradient. And since our given dataset is relatively small with only 600 datasets, which is really close to 512, the difference is not obvious. To conclude, the best case is when batch case = 512 with accuracy = 0.78. The mini-batch SGD, on one hand, it avoids the noises in SGD, on the other hand, it is usually faster than full-batch gradient if the dataset is huge.

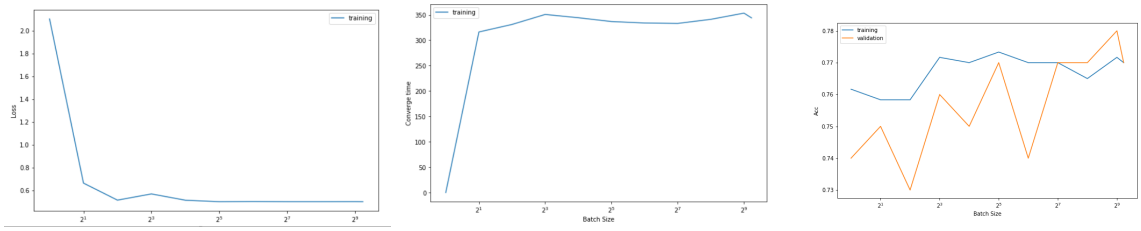


Figure B, Figure C and Figure D

1.4 Add momentum to full-batch GD

Furthermore, we implement and explore the use of momentum in gradient descent. Specifically, we analyze the impact of momentum on the convergence speed and the quality of the final solution. In order to know how momentum affects the model, different Momentum Coefficient (in the rest of the report, we will call it ß) are tested with full-batch gradient descent. The values 0, 0.2, 0.4, 0.6, 0.8, 0.9 are chosen for testing and comparison. When ß = 0, the gradient descent is simple (without momentum). To make the test comparable, the convergence criteria is set to "err_specified = 0.55". The result shows that the validation accuracy does not change with different values of ß (acc = 0.72) (See Fig.1 in the Appendix), and the number of iterations only changed by 6 which can be ignored since the base number 545229 is very large (See Fig.2).

1.5 Add momentum to mini-batch SGD

In addition to full-batch gradient descent, we also explore gradient descent method on SGD with different mini-batch sizes and compare their effectiveness. In order to make the results comparable, all the parameters taken by the model are kept the same as the ones in Section 2.4. In this part, both the smallest and the largest batch sizes are taken for comparison, and they are 1 and 512 separately. For the smallest batch size, the accuracy is the highest (acc = 0.73) (See Fig. 3) when ß = 0.4 with the minimum number of the iteration = 665280 (See Fig.4). For the largest batch size, the accuracy is the highest (acc = 0.73) (See Fig.5) when ß = 0 with the minimum number of the iteration equals to 637181. (See Fig.6)

As a result, compared with the tests in 2.4, we find that the full batch the most effective without using momentum.

2 Part2: Text Classification

In this part, we report our results for "fake news" detection and explain how we get 80% accuracy on the test set.

2.1 Precess the data

Firstly, we have come up with an innovation that improves the accuracy most: using both unigram and bigram instead of just unigram. It improves the accuracy by more than 3%. This is reasonable since in english, there are a lot of phrases that contain more than 1 word and represent some meaning as a whole. Bigram, a sequence of two adjacent words, empowers our model to "understand" a word within a particular context.

Moreover, we are interested in the "max_df" and "min_df" variables, which means the max and min data frequency. In the appendix, Fig.7 and Fig.8 show that the validation accuracy decrease with lower "max_df" and with larger "min_df". This shows us an interesting thing: we are rewarded by using as much extracted words as possible. And deleting any word, even if it has a data frequency too high or too low, will not help at all. We explain it as the special property of "fake news" dataset. Because the fake news are generated by machines, there's no guarantee that it will follow the english language rules, which mean the english stop-words or word frequency will not apply to our case, too.

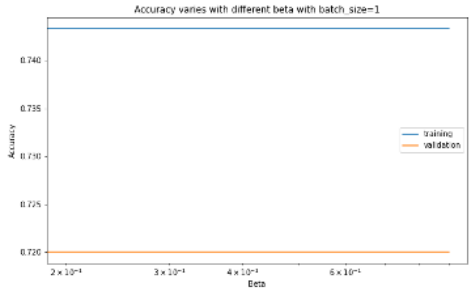
2.2 Pick model and hyperparameters

For the model, we use the SGDClassifier from sklearn. We also tune the parameters by grid search. And we find the following parameters gives us the best validation accuracy, in which case the test accuracy is over 80%.
SGDClassifier(penalty="l2", alpha=0.00001, loss="hinge", max_iter=1000, verbose=0, learning_rate="adaptive", eta0=0.1)

3 Discuss and Conclusion

To sum up, in part 1, we implement different techniques to achieve optimization, and each method has its own strengths and limitations. We find Mini-Batch SGD in between SGD and full-batch gradient descent, which avoids too large noises and also has a relatively high speed.

