Multi-Class Logistic Regression and Gradient Descent

COMP 551 Project 2
Xueyang Zhang - 260886286 Zhangyuan Nie - 260924723 Ridwan Kurmally - 260851367

Abstract

In this project, we implemented multi-class logistic regression from scratch and compared it with K-Nearest Neighbours (KNN) and Naive Bayes on the digits dataset and letters dataset from OpenML. We tracked performance indicators such as the accuracy and the running time of models with different hyper-parameters using a simple grid search. We have found that both logistic regression and KNN work well in classical multi-class classification problems. Logistic regression is fast during prediction while KNN struggles to scale with large datasets. Naive Bayes, on the other hand, has lower accuracy due to holding a strong and wrong assumption for our datasets.

Introduction

Stochastic gradient descent, as an optimization algorithm, first appeared in the literature around the 50s[4]. After that, plenty of research has been done on it, which we take for granted today. This decades-old algorithm is still one of the most used in today's industries and academia. In order to become more familiar with Gradient-based optimization, we have decided to dive deep inside its implementation. We implemented a multi-class logistic regression model and mini-batch optimization using gradient descent with momentum. To train our models, we decided to use some classic multi-class image recognition datasets namely the digits dataset and the letters dataset. We have found that with a good combination of hyperparameters, both Logistic regression and KNN gives an impressive accuracy on both training and validation sets. However, optimal accuracy seems to always require longer training time. On the other hand, Naive Bayes seems to have way lower accuracy predicting the correct label, which can be attributed to its assumption on the independence of features, which is not true with our datasets.

Datasets

We used two image classification datasets in our experiments: the UCI ML hand-written digits dataset[3] included in scikit-learn and the letter recognition dataset[1] loaded from OpenML. The features of the letters dataset are the normalized values of statistical moments and edge counts of each letter. The classes are pretty evenly distributed. We used datasets that are easily accessible and free of missing values to focus on the implementation and the analysis instead. All the features in both datasets are continuous, so we used one-hot encoding only for the categorical-like labels.

Results

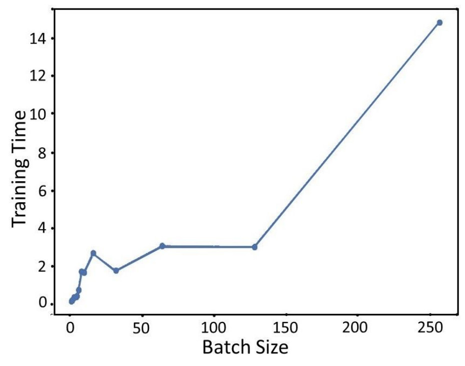

Figure 1: The effect of batch size on the training time of the digits dataset

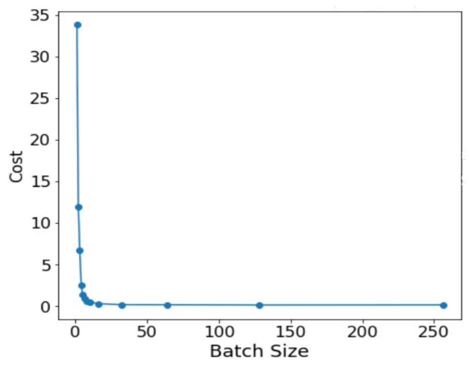

Figure 2.1: The effect of batch size on the average cost of the digits dataset

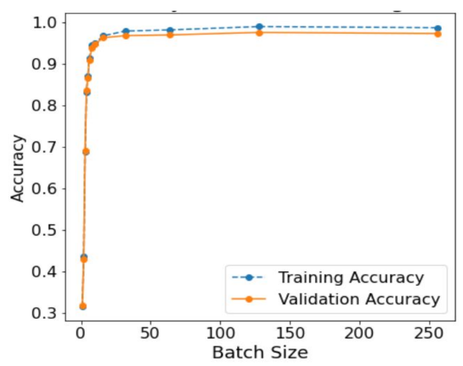

Figure 2.2: The effect of batch size on the accuracies of the digits dataset

As we can see in Fig. 1 for the digits dataset, the training time increased drastically as the batch size increased. However, the cost function decreases rapidly until the batch size is 32 and almost stops decreasing after that. Same with the accuracy, which stops increasing after a batch size of 32. This seems to confirm that 32 is a good starting point for optimal batch size[2]. The training time and the average cost behave similarly for the letter datasets. However, the accuracy continues increasing noticeably with higher batch size. The training accuracy and the validation accuracy are similar in both cases, which indicates a low variance in the model, suggesting we might have underfitted the letters dataset, certainly not overfitting.

