Lab 2 Report - f85709jw

3.2 Report

> 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 equation to calculate the weights for a l_2-regularised least squares model is $W = (\tilde{X}^T\tilde{X} + \lambda I)^{-1}\tilde{X}^T Y$. Where $\tilde{X}$ is the feature matrix containing the input data, it is expanded with 1s on the leftmost column. Y is the input labels to train the model against. λ is a hyper-parameter which controls the balance between the data dependent error function and the regularisation term.

This equation minimises the least squares error function (the error function being the difference between the predicted output and the ground truth output squared). With l_2-regularisation we add a regularisation term which helps avoid over-fitting of the training data to the detriment of performance on new data, by essentially adding distraction to our model. With the trained weights W the gradient of the error function will be `0`, showing that the model is optimised.

In the implementation of `l2_rls_train` we accept `data`, `labels` and `lmbd` as arguments. `lmbd` is an argument that was added to the function in order to allow the user to specify a lambda value. The `data` feature matrix is then expanded with 1s in the leftmost column, and then the weights `w` are computed using the equation outlined above when `lmbd` lambda is set to a non zero value. When `lmbd` is zero, we calculate `w` using the equation $W = \tilde{X}^{+}Y$ ($^+$ is meant to denote the pseudo-inverse) which follows the same principle outlined above, but doesn't have regularisation and is more prone to over-fitting. `w` is then returned.

`l2_rls_predict` takes `w` and `data` as arguments. `w` is the trained weights that will be used to produce a prediction, `data` is the input data that will be used to produce a prediction. `data` is yet again expanded, then the prediction `predicted_y` is computed through $\hat{Y} = \tilde{X}W$.

4.2 Report

> 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.

The experiment was set up by splitting the data into training and test set, where training had 5 images per subject and test had the remaining images. Then there is a loop which iterates over different values for lmdb, increasing in powers of `10` from `0.001` to `100`. The training data is then split into training and validation data through 5-Fold CV. The model is trained on the training data, and then predictions are run on the validation data which is then assessed. If the mean accuracy of the predictions on the validation sets is higher than the previous best, then the best lambda is set to the current value.

Finally, the model is trained one last time using all of the training data with the best lambda found and is then assessed against the unused test data. Over the multiple runs the selected hyperparameter lambda would change, but in general a lambda value of 1 was the best, and there would be an accuracy of around 95%.

From looking at the easiest and most difficult subjects to classify, there aren't very clear patterns that distinguish easy sets of images to classify and difficult ones. However, there may be some patterns that may make it slightly harder to classify. For instance in 2 of the top 3 hardest to classify images we see that the subject is sometimes wearing glasses and other times isnt', this probably makes it harder to classify. With that being said, there is also a subject in the top 3 easiest to classify images who also has a mix of photos with and without glasses. Another thing we may be able to pick up on is that in one of the missclassified photos, the subject's eyes are closed so this probably makes it harder to classify. Also, it seems that facing sideways can also make classification harder. So to bring all of this together, it is likely that in order to improve the accuracy of the classification it would be good to require the test subjects to face the camera straight on, without glasses and with eyes open.

5.2 Report

> 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.

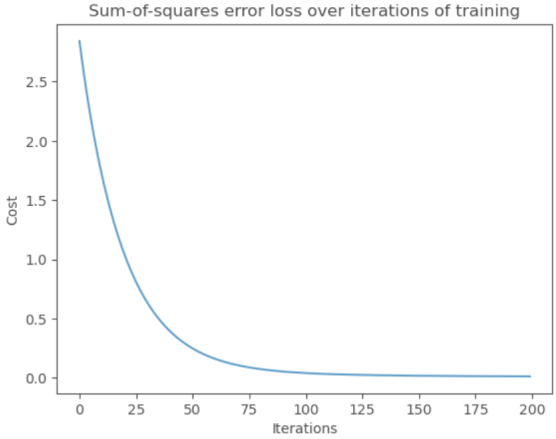
The result varies a lot in terms of how well the predictions depicted a coherent face. With some, one wouldn't know that the right hand side was generated, with others it is painfully obvious as the result barely looks like a face. The MAPE value is around 23%, which seems low however I think over all the model was quite impressive. It is difficult to measure quantitatively how good the model is, because ultimately it comes down to human perception and judgement as to how well the face is depicted which is intuitive. Measuring the mean absolute percentage error doesn't really capture this intuitive sense of how good the prediction was.

In order to improve the model, it would be advantageous to standardise the background colour, the orientation of the person towards the camera, the framing of the face, and also to ensure if there is going to be test input of the person with glasses that the model should have been trained with a photo of the person in glasses and vice versa.

6.3 Report

> Analyse the impact that changing the learning rate has on the cost function and obtained testing accuracies over each iteration in experiment 6.2. Drawing from what you observed in your experiments, what are the consequences of setting the learning rate and iteration number too high or too low?

