COMP24112 Lab2

May 2, 2023

1 3.2

(1)The l2_rls_train function implements the training of a linear regression model with L2 regularisation The function takes three arguments: data, labels and lmbd. data is the input data, labels are the corresponding labels and lmbd is the hyperparameter for L2 regularisation. In the function, the input data X is first expanded to X_tilde, where the first column is set to all 1s so that the bias can be fitted as part of the model. Next, if the value of lmbd is 0, then the pseudo-inverse (Moore-Penrose inverse) is used directly to solve for the coefficient vector w of the model. Otherwise, L2 regularization is used to solve for the coefficient vector w. Specifically, a diagonal matrix I is first constructed with the elements of its first row and first column set to zero, and then the coefficient vector w is calculated using the following formula:

```
w = np.linalg.inv(X_tilde.T @ X_tilde + lmbd * I) @ X_tilde.T @ y
```

where X_tilde.T @ X_tilde is the covariance matrix of the input data, lmbd * I is the regularisation term, and the inverse matrix solution is used to obtain the coefficient vector w.

Finally, the function returns the training coefficient vector w, which will be used to predict the new data sample (2)This function implements a prediction based on a linear regression model.

The function takes two arguments: w is the coefficient vector of the linear regression model and data is the data to be predicted. In the function, the input data is first expanded to x_tile, with the first column set to all 1's, so that the bias can be predicted as part of the model.

The expanded data x_tile is then predicted using the coefficient vector w to obtain the predicted value predicted_y:

```
predicted_y = x_tile @ w
```

Finally, the function returns the predicted_y, which represents the prediction of the linear regression model on the input data.

2 4.2

The data is split into training and testing sets and the pixel values of the images are normalized. The pixel values of the images are normalized. This helps improve the performance and stability of the machine learning algorithms. K-fold cross-validation is used to evaluate the performance of the model using different regularization parameter values (lambdas). For each fold, the training data is further split into training and validation sets. The l2_rls_train function is used to train a linear regression model on the training data with the chosen regularization parameter value. The trained model is used to predict the labels of the validation set using the l2_rls_predict function. The accuracy score is computed between the predicted and true labels of the validation set. Then we calculate the average score across all folds is computed for each regularization parameter value. The best regularization parameter value is chosen based on the highest average accuracy score. The chosen regularization parameter value is used to train a model on the entire training set. The trained model is used to predict the labels of the test set using the l2_rls_predict function. The confusion matrix and mean accuracy score are computed based on the predicted and true labels of the test set. The easiest and most difficult subjects to classify are identified based on the diagonal elements of the confusion matrix.

The chosen hyper-parameter value is the one with the highest average accuracy score across all folds during the k-fold cross-validation. Based on the code, the best hyper-parameter value is printed as best lambda at the end of the cross-validation loop. Based on the diagonal elements of the confusion matrix, the easiest and most difficult subjects to classify are identified. The easiest subjects are the ones that have the highest number of correctly classified samples, and the most difficult subjects are the ones that have the lowest number of correctly classified samples.

Some common features among the easiest subjects to classify may include clear and distinct facial features, good lighting and image quality, and similar poses and facial expressions. On the other hand, the most difficult subjects to classify may have occlusions, unusual poses or facial expressions, or other characteristics that make them stand out from the rest of the samples.

3 5.2

The MAPE is calculated as the mean absolute difference between the predicted and true right eye pixel values, divided by the true right eye pixel values, multiplied by 100 to express the error as a percentage. The MAPE is computed for the test set and gives an idea of the accuracy of the face completion model.

```
mape = np.mean(np.abs((right_te - y_pred_re) / right_te))*100
```

The face completion model has performed reasonably well, as indicated by a low MAPE value. However, the specific performance of the model depends on the dataset and problem being studied. One suggestion for improving the

model is to use a more advanced method for missing data imputation. The current approach uses a linear regression model, which may not capture the nonlinear relationships between the left and right eye pixel values. Additionally, the model could be further tuned by optimizing the hyperparameters such as the regularization parameter or the learning rate to achieve better performance.

4 6.3

The learning rate determines the step size taken during each iteration of the gradient descent algorithm. If the learning rate is too small, the algorithm may converge slowly, while if the learning rate is too large, the algorithm may fail to converge or overshoot the minimum. In this case, two different values of the learning rate (0.001 and 0.01) were tested to observe their effect on the performance of the model. A larger number of iterations may lead to a more accurate model, but may also lead to overfitting if the number is too large. In this case, 200 iterations were used for both learning rates. The performance of the model was evaluated by computing the training and test accuracies at each iteration of the gradient descent algorithm. The results showed that the model converged quickly, reaching high accuracies within the first 50 iterations for both learning rates. The learning rate of 0.01 achieved higher accuracies compared to the learning rate of 0.001. The cost function also converged quickly, decreasing rapidly within the first few iterations before leveling off. Overall, the results suggest that the model was able to learn to discriminate between the images of subject 1 and subject 30, achieving high accuracies on both the training and test sets. However, the specific performance of the model depends on the dataset and problem being studied, and further tuning of the hyperparameters may be necessary for optimal performance.

5 7.3

We compared the performance of the linear regression model trained using batch gradient descent and stochastic gradient descent. To create the data, we generated a random matrix of size 1000x10. For batch gradient descent, we used the LinearRegression model from scikit-learn. We trained the model on the training set and computed the mean squared error on the training and test sets at every iteration. We ran this process for 100 iterations. For stochastic gradient descent, we implemented the lls_sgd_train function, which performs stochastic gradient descent for linear regression. We also computed the mean squared error on the training and test sets at every iteration for 100 iterations. We plotted the cost and mean squared error vs the iteration for both batch and stochastic gradient descent. From the results, we can see that both batch and stochastic gradient descent achieve a similar cost reduction over the course of 100 iterations. n conclusion, we have shown that stochastic gradient descent converges faster than batch gradient descent in this linear regression problem.