Multi-Class Logistic Regression and Gradient Descent

Julian Armour (260804046), Saksham Mungroo (260768072)

Abstract

In this project, the softmax regression model with stochastic gradient descent as its optimizer was implemented from scratch and studied using two well-known datasets in Machine Learning: the Digits dataset and the Iris dataset. The accuracy and runtime of the model was analyzed with respect to different parameters for the optimizer. Finally, the accuracy of our implemented softmax regression model was compared to the accuracy obtained by an off-the-shelf implementation of K nearest neighbours from the sklearn library.

Introduction

This project undertakes implementing multi-class logistic regression and comparing its performance against K-Nearest Neighbors (KNN) on two datasets. The model is implemented entirely from scratch, as well as the cross-validation algorithm used to compare it to KNN. Multi-class logistic regression finds its optimal parameters by gradient descent (GD). This project's GD algorithm uses mini-batching, momentum, and 3 termination conditions: the first stops GD if its validation error does not improve for T iterations, the second stops after some number of iterations is reached, the last stops if the magnitude of the gradient is sufficiently small (near-zero). Therefore, the model has 6 hyper-parameters: learning-rate, batch-size, momentum, validation-stop, max-iters-stop, gradient-magnitude-stop. This project fixes validation-stop to 40, max-iters-stop to 1000, and gradient-magnitude-stop to $1x10^{-8}$. "Close-to-optimal" values of learning-rate, batch-size, and momentum are found through exhaustive grid-search. Following this step, the accuracy of softmax regression as well as the time taken to build the model is analyzed through 5-fold cross-validation with respect to different values for momentum, learning rate and batch size used in the optimizing step.

Dataset

The first dataset explored in this project is the digits dataset provided by Scikit-Learn. This dataset contains 8x8 pixel images of the digits 0 to 9, which are the labels. Each feature corresponds to pixel values in the range 0 to 16 indicating white to black. The second dataset is the Iris dataset found on OpenML(www.openml.org/d/61). It has 3 classes: Iris Setosa, Iris Versicolor, and Iris Virginica, each denoting a type of iris plant. There are 4 features: sepal length, sepal width, petal length, and petal width, each in centimeters

Results

For both datasets, grid search is performed on the hyper-parameters: batch size, learning rate, and momentum. The search is performed over the following sets: batch-size{1, 5, 10, 20, 30}, learning-rate{.0001, .001, .01, .1}, momentum{.25, .50, .75, .95, .98, .99}.

On the digits dataset, the best combination of these hyper-parameters was found to be batch-size=20, learning-rate=0.01, momentum=0.5. For KNN, K= {1, …, 20} are compared with cross-validation to find the best K, which was 1. The performance of these two models on the digits dataset is shown in Table 1.

Table 1- Performance of multi-class logistic regression and KNN on the digits dataset

Multi-class Logistic Regression	KNN
96.49	98.55

The best combination found for the iris dataset was batch-size=10, learning-rate=0.1, momentum=0.25. Once again KNN was compared, and the best K was found to be 13. The performance of these models is shown in Table 2.

Table 2- Performance of multi-class logistic regression and KNN on the iris dataset