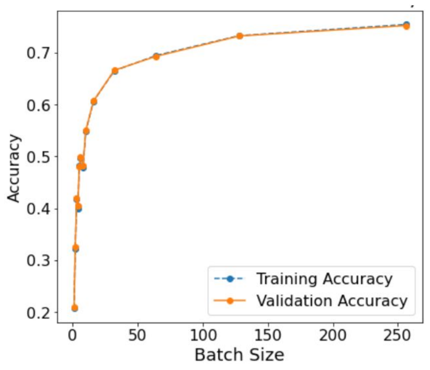

Figure 3: The effect of batch size on the accuracies of the letters dataset

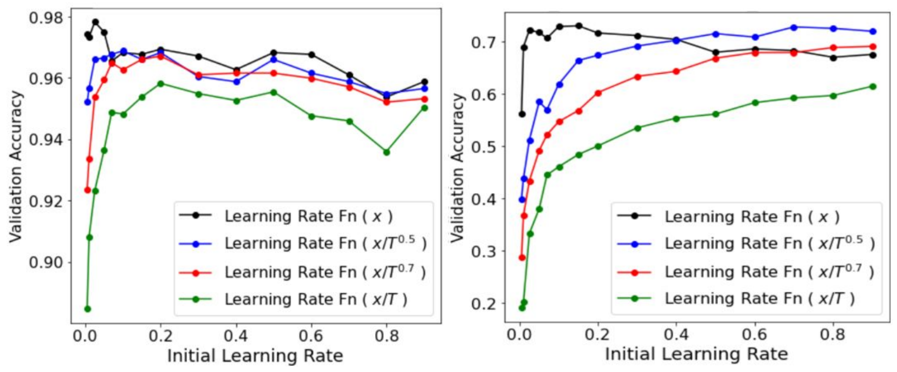

Figure 4: The effect of initial learning rate and the learning function on the training time of the digits (left) and the letters (letters) datasets

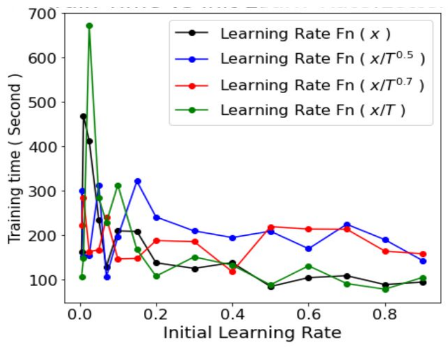

Figure 5: The effect of initial learning rate and the learning function on the running time of the letters datasets

For the digits dataset, validation accuracy peaks at around 0.1 for each learning rate function, meaning a small learning rate stops too early because the rate of change in w is too small. A large learning rate creates oscillation near the optimal point, which results in low accuracy models. Similarly for validation accuracy in constant learning rate of the letters dataset, but not for decreasing learning rate functions: the increasing tendency suggests that the validation accuracy can potentially increase if we further increase the initial learning rate or the max allowed iteration. The opposite trend happens on cost, the least cost happens at some value and increase, suggesting a worse model, except for three decreasing learning rate functions for letters whose costs are monotonous decreasing, indicates underfitting. Training time for small initial learning rates tends to be high, and the constant learning rate is generally the lowest.

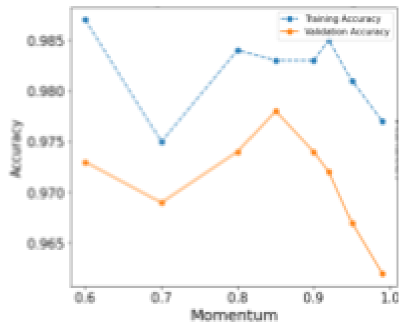

Figure 6: The effect of the momentum on the accuracy of the digits dataset

For the momentum, we can see from Fig. 6 and Fig. A1 that unlike the batch size, a higher accuracy usually implies a lower training time. This is expected since a higher momentum may miss the global optimum and result in a higher training time while a smaller one may not be helpful to accelerate the training process.

To dig deeper into the possible optimizations, we also added adaptive per-parameter learning rate into account, namely the gamma for the second moment. Though the change between different betas is limited, a higher RMS beta is favoured as we can see in Fig. 7, where the parameters are more adaptive to their own changing rate, so the resulting accuracy is higher and the cost lower.

