Machine Learning: Practice 2

Pavel Ghazaryan

May 2 2023

1 3.2: 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.

In *l2_rls_train* I have used the following formula for calculating weight vector for L_2 regularised least squares approach: $w = (X^T X + \lambda * I)^{-1} \cdot X^T y$. Here, X is the given dataset with 1s added in the first column, y is the set containing labels for X, I is the identity matrix and λ is the hyperparameter. I have only added the hyperparameter to the parameters of the function in order to be able to train the model with the desired hyperparameter when calling it.

In *l2_rls_predict* I have used the following formula: $y = X \cdot W^T$ where X is the testing data and W is the vector of model coefficients learned during training. I did not add any new parameters to the predict function.

2 4.2: 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.

I have carried out k-fold cross validation for hyperparameter selection and in order to be able to apply multi-class classification I made sure that I have implemented one_hot_encoding for training my model. After training and obtaining the predicted value on validation data set I have used the $argmax$ function to revert bank to normal numbers and then found the accuracy of each fold. Using these obtained accuracies I have selected the best hyperparameter $\lambda = 10$ and and applied it on the test data. I have achieved an accuracy of 92% which is considered pretty accurate.

Carefully examining the subjects which were easier to classify, I have realised that in all those images the subjects were directly facing the camera while the subjects which were hard to classify had images where their face was not directly facing the camera but was slightly facing in other direction. I believe this was the main cause for misclassifying some of the images. However, I think that in some scenarios facial expressions or objects like glasses can also affect classification accuracy.

3 5.2: 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.

I achieved a MAPE value of 20.8%, which is relatively low but still acceptable for accuracy. This suggests that my model's predictions are somewhat reliable, despite not being highly accurate. Upon examining the images, it is apparent that there are noticeable flaws, particularly in the realism of the right halves of the faces. Nevertheless, these flaws are acceptable. To enhance the

MAPE score, I can fine-tune hyperparameters. For instance, in this example, I arbitrarily chose 0, so conducting k-fold cross-validation to assess a range of hyperparameters and selecting the best one for the test set may yield better results.

4 6.3: How did you choose the learning rate and iteration number? Explain your results.

When the learning rate is too large, it can cause the optimization algorithm to overshoot the minimum of the loss function, leading to instability and poor model performance. In this case, the sum-of-squares error may increase or fluctuate erratically during the training process, as the algorithm bounces around the minimum without converging.

Conversely, when the learning rate is too small, the optimization algorithm may converge very slowly or get stuck in a local minimum, leading to suboptimal model performance. In this case, the sum-of-squares error may decrease gradually but may require many iterations to converge to a good solution.

In general, a higher value for the loop variable may lead to better model performance, as it allows the optimization algorithm to make more updates to the weights and converge to a better local minimum. However, increasing the value of the loop variable can also result in longer training times and increased risk of overfitting if the model is trained for too many iterations.

On the other hand, a lower value for the loop variable may lead to faster training times and decreased risk of overfitting, but may result in a model with lower accuracy or performance.

For these reasons, in the first experiment I have chosen learning rate of 10^{-3} and N to be 200. These were the values which have perfectly presented the gradual conversion of sum of squares error, training and testing error to a lower value. For the second experiment for comparison I have increased my learning rate to 10^{-2} and found out that my initial understanding was correct and that instead of converging to a lower value, sum of squares error has increased throughout the iterations as the algorithm overshooted the minimum of the loss function.

Additionally for a lower learning rate the train and test error decreased and converged to the lower minimum while for higher learning rate of 10^{-2} they both stayed at constant rate without converging to an optimal value.

5 7.3: Explain in the report your experiment design, comparative result analysis and interpretation of obtained results. Try to be thorough in your analysis.

I have carried out the same experiment for binary classification of subjects 1 and 30 with GD and SGD approaches.

The main difference between stochastic gradient descent (SGD) and gradient descent (GD) is the way they update the model parameters during training.

In GD, the model parameters are updated by computing the gradient of the loss function with respect to the entire dataset and then taking a step in the direction of the negative gradient. This means that in each iteration, the model

parameters are updated based on the average gradient of the loss over the entire dataset.

On the other hand, in SGD, the model parameters are updated based on the gradient of the loss function with respect to a randomly selected subset of the data (also called a mini-batch). This means that in each iteration, the model parameters are updated based on the gradient of the loss function on a smaller subset of the data, which introduces randomness in the update process.

In general, the updates to the model parameters in SGD are more noisy and stochastic compared to GD. This can result in slower convergence towards the optimal set of parameters, but can also help avoid getting stuck in local optima and generalize better to unseen data.

The main difference I have noticed with my experiment is that SGD was handling lower learning rates than GD. For example, at 10^{-3} GD was not able to converge the sum of squares error to the minimum and testing and training errors were constant but SGD was still converging the sum of squares error and test and train errors were being minimised. However SGD needed 500 iterations in order to converge the sum of squares instead 200 iterations