4 Appendix



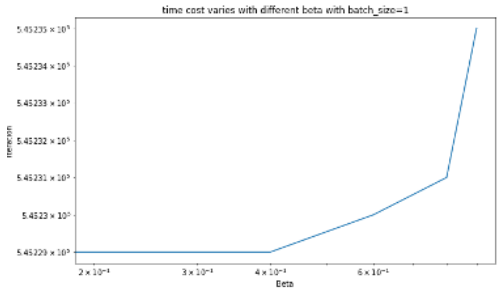
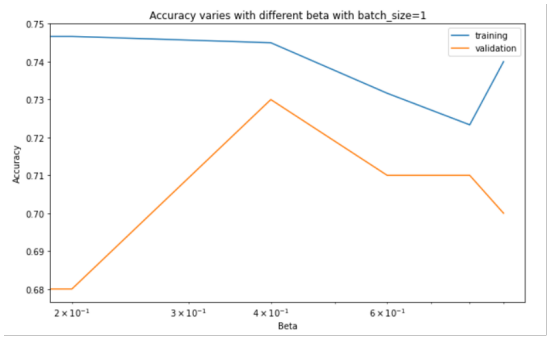


Fig.1 and Fig.2



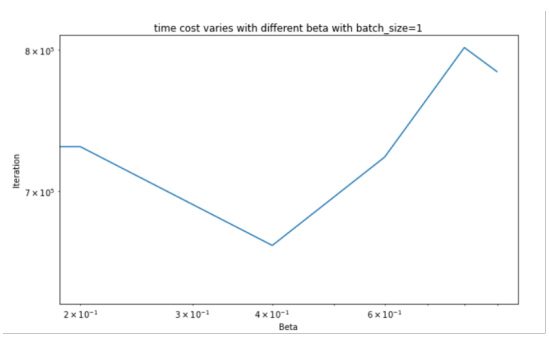
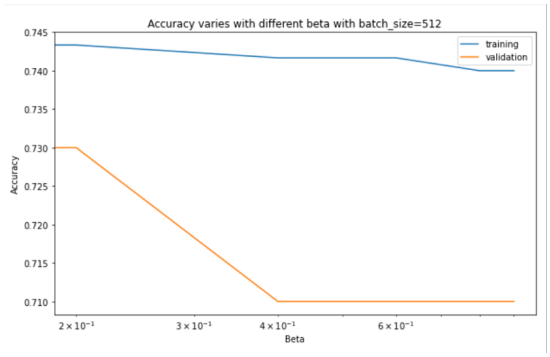


Fig.3 and Fig.4



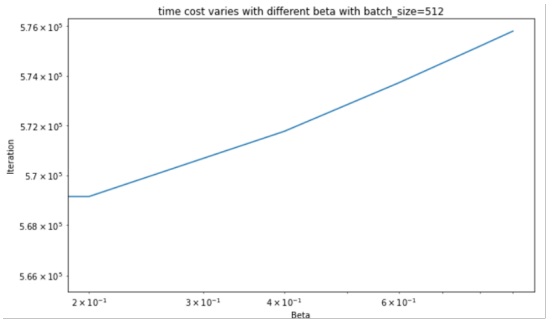
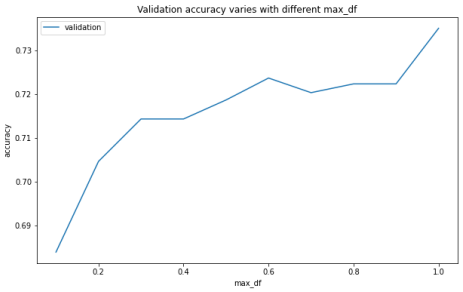


Fig.5 and Fig.6



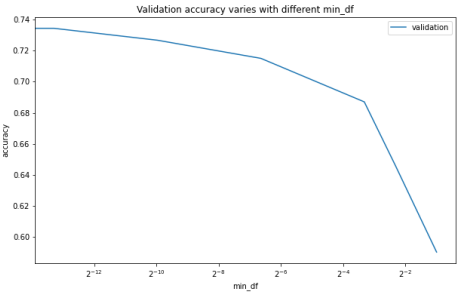


Fig.7 and Fig.8

References

[1] Keshav Dhandhania. *End-to-End Data Science Example: Predicting Diabetes with Logistic Regression*. 2018. URL: https://towardsdatascience.com/end-to-end-data-science-example-predicting-diabetes-with-logistic-regression-db9bc88b4d16.