COMP24112 Machine Learning
Lab 3 Report

Question 3

I have implemented the Regularised Least Squares method which takes as input some data and labels, alongside a parameter lambda, and returns the predicted weights of the model. The method's purpose is to reduce the error and the weights outputted are used to predict data for the future model. (w is also known as the "bias" parameter)

For the l2_rls_train method, the first thing that is being done is creating a matrix X_tildle which is precisely the same as X, with the only difference being that there is a column of 1's on position 0. This has been done for the weight vector W to work accordingly (w is also known as the "bias" parameter).
After this, we are presented with 2 cases. There can be a single-output case or a multi-output case. I split the 2 cases (labels.shape) and then checked the value of lambda. If lambda is 0, then the normal predicted weights are returned. Else, we use a pseudo-inverse matrix to implement the solution.

For the l2_rls_predict method, all that is being done is predict the output using the corresponding formula: X_tilde times the weight vector.

Experiment 1

Binary classification is the process of assigning the data one of 2 values. For my experiment, the 2 values were the labels for the 1st and 30th subjects. In order to assign the data one of the 2 values, we first need to have a threshold. A suitable value would be the average of the 2 labels.
After that, the data needs to be partitioned into training and testing data, needs to be assigned in the corresponding arrays, the labels for the training data need to be correctly assigned (to have a correct solution to train our data by), and then to train the model itself.

The model is being trained using as parameters the training data and the training labels, alongside a parameter lambda which we can modify to obtain the best results. It is calculated 50 times so that the average training and testing error can be calculated after everything is done.

In regards to whether changing the class labels affects the model performance, I have not noticed any substantial differences after testing. I believe that it does not impact the performance because of data normalization. Also, the fact that the threshold is always the average between the 2 values makes it so that whatever 2 class labels we would choose, we would be getting the same results in all the cases.

The accuracy of the training model tends to be above 90%, which is a fairly decent result. The reason that this happens is that the model has been trained on sufficient data. Since the training mean and deviation are 0, that means that the model can accurately predict the class more often than not. Another factor could be the optimal choice of the hyperparameter lambda that controls the balance between the data-dependent error function and the regularisation term.

Experiment 3

As printed in the lab, the error rate for the face completion experiment hovers around 20%. In all honesty, it is not any groundbreaking performance and I believe that the model can be tuned to obtain slightly better results. The first thing that should always be done to improve the performance of the model would be to feed it more data. Since each subject only has 10 faces to train and test the model on, those 10 faces need to be split in some way, a fact that limits the performance of the model. Other means of improving the model are trying out different algorithms which might have better or worse outputs, modifying the parameters to fine-tune the model, or even resorting to a technique called boosting to improve the performance.

Gradient descent

The learning rate affects the size of the steps that are being taken toward the solution. If the learning rate is too big, we might overflow, meaning that the model jumps over the desired minimum output and goes over the boundaries. On the other hand, if we choose too small of a learning rate, then the number of iterations might not be big enough to train the model. Then the obvious solution would be to increase the iteration number. However, in doing so we also increase the time needed for the model to output the results and such, a balance needs to be found between the learning rate and how many times the model is being trained.

The first three graphs represent the output of a sensible choice of the last two parameters for the lls_gd_train function. In order from top to bottom, they show the descending cost of all the iterations (we can see that the cost goes lower and lower until it reaches 0), the training accuracy (which rapidly goes to 100%) and the testing accuracy (which also reaches 100%, but is also dependent on the training accuracy).

The last three graphs however show how unbalanced arguments create an undesirable output. The cost of the iterations overflows, which means that it tends toward infinity, whereas the training and testing graphs are stuck at an accuracy of 50%. This is happening because the model can only choose one of the two labels and remains stuck with it. Since the data is split 50/50 between the two labels, the model will always be right precisely half the time.

Stochastic gradient descent

The experiment that I have done puts side by side the results of the normal gradient descent with the stochastic gradient descent. Both of the models are trained on the same set of data, so the differences are purely related to the algorithms. When talking about the cost, we can see clearly the difference between the 2 approaches. The gradient descent performs way better than the stochastic descent, with very little to no fluctuations. The latter also requires a lot more iterations to reach a similar result than the former. The training and testing accuracies eventually become the same for the 2 descents, although the stochastic one takes noticeably more iterations to reach a desirable accuracy.

All these differences between the two models are because the stochastic gradient descent is being trained every time only on one random sample, whereas the normal gradient descent is always modifying its parameters. However, it is worth mentioning that even though stochastic descent's results are inferior to the other ones, the computational power required for the operations is lower as well. Nevertheless, in order to achieve good accuracy, the model needs to be trained on more iterations, which balances out the reduction in processing power.