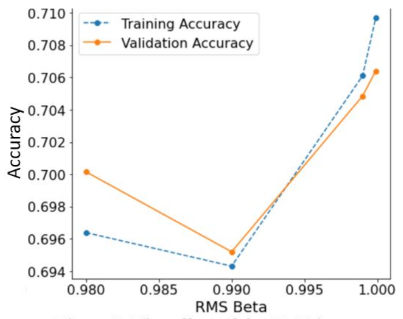

Figure 7: The effect of the RMS beta on the accuracy of the letters dataset

Regularization prevents overfitting by punishing high weights; since we are terminating around the lowest validation error. It cannot overfit, so lambda is unnecessary. A greater lambda value leads to an underfitting model with lower accuracy, probably due to early stopping. That’s what we see in Fig. 8: with validation and training accuracy nearly always decreasing, training accuracy becomes less than validation accuracy, suggesting underfitting. We can also see from Fig. 9 that we have significantly less training time as lambda increases, suggesting GD stopped too early for the letter dataset.

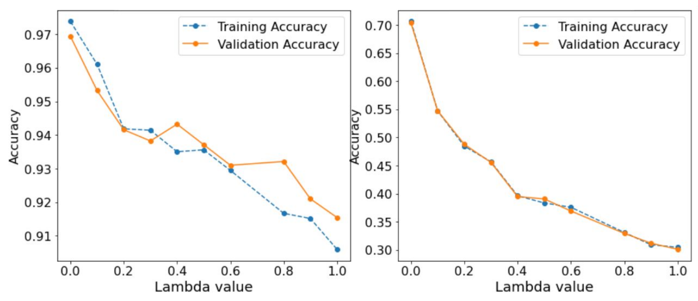

Figure 8: The effect of the lambda value on the accuracy

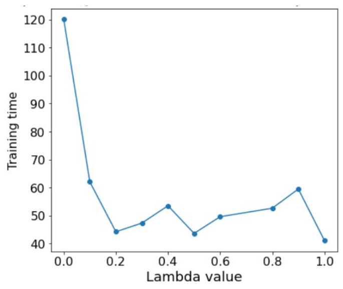

Figure 9: The effect of the lambda value on the running time on the letter dataset

Overall, KNN performs better than the logistic regression in both of our datasets as we can see in Fig. 10. This is likely because our datasets are already normalized and well balanced, and more importantly, the labels of neighbours of a record do suggest its label, especially for image classification on pixel values. Missing values also do not exist which seems to be an ideal use case of KNN. It is difficult to compare their running time since KNN does not really require any training. But we can say that logistic regression is fast when predicting while KNN struggles to scale with large datasets during prediction.

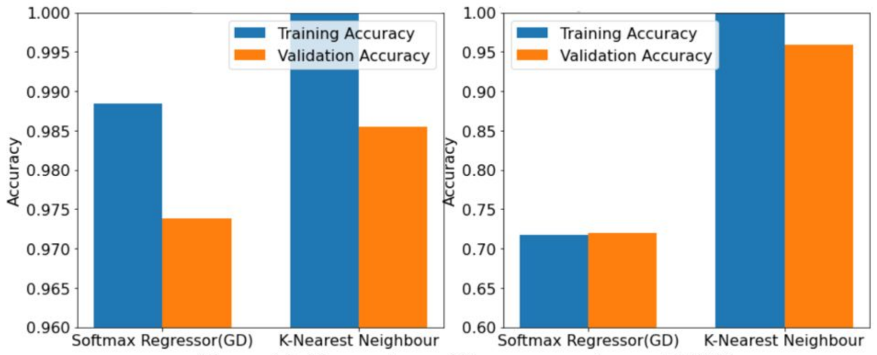

Figure 10: Comparison of linear regression and KNN

Discussion and Conclusion

The performance of logistic regression, KNN and Naive Bayes is consistent with the theory. Logistic regression linearly classifies labels, and such boundary exists for pixel values, resulting in high accuracy. However, It does not exist for statistical values in the letters dataset, resulting in low accuracy. KNN performs well because scales are consistent, and similarities with neighbours do suggest true labels. Naive Bayes performs badly because neither pixel values nor statistical values are independent in our datasets. As a result, when dealing with image classification problems based on pixel values or statistical data, KNN is a better choice. Logistic regression should be used with caution since linear boundary does not always exist on features to classify.

