Face Recognition

A. Khabir

May 2, 2023

1 Implementation

The implementation is based on the concept of linear regression, which is a method used to model the relationship between a dependent variable and one or more independent variables. L2 regularization is introduced to prevent overfitting by adding a penalty term proportional to the squared magnitude of the model parameters to the loss function.

1.1 Design of l2_rls_train Function

The `l2_rls_train` function takes three arguments:

- `data`: The input data matrix.
- `labels`: The target output matrix.
- `lmbd`: The regularization parameter lambda controlling the trade-off between minimizing the loss function and the complexity of the model.

The steps in the function are:

1. Expanding the input data with a column of ones to account for the bias term.
2. Conditionally computing the model parameters based on the value of lambda:
    - When lambda is zero, using the pseudo-inverse to obtain the model parameters.
    - When lambda is non-zero, computing the model parameters using L2 regularization.
3. Returning the model parameters.

1.2 Design of l2_rls_predict Function