When the learning rate is set too high (such as an η of 0.01), the model simply doesn't work as the change in the weights are so big that it's not possible to output a reasonable value as the change always overshoots what is a reasonable value. When the training rate is low, the cost decreases very slowly, requiring more iterations overall in order to get to a low cost. When the learning rate is just right, we observe a nice curve on the graph of cost over iterations as can be seen below:

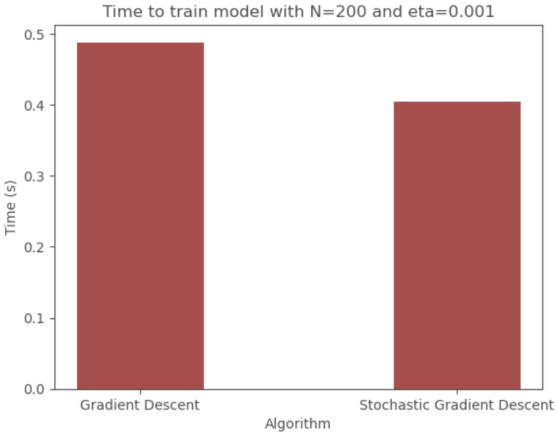


This is due to the error function returning a high value when the error is high, which leads to a large change in the weights leading to a large decrease in the error. And when the error is already low, the change to the weight will also be low.

7.3 Report

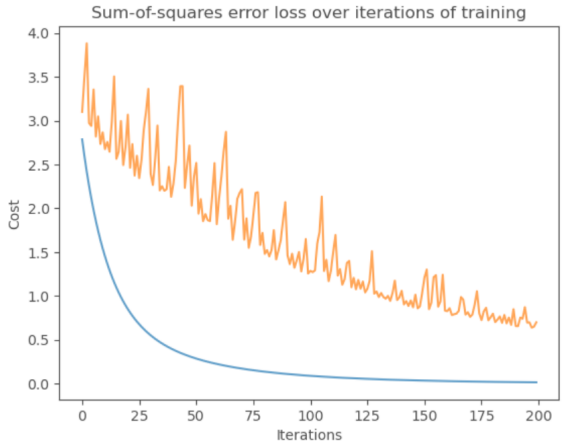
> Explain in the report your experiment design, comparative result analysis and interpretation of obtained results. Try to be thorough in your analysis.

The experiment chosen was to compare regular GD (Gradient Descent) to SGD (Stochastic Gradient Descent). The experiment was to run GD and SGD with the same parameters and compare the running times and accuracy. Both were run with an N value of 200, and an η of 0.001. The result was that SGD was faster by about 15%, as can be seen in the chart below:



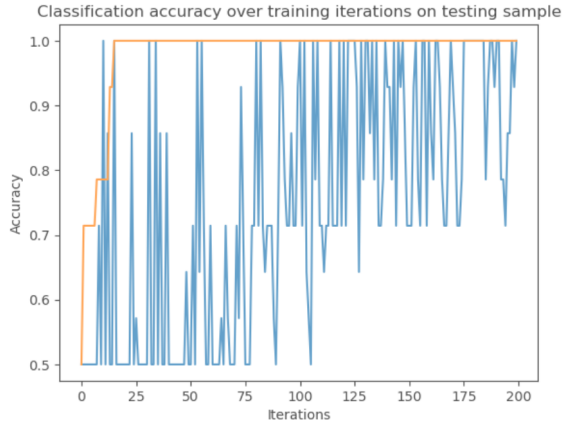
It took the GD model about 0.48s to train, and the SGD model about 0.41s. This makes sense as we pick out only a single value for the gradient descent in SGD which requires less computation per iteration, whereas in GD all of the data is used in each iteration. The difference in runtime is not as stark as one might expect, perhaps there are good optimisations in place for doing the large matrix multiplications used for GD.

In the following graphs blue represents GD, and orange represents SGD. When looking at the result of the cost function over iterations of the training, it can be seen that GD stays ahead the entire time:



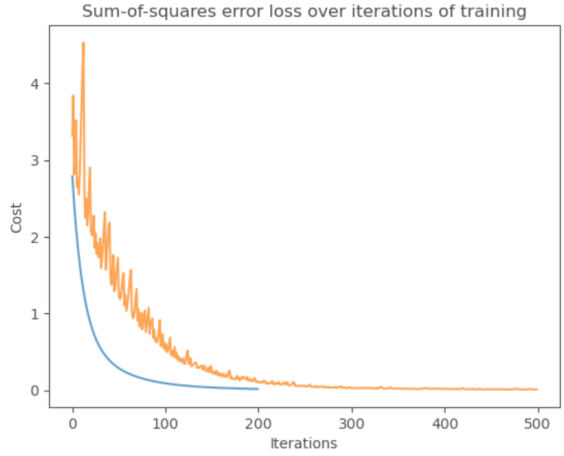
This is to be expected as the entire dataset is being considered at each iteration of the model. Also we can see a smooth curve for GD, but an erratic curve on the SGD. This, again is to be expected as only one sample is being optimised for at each iteration on SGD which may deviate from the global optimal.

The accuracy of the model has a corresponding noise for SGD and a corresponding stillness on GD:



The accuracy at the end when testing on new test data was 100% for both, however this is less stable as we can see from the graph above.

When the SGD model is trained again with a larger number of iterations, the cost can become comparable. It takes around double the amount of time to train compared to GD.



Because the dataset is quite small in this experiment, regular GD is going to be more efficient. The small performance gain from an iteration of SGD compared to GD doesn't end up in an overall performance gain when looking for comparable error. If however, a large amount of data is being worked with then the benefit of SGD will compound and will give a better overall performance.