We believe that a decreasing learning rate at -0.5 power should work better than constant learning rate for large datasets, so we can further increase the initial learning rate to see if it gives higher accuracy. Adam GD should give better results overall. However, this is not the case since we used the best hyperparameters of the momentum GD, which are not necessarily good for Adam. Since our grid search took several days already, our time does not allow us to find the best hyperparameters for ADAM. The termination condition has been set at T=100, which works well for the digits dataset, but not for the letters dataset. Since for a given batch size, a batch represents less of the overall dataset, using a small number of batches to determine the w may lead to underfitting. However, the training time increases with T, and we do not have that much time to train our models.

Statement of Contributions

Xueyang Zhang implemented the modules for Gradient Descent. Zhangyuan Nie implemented the modules for Softmax Regression and K-fold validation. The analysis part is done by Ridwan Kurmally with the help of Xueyang Zhang. Zhangyuan Nie and Xueyang Zhang are responsible for most of the current writeup.

References

[1] "Letter Recognition Data Set." *UCI Machine Learning Repository*, https://archive.ics.uci.edu/ml/datasets/Letter+Recognition. Accessed 17 November 2020.

[2] Masters, Dominic, and Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks." *CoRR*, 2018. arXiv: 1804.07612.

[3] "Optical Recognition of Handwritten Digits Data Set." *UCI Machine Learning Repository*, https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits. Accessed 16 November 2020.

[4] Robbins, Herbert, and Sutton Monro. *A Stochastic Approximation Method The Annals of Mathematical Statistics*. vol. 22, 1951.

List of additional figures

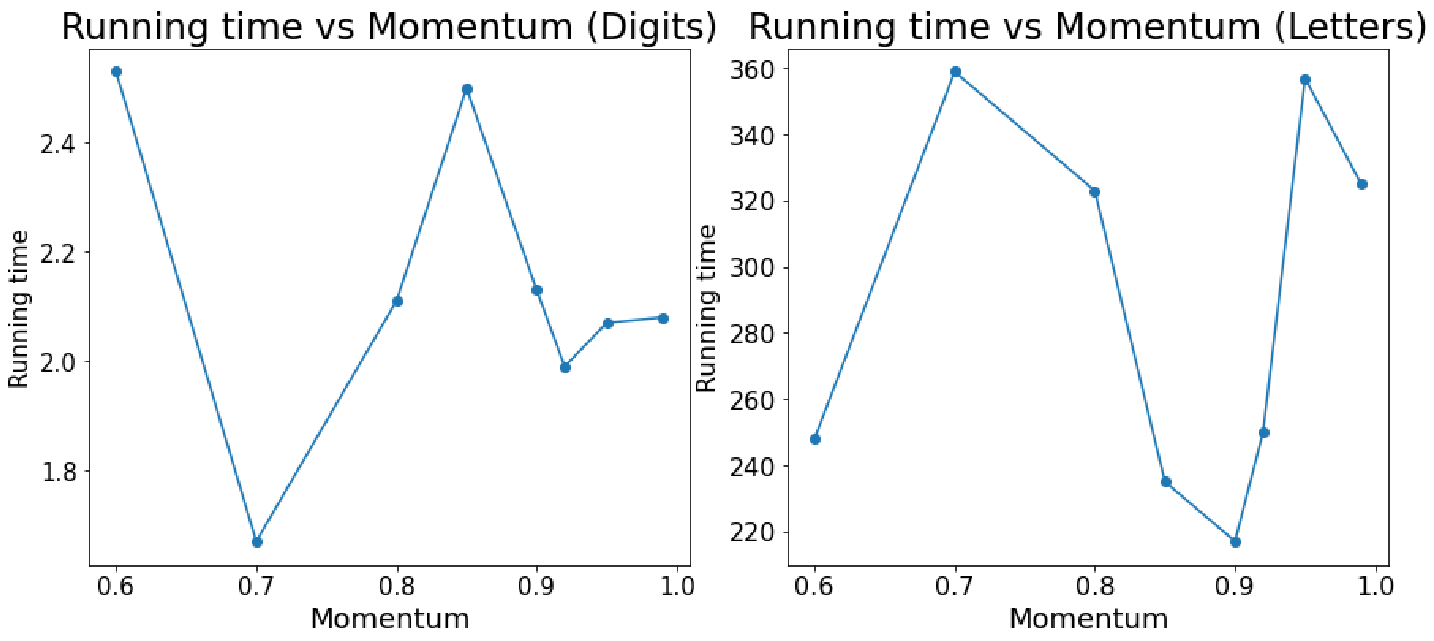

Figure A1: The effect of momentum on the training time of the digits and the letters datasets