COMP24112 Lab 1 Report

Amaan Ahmad

May 2, 2023

0.1 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.

The functions, l2_rls_train() is used to train a linear equation by minimizing the L2-regularized sum of mean square error. It takes training data, labels, and a hyper-parameter lambda as parameters. In this function, I have first created a array containing ones of size of column of the data(X). I appended this array in the data array, x, and named it X_tilde. If the value of lambda is 0, I returned the computed coefficient vector, w, using pseudo-inverse implementation as np.linalg.pinv(X_tilde)@y, where y is the input labels. If lambda is not 0, I calculated L2-regularized least squares solution using the formula reffered in the lectures : ((X_tilde.T@ X_tilde + lmbd*I)ˆ-1@X_tilde.T@y). Here, I is the identity matrix created by np.eye and the inverse was calculated by np.linalg.inv.
The function, l2_rls_predict, takes trained weights, w, and query data as input and returns the predicted labels. To achieve the following, I calculated X_tilde by appending a column of ones to data, as done in the l2_rls_train() function. Then, I calculated predicted labels by the formula X_tilde@w.

0.2 Explain the classification steps, and report your chosen hyper-parameter and results on the test set. Did you notice any common features among the easiest and most difficult subjects to classify? Describe your observations and analyse your results.

In this part of the assignment, I have calculated the best lambda, using oneHotEncoding and random sub-sampling method, to train the data. First, I split the it into training and testing data and labels by assigning 5 random classes to training index, and the rest to testing index. I have also created a function changeToZeroAndOne(), that takes a 2D array and change the predicted labels to binary predicted labels. To find the best lambda, I created an array with 75 random lambda values from 0 to 1. The values contains a mix of numbers like 0.3 and 0.003. Then, for each lambda, a subset of size 3 is randomly selected for each class, and the rest of the data is used for validation. The algorithm is trained on the training data with the selected lambda and used to predict the labels of the testing data. The predicted labels are converted into binary labels, and the error matrix is calculated by subtracting the predicted labels from the true labels. The error matrix is then summed to obtain a scalar value, which is stored in an array. The lambda with the minimum error is selected, and the algorithm is retrained with this lambda on the whole training set. The trained algorithm is used to predict the labels of the testing data, and the predicted labels are converted into binary labels. The error matrix is calculated as before, and the error rate and accuracy are calculated. The lambda value chosen, is the one that gives minimum error. The result of the algorithm on the testing data is:Lambda : 0.1730546238576981 ; Accuracy : 95.0

Yes, I noticed common features among the easiest and most difficult subjects to classify. The most easiest images had, neutral or a common facial expression in all their images. They were also looking in a particular direction in all their images. Thus, it was easy for the algorithm to classify their images. The tougher images had people who had varying facial expression in their images. Some of them were looking in another direction (left or right), in some of their images. There were also some classes which had images where a person is wearing glasses in one image and not in other. Thus features like these made it tough for the algorithm to classify their image.

0.3 Report the MAPE and make some observations regarding the results of the face completion model. How well has your model performed? Offer one suggestion for how it can be improved.

The value of MAPE computed by my algorithm is: 0.4000560458252037 . To calculate this, I have first partitioned the data into training and testing data with 5 random classes to training set. The training data is used to train the model using the funtions: l2_rls_train and l2_rls_predict. Then the MAPE was calculated using the formula MAPE = (1/n) * (—(actual - predicted)/actual—) * 100.
In the face completion model, the algorithm has performed well in all the cases with giving an average MAPE of 0.39. However, the performance varies based upon the input data. In few cases the algorithm performed better when the images of other half of the person were easy to identify because of factors like same facial expression and alignment i.e the person is not looking in any particular (left or right) direction. Similarly, if the person has various facial expression varying in their images, or for example, they are wearing glasses in some images and not in some, then the algorithm suffered as it gave correct but a little blur image.
The algorithm's performance can be improved by providing a fair data set, with images that are constrained to certain restrictions. Also, we can choose a better value of lambda. In this algorithm, we have lambda as 0. By choosing the best value of lambda, we can minimise overfitting or underfitting and get the best results possible.

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

It's essential to choose the right learning rate and number of iterations when using gradient descent algorithm for optimal performance. The algorithm can take a large amount of time to display the results if the learning rate is too small. On the other hand, if it is too big, the algorithm might not converge. Therefore, to overcome this, I started with a low learning rate and gradually increase it until the cost converges within a reasonable number of iterations. Similarly, the number of iterations should be neither too small nor too large, as it might lead to underfitting or overfitting.

In our first experiment, we used a learning rate of 0.001 and 200 iterations. The sum of squares error loss decreased with each iteration until it reached 0, which is good. The accuracy in both training and testing data improved as iterations occurred.

However, in the second experiment, we used a learning rate of 0.01 and 200 iterations. The sum-of-square error increased steeply after 120 iterations, which indicates that the learning rate was too high. This caused the weight updates to be too large, causing the algorithm to overshoot the minimum and diverge, leading to the cost function increasing at each iteration instead of decreasing. This caused all predicted values to be 1, resulting in a accuracy of approximately 50 percent.

0.5 Explain in the report your experiment design, comparative result analysis and interpretation of obtained results. Try to be thorough in your analysis.

In this part of the assignment, I performed a linear least squares model using stochastic gradient descent for classification. The function lls_sgd_train() takes in four parameters: 'data' and 'labels' are the input data and corresponding labels respectively, 'N' is the number of iterations to run the algorithm, and 'varr' is the learning rate for the algorithm. The function returned an array containing the cost of the linear least squares model for each iteration of stochastic gradient descent, and an array containing the weights of the linear least squares model at each iteration of stochastic gradient descent. The formula used to calculate the sum of mean square error is SSE = 1/2 * sum(y - y_pred)$\hat{2}$.
The experiment shows three graphs. The first graph displays the sum of squares error in each iteration. We can see that the graph is a bit irregular compared to the gradient descent approach. This is because we randomly choose one data point in each iteration, leading to fluctuations in the graph. However, the overall trend is a continuous decrease in error. The second and third graphs show the accuracy in both training and testing data, which improves with each iteration.

We can observe that the sum-of-squares error of SGD decreases more slowly than that of gradient descent. This is because GD takes all data points into account on all iterations, whereas SGD considers only one data point at a time. Similarly, the accuracy of SGD and GD also follows the same pattern.