Multi-class Logistic Regression	KNN
98.67%	96.67%

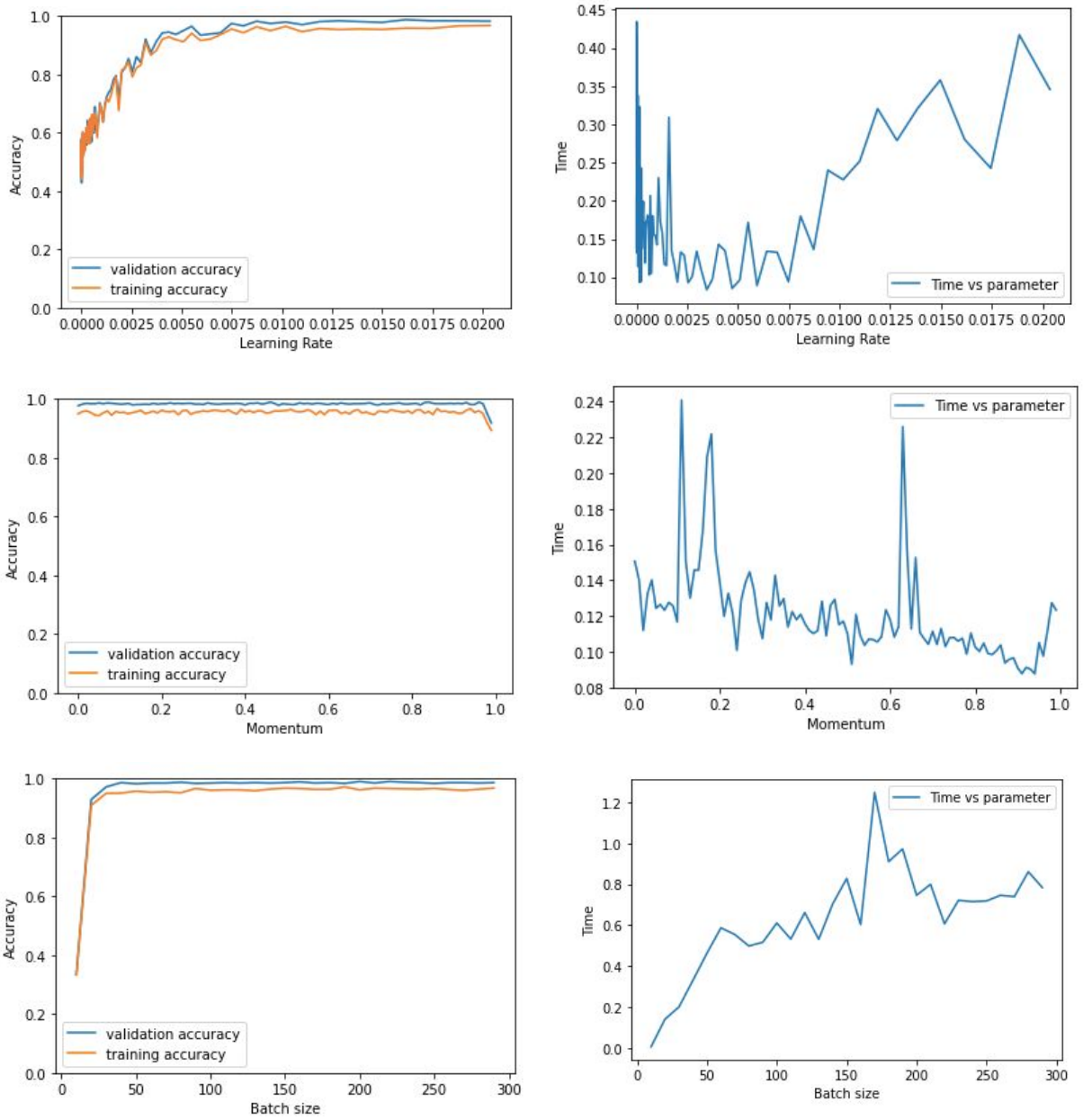
These models produce very similar results on both datasets and both perform really well. Though multi-class logistic regression is slightly favoured in the iris dataset by 2%, it isn't for the digits dataset. It probably performs better on the iris dataset due to the data not being normalized, which affects the performance of KNN. Multi-class logistic regression is not affected by the lack of normalization since it assigns C (the number of classes) parameters to each feature. When testing this hypothesis by normalizing the data, Logistic Regression and KNN achieved 94.67% and 95.33% respectively. So KNN slightly outperforms Logistic Regression when the data is normalized.

Next, we visualize the accuracy of softmax regression as well as the time took by the algorithm to build an optimized model using stochastic gradient descent. In particular, we are going to plot the training and validation accuracy of our self-implemented softmax regression model using 5-fold cross validation as well as the running time of our algorithm with respect to 3 parameters: momentum, batch size and learning rate which are the parameters used in the optimization step (ie in SGD). To fulfill this task, we are going to fix 2 of these parameters with their previously found "close-to-optimal" values from grid-search and vary the 3rd parameter.

Important Note: In the case of momentum, epsilon was increased to 1e-1 to be able to observe the effects of larger momentum on the running time of our algorithm. If the previous epsilon of 1e-8 was used, the max number of iterations required by the algorithm to build an optimized

model would be reached in all cases at the same time and hence would not reflect the effects of momentum.
Here are the results obtained on the iris dataset:

Figure 1 - Accuracy of softmax regression and model building time with respect to learning rate, momentum and batch size using stochastic gradient descent as model optimizer.



Discussion and Conclusion

The softmax logistic regression model performs very well on both the datasets and achieves similar results to off-the-shelf KNN. Logistic Regression likely outperformed KNN on the iris dataset due to the data not being normalized. When the iris set was normalized, KNN outperformed Logistic Regression, showing this hypothesis is likely true.

From the results obtained on the iris dataset (similar results were obtained on the digits dataset) we can confidently say that the learning rate strongly influences the accuracy of the softmax regression model using SGD. Indeed, it is clear that the accuracy of the model increases with the learning rate to reach a plateau. With a small learning rate, we cannot reach the global minimum of our cost function using SGD prior to exceeding the maximum number of iterations allowed (the algorithm stops), making our prediction model less accurate. With an increase of the learning rate value, SGD converges faster to this global optimum while increasing the accuracy of the model. The plateau in the learning rate graph indicates that the optimization function is very “wide”, so that even large changes in the weight will not affect the model’s prediction accuracy by a lot. This is also aided by the validation termination condition, which ensures that a good solution is found as long as one weight-vector with high accuracy is found. Furthermore, a very interesting trend can be seen on the “time vs learning rate” graph. We observe a decrease in the running-time of the algorithm with a higher learning rate but rapidly followed by a steep increase. This observation is probably due to the gradient **converging faster** to the global minimum with a better increasing learning rate at the beginning. However, once we exceed this optimal learning rate by too much, the gradient starts **diverging**, leading to an increase in time to find an optimal model (and possibly not finding one at all).

The momentum does not have an effect on the accuracy of our model. This is justified, since momentum is a parameter that only helps with oscillations and permits faster convergence. However, as SGD is given enough time (max # of iterations) it finds the global minimum leading to a high accuracy. And this is clearly reflected through the “time vs momentum” graph where we can perfectly see that a larger momentum leads to a faster running-time since SGD can now converge faster to the minimum of the cost function due to less oscillations in the gradient.

Batch size influences the model’s accuracy, with larger batch sizes giving more accurate models.This is in agreement with what was discussed in class: smaller batch sizes lead a noisy estimate of the gradients with steps that are “in average” in the right direction. Hence even close to the minimum, smaller batch sizes give a noisy approximate of the weight vector leading to less accurate models. On the other hand, larger batch sizes are much more smooth/have less noise. Given that the learning rate fixed is optimal, there is a guaranteed improvement at each step of SGD which is reflected through high accuracy of the model. However, the tradeoff between small vs large batch size is time. Larger batch sizes are computationally more expensive to deal with than smaller batch sizes, hence leading to an increase in the running-time of the SGD algorithm, clearly seen on the plot “Time vs batch size”.

Finally, the tight evolution of training and validation accuracy close to each other suggests that our model generalizes well on our datasets (ie, does not overfit nor underfit).

Statement of Contribution

Julian implemented gradient descent with minibatching, momentum, the validation termination condition, cross validator, softmax function, model comparison on the Digits and Iris datasets.

Saksham implemented part of softmax regression class + helper methods, methods for plotting accuracy and time vs parameters and grid-search cross-validation.