ML Lab 2

Brian Ezinwoke

April 2023

3 Explain briefly the knowledge supporting your implementation and your design step by step. Explicitly comment on the role of any arguments you have added to your functions.

3.1 l2_rls_train

Additional parameters:

- lmbd [float]: the regularisation parameter. Default is 0
- classification [boolean]: specifies if the labels should be converted to one hot encoding or not

In my solution, I first converted my labels from ordinal encoding to one-hot encoding as this provides the correct format for multi-class classification.

Next, in order to accommodate a bias parameter, I extended my data set with a column of ones at the start of each sample.

For my calculation of the weights, I minimised the L2 regularised sum of squares through zeroing the loss gradient and used to corresponding Moore-Penrose inverse obtained from solving the linear systems to formulate the set of equations for the weights. In the case where the regularisation paramter is 0, I simply used the inbuilt pseudo-inverse function in numpy and for the case where lambda is not 0, I manually wrote the calculation.

3.2 l2_rls_predict

Additional parameters:

- classification [boolean]: specifies if the output should be converted to ordinal encoding or not

For my prediction function, I first augmented the data parameter to have an additional column of ones. Then I calculated the predicted labels by calculating $\tilde{X} \times W$ and then choosing the label with the highest value.

4 Observations and Analysis

Firstly I split the data into test and train samples. Using random sub-sampling I ran the

To determine the hyper-parameter, I iterated over 10 different lambda values and for each one, I used random sub-sampling over 5 iterations to calculate the error rate. I repeated this process 3 times and calculated the best lambda value. From this, I chose my lambda value to be 1.

After running on my test set, I found that I could achieve 94% accuracy in my classification.

The easiest subjects to classify had unique identifying features such as glasses and unique facial features. Whereas the hardest samples to identify tended to have blander features and less contrast in the image.

5 MAPE and analysing model performance

The MAPE was 22%. The Error is quite high as the model struggles to predict the right side of images which are asymmetric or have low contrast. Additionally, the model works best on subjects that are facing directly forward as the image is symmetric.

Our model could be improved by choosing a different hyper-parameter as opposed to simply using 0 because the model could be struggling due to over-fitting to the training data.

6 How did you choose the learning rate and iteration number? Explain your results.

I chose my learning rate as 0.001 and my iteration number as 200. This is because other smaller learning rates resulted in the number of required iterations being too high before a solution was reached and larger learning rates meant that no solution was ever reached. My learning was 200 because this was a safe amount of iterations for the algorithm to find a solution if possible although from my tests, solutions found around the 100 iterations mark.

When the learning rate was 0.001, the accuracy of the model was good and reached 100% accuracy for training data and 70% accuracy for test data. However, when the learning rate was set to 0.01, we see the error rate increase over N and the accuracy stayed at 0%. This is because, with a larger learning rate, the gradient descent function overshot the minimum point of the graph hence the cost function is never minimised and the error rate stays high.

7 Hinge Loss

7.1 Implentation of hinge_gd_train

To derive the cost function, I adopted the approach for soft margin SVM with hinge loss. The equation I was left with was

$$O(w) = C\sum_{i=1}^{N} \max(0, 1 - y_i(w^T x_i + b)) + \frac{1}{2} w^T w$$

For the gradient descent, I derived the cost function with respect to w and was left with:

$$\frac{\partial O}{\partial w} = w + C\sum_{i=1}^{N} \begin{cases} 0 & -y_i w^T x_i \geq 1 \\ -y_i x_i & -y_i w^T x_i < 1 \end{cases}$$

7.2 Experiment Design

7.2.1 Experiment 1

For my first experiment, I compared the performance of both algorithms and varied the iteration number and kept the learning rate constant. I tested with 5 different iteration numbers - 100, 200, 300, 400, and 500 - and for each number, I plotted graphs for the change in error loss, training accuracy and testing accuracy, comparing both loss functions.

7.2.2 Experiment 2

For my second experiment, I varied the learning rate and kept the iteration number constant at 250. I tested with 5 different learning rates - 0.0001, 0.0005, 0.001, 0.005, 0.01 - and for each number, I plotted graphs for the change in error loss, training accuracy and testing accuracy, comparing both loss functions.

7.3 Result Analysis

From this experiment, I deduced that the sum of squares loss cost function approached 0 at a faster rate than the hinge loss and plateaued at a loss lower than the hinge loss. Additionally, there were high fluctuations in the hinge loss whereas the sum of squares was a smooth curve. However, the accuracy over iterations plateaued faster for hinge loss than the sum of squares loss and in all cases reached higher accuracy despite extremely high fluctuations. An iteration number of 200 displayed the best performance for both cost functions, both minimising fluctuations and also converging at a fast rate.

As in the previous experiment, the sum of squares error loss approached 0 faster and plateaued closer to 0 than the hinge loss. As before, hinge loss also had high fluctuations however, it is also observed that for large learning rates ≥ 0.005, the results are highly inconsistent and bad results are displayed for both. The accuracy was best when the learning rate was 0.0005, and there were less drastic fluctuations.

Overall, my results showed that the hinge loss gradient descent is much more unstable and highly volatile however can still find accurate weight values. In comparison, the Sum of squares error is much more stable however is unable to find any solution with sub-optimal choices for the learning rate and iteration number.