ROB313 Assignment 3 – Gradient Descent

Objective

The objective of this assignment is to explore the effects of full-batch gradient descent (GD) and stochastic gradient descent (SGD) with various learning rates first on the *pumadyn32nm* regression dataset. Both gradient descent methods were also used on the classification problem *iris,* to gain new insights on log-likelihood functions and their comparison to the binary accuracy metric. Overall, both regression and classification models were used to observe the differences in GD and SGD, and the effect of too large and too small learning rates on a model.

Code structure and strategy

The code was all written in one file, labelled *a3.py.* In this file, I first copied over some of the commonly used functions from previous assignments, then proceeded to write the code in vectorized form in two sections. First, there is the section for Q1, the gradient descent on the linear regression model (pumadyn32nm). The main functions for this section are:

- grad_desc (): gradient descent algorithm that can handle both full-batch and mini-batch gradient descent
- plot_losses (): given a list of losses for each iteration, makes the plots of "losses per iteration" and the linear regression benchmark RMSE plot (see fig. 1 and 2)

For the logistic regression model gradient descent (Q2), the same code structure was used, but there were more helper functions to compute vectorized versions of the sigmoid and the log-likelihood functions outlined in the handout. As in past assignments, since this is a classification problem, we also compute the accuracy of correct classifications that the model produced (in addition to the log likelihood). The main functions for this section in the code are:

- log_grad_desc (): logistic gradient descent algorithm that outputs the optimal accuracy and log likelihood learning rates
- plot_logistic_losses (): makes the plots of the negative log likelihood values per iteration for both GD and SGD

More information about the code can be found in the README.md file.

Gradient descent

As a part of the regression gradient descent algorithm, we first computed a set of weights and RMSE values using the linear regression algorithm developed in previous assignments to use as a benchmark. Then using both full-batch and stochastic (mini-batch 1) gradient descent, we computed the weights using gradient descent at multiple different learning rates.

In GD, we saw that the lower learning rates (i.e. 0.0001) not converging at all to a model that had performance even close to that of the linear regression model, whereas in SGD it was found that the higher learning rates (i.e. 0.1) created too much fluctuation in the graphs of loss vs. iteration number, which cluttered the graph and resulted in unexpected results at times. Additionally, in SGD, we found that one of the "middle" learning rates resulted in the optimal RMSE since a low learning rate resulted in the model converging too slowly, and a high learning rate resulted in the model uncertainty being high, and large variations between iterations. See figure 2 for a visualisation of the SGD results on the pumadyn32nm dataset.

Table 1, Results from gradient descent on a linear regression model (pumadyn32nm dataset)

Gradient descent	Learning rates used	Optimal learning rate	Test RMSE
Full-batch	[0.001, 0.01, 0.1]	0.001	0.8668
Mini-batch 1	[0.0001, 0.001, 0.01]	0.001	0.8825

Note that the optimal learning rates of both gradient descent methods ended up being the same, however resulted in a slighted larger root-mean squared error for the stochastic method vs. the full-batch.

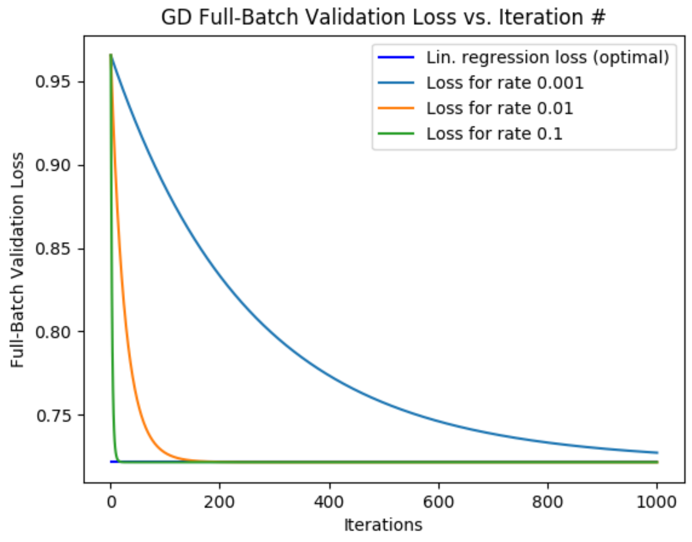

Figure 1, Full-batch gradient descent validation RMSE with linear regression benchmark (pumadyn32nm)

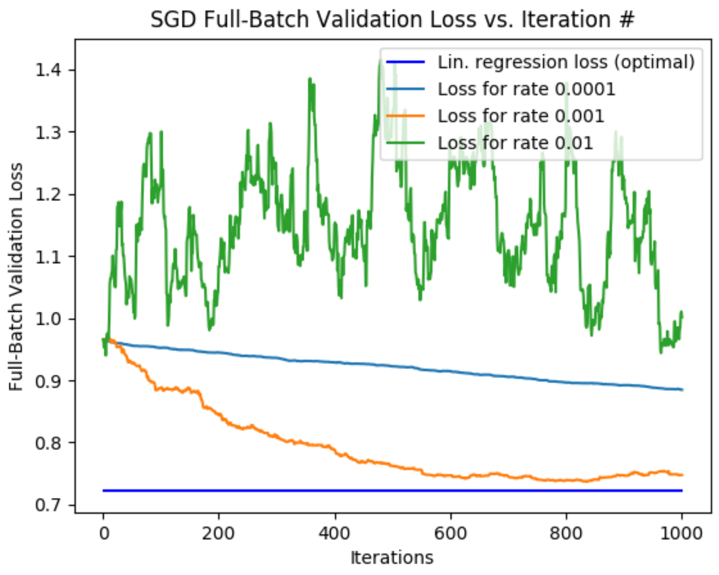

Figure 2, Stochastic gradient descent validation RMSE with linear regression benchmark (pumadyn32nm)

Logistic regression gradient descent

For the iris classification problem, we used full-batch and stochastic gradient descent again, but this time with both log-likelihood and accuracy metrics for determining the best model. When using the accuracy to determine the best model, we only get binary information from whether or not a prediction is correct, whereas with the log-likelihood metric gives a float result that tells us how "correct/incorrect" our model currently is. This inherently makes the log-likelihood a better metric since it provides more insight to our model.

That being said, if the value of our log-likelihood is $\hat{f}\left(x^i, w\right) = 1$ then we will have a 0 within the $\log(1 - \hat{f}\left(x^i, w\right))$ formula, which would result in a log probability of ∞. If the actual y value is 0 in this case, the model will be predicting the exact opposite of the results, which is unreasonable behaviour. Because of the log-likelihood formula, guessing a value of 0 or 1 with 100% certainty will result in unanticipated probabilities.

As for the model training parameters, learning rates higher than 0.01 were not used for the logistic regression model because there was too much fluctuation in the results for stochastic gradient descent. For both GD and SGD, the learning rates used were [0.0001, 0.001, 0.01]. Various gradient descent iteration numbers were tested, but it was found that approximately 2000 was the optimal value. When we ran only 1000 iterations, the value of test log likelihood would be larger, and if we ran 5000 iterations, the algorithm would take much longer but the results would not improve substantially.

Table 2, Results from gradient descent on a logistic regression model (iris dataset)

Gradient Descent	Optimal accuracy rate	Test accuracy	Optimal log likelihood rate	Test log likelihood
Full-batch	0.0001	73.33 %	0.0001	-7.07
Mini-batch 1	0.01	73.33 %	0.0001	-10.14

It is worth noting that for both GD and SGD, the accuracy values are the same, whereas the log likelihood values differ, confirming our hypothesis that the log-likelihood metric would provide more information about the model.

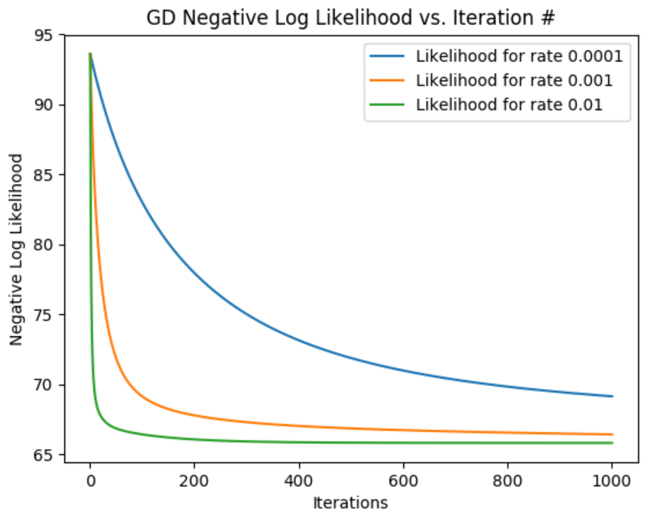

Figure 3, Un-normalized GD negative log-likelihood per iteration for the validation set (iris)

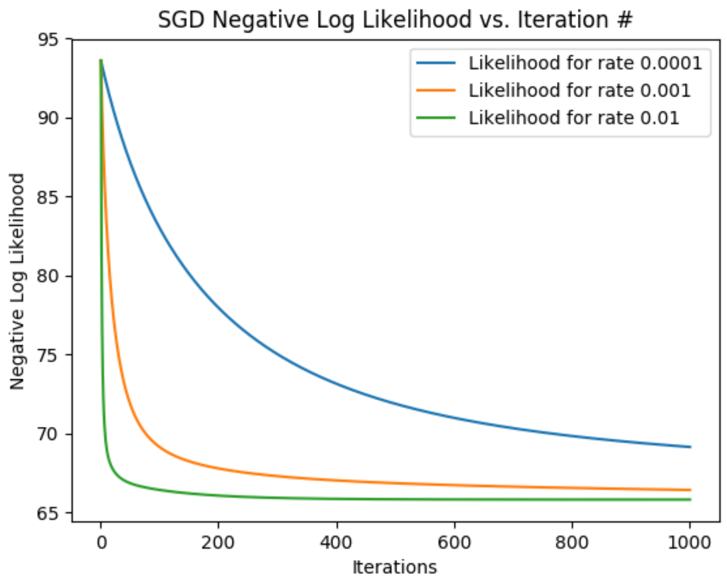

Figure 3, Un-normalized SGD negative log-likelihood per iteration for the validation set (iris)