

ML-Lab3-Report

Ziyi Wang

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.

Function `l2_rls_train`: I added a hyperparameter `lmbd(λ)` with a default value of zero, which is used in the regularised term. The function support both single-output and multi-output cases.

When λ is zero, use a pseudo-inverse to calculate the weight: $w = \tilde{X}^\dagger y$.

\tilde{X} is obtained by expanding training sample dataset X with a column of ones.

\tilde{X}^\dagger is the pseudo-inverse, computed by the built-in method `np.linalg.pinv()`.

y is the label matrix of the training data.

Otherwise(λ is not zero), the optimal weight vector is calculated by:

$$w = (\tilde{X}^T \tilde{X} + \lambda I)^{-1} \tilde{X}^T y.$$

I is the identity matrix, whose shape is the same as $\tilde{X}^T \tilde{X}$.

The inverse of the matrix is computed by the built-in method `np.linalg.inv()`.

Function `l2_rls_predict`: using the trained weights to predict the label of the query data. The predicted label is computed by: $\text{predicted_y} = \tilde{X}w$.

\tilde{X} is obtained by expanding training data with a column of ones.

w is the weight obtained by the function `l2_rls_train`.

4.2

(1) Explain the classification steps and report your chosen hyper-parameter and results on the test set.

Classification steps:

- 1) Splitting the entire dataset into a training set and a testing set.
- 2) Process multi-class labels using one-hot encoding.
- 3) Select the hyperparameters using random subsampling. The value of lambda ranges from 0 to 50 in steps of 0.5. For each value of lambda, run 10 trials.
 - i) In each trial, randomly select 50 samples as sub-testing (validation) datasets and the rest as sub-training datasets. The labels of these sets should be multi-class labels.
 - ii) After training, a weight matrix of the shape (1025, 40) should be obtained. Multiply this with the sub-test data to obtain the prediction matrix.
 - iii) The shape of the prediction matrix is (50,40), where 50 is the number of sub-test samples and 40 is the predicted value for each subject. For each row, pick the one predicted value that is closest to 1 as the final predicted class for that sample.
 - iv) The error rate was calculated by dividing the number of incorrect predictions by the total number of predictions. The mean of these ten error rates was calculated and recorded.
- 4) The final hyperparameter is then the one with the lowest mean error rate among the 50 different values.

Final selected λ : 3.5

- 5) Then this lambda was used in evaluate step. Get the new weights using the selected lambda, then, make a prediction to the testing dataset and print the confusion matrix.

Final error rate: 0.045



Misclassified image index: [2 23 24 61 63 64 97 126 189]

Note: This method introduces random variation in the model. It may result in a different lambda value being chosen each time.

(2) Did you notice any common features among the easiest and most difficult subjects to classify? Describe your observations and analyse your results.

Easy recognise: I randomly select 5 subjects which are predicted 100% correctly. Here, I get subject [22 33 17 27 28] (index starting from 1).

I find that the features are obvious in some of the pictures. For example, subject

17  and subject 28.  They both wear glasses and have beards, but the shape of their beards is very different.

Difficult to classify: I print out all the subject that is being misclassified: [1 5 13 20 26 38] (index starting from 1).

Some look so similar to others that it is difficult to tell them apart and therefore are likely to be misclassified. Some are from the same person but look very

different. For example, subject 20.



Some of the pictures are with glasses on, some without glasses and with funny expressions, resulting in a very different look. It is therefore likely that confuses the model, making it difficult to distinguish.

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.

The value of MAPE calculated is approximately 0.22. It's an accuracy measure based on the relative percentage of errors.

Although the MAPE is low, the image completed is not very well. For some front-facing faces, the predictions are a little blurred, but a complete face can be roughly formed. However, for some slightly sideways-facing faces, the model's predictions are not as good, and its completed image looks strange.

In this experiment, I use the left half of the face as training data and the right half of the face as the labels. So, I need to predict the right-half face using the left-half face. The weight is calculated by the *l2_rls_train* function in 3.1, predictions are made by the *l2_rls_predict* function in 3.1, and the λ is simply set to zero.

Suggestion: Select a good hyperparameter for the model.

6.3 Analyse the impact that changing the learning rate has on the cost function and obtained testing accuracies over each iteration in experiment 6.2.

From the first experiment, when the learning rate is 10^{-3} and the iteration number is 200, the Sum-of-square error loss continuously decreases towards 0, and when the number of iterations is approximately greater than 120, the error loss is almost 0. The classification accuracy for test samples is one after 20 iterations, which is very high.

However, when the learning rate is 10^{-2} and the iteration number is 200, the program got a runtime warning that there is an overflow. This means that there is a very large sum-of-square error loss (cost), which Python cannot handle. This learning rate is set too high.

What are the consequences of setting the learning rate and iteration number too high or too low?

Learning rate too high: leads to large update steps, the gradient descent algorithm may not converge to the optimal solution, or it may bounce around the optimal solution, taking long iterations.

Learning rate too low: leads to small update steps and the algorithm may converge very slowly, requiring long iterations to find the optimal. It is also possible to get stuck in a local minimum rather than converging to the global minimum.

Iteration number too high: the model may overfit the training samples. Besides, if the number of iterations is already sufficient to get the optimal solution and if it continues to iterate, then its performance may not improve any further.

Iteration number too low: the model may not learn enough and results in underfitting.

7.3 Explain in your report the following:

(1) Your implementation of `hinge_gd_train`. If you analytically derived the loss function, please include it here.

The hinge loss function with regularisation term is:

$$loss = C \sum_{i=1}^N \max(0, 1 - y_i f(\theta, x_i)) + \frac{1}{2} w^T w$$

C is the regularisation hyperparameter.

$\sum_{i=1}^N \max(0, 1 - y_i f(\theta, x_i))$ is the hinge loss.

$\frac{1}{2} w^T w$ is the regularisation term.

y_i is actual (ground truth) class label, which is $\in \{-1, 1\}$.

$f(\theta, x_i)$ is the prediction model.

Rewrite the max function: $\max(0, 1 - y_i f(\theta, x_i)) = \begin{cases} 0, & \text{if } y_i f(\theta, x_i) \geq 1 \\ 1 - y_i f(\theta, x_i), & \text{otherwise} \end{cases}$

where $f(\theta, x_i)$ in here is: $f(\theta, x_i) = w \cdot x$.

To find new weight, take derivative of the total hinge loss. Gradient of each component:

$$\frac{\partial \max}{\partial w} = \begin{cases} 0, & y_i \times w \cdot x \geq 1 \\ -y_i \times x, & \text{otherwise} \end{cases}$$

$$\frac{\partial loss}{\partial w} = C \times \frac{\partial \max}{\partial w} + w = C \times \left(\sum_{i=1}^N \begin{cases} 0, & y_i \times w \cdot x \geq 1 \\ -y_i \times x_i, & \text{otherwise} \end{cases} \right) + w$$

The new weight is obtained using: $w_{new} = w_{old} - learn_rate \times \frac{\partial loss}{\partial w}$.

(2) Your experiment design, comparative result analysis and interpretation of obtained results.

I. Splitting the dataset, 3 images from each subject are used for training and the remaining 7 images for testing.

II. Encoding the label. -1 for subject1 and +1 for subject30.

III. Hyper-parameter selection: using Leave one out(LOO), i.e. N-fold cross validation (where N is the number of the training samples). Because there are only 6 data in the training samples, which is very limited. The value of C ranges from 0.1 to 1.0 (included) in steps of 0.1. For each C, perform a 6-fold CV (6 loops).

For each loop, use 5 partitions for training and the remaining 1 (validation set) for estimating the error rate E_i . And in each loop, I could get two NumPy arrays: $cost_all$, and w_all .

(i) $cost_all$ is a record of the hinge losses over N iterations. I pick the lowest hinge loss as the final hinge loss for the current c.

(ii) w_all is the record of the weights over N iterations, likewise, after calculating the accuracy of these weights, I choose the highest accuracy as the accuracy of the current c.

After completing the loops, I should obtain a list of error rates and a list of hinge losses for different c, from which I select the value of c with the lowest error rate and the lowest hinge loss.

From my experiment, when the learning rate is 0.001 (l_rate1 , this name will be used in part V),

the final error rates are: [0.83, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],

the final error losses are:

[0.213, 0.155, 0.134, 0.135, 0.137, 0.139, 0.142, 0.145, 0.147, 0.146].

The selected C of the lowest error rate is 0.2 and of the lowest error loss is 0.3. From the above lists, we could see that the error rate is all zero when C is greater and equal to 0.3. Therefore, I decided to select 0.3 as the value of C.

IV. Retrain and evaluate. Then, I retrain the sample to get a new list of weights and hinge loss using the selected C. Calculating the training and testing accuracy and drawing the graph.

We can see from the graph that the error loss is gradually decreasing, with a small fluctuation in the later stages. After about 80 iterations, the training accuracy is stable at 1. The test accuracy is not very stable but fluctuates between 0.85 and 1, which shows that the model is very well-trained.

V. Compare with a larger learning rate.

Set the learning rate to 0.01(l_rate2). The value of the selected C is 0.1. As previously, I also draw three graphs. Then I compared the hinge loss and training and testing accuracy for different learning rates and made compared graphs.

- a) Hinge loss: Although both hinge losses are close to zero, it is obvious that the hinge loss after 160 iterations of l_rate1 is much more stable than that of l_rate2 . This is because a higher learning rate will have a larger update step, which leads to a greater amplitude in jumping around the optimal solution. And for l_rate1 , the hinge loss becomes relatively stable after about 200 iterations, whereas for l_rate2 , it only takes about 40 iterations. Again, this is also because the higher learning rate has a larger update step, leading to fast learning.
- b) Training Accuracy: From the graph, we could see that both training accuracies are stable at one finally, but the accuracy for l_rate2 reaches one earlier than l_rate1 . This is because a higher learning rate will lead to fast learning.
- c) Testing Accuracy: Although the testing accuracies of both learning rates are close to one, it is obvious that the testing accuracy of l_rate2 is much more volatile than that of l_rate1 . The reason for this is the same as (1).

VI. Compare with least squared loss.

I set the learning rate to 0.001 and the iteration number to 300. The value of C for the hinge loss method is 0.4. Then I trained the model and made compared graphs.

- a) Hinge loss: Although both final error losses are close to zero, it is obvious that the hinge loss is much smaller than that of the least squared loss. Because the least squared loss function penalises the difference between the predicted and true values of the model, squared. While the hinge loss function only penalizes misclassifications and allows for some error in the predictions.
- b) Training Accuracy: From the graph, we can see that both training accuracies are stable at 1, but the accuracy for least squared loss reaches earlier than hinge loss. Because the hinge loss places a higher penalty on misclassifications, which may lead to a slower rate of convergence and a slower increase in training accuracy. Besides, the least squared loss is a smoother and more continuous function than the hinge loss, which means it may be easier to optimize using gradient descent.
- c) Testing Accuracy: The final testing accuracies of least squared loss is stable at 0.85, but that of hinge loss fluctuates between 0.89 and 0.93. One potential reason for this is the over-fitting of the model to the training data. In addition, the function of the hinge loss is non-smooth, which has multiple local optima, meaning that the gradient descent algorithm may converge to different optima depending on the initialisation and hyperparameters. If the model converges to a different optimum each time it is trained, then the test accuracy may be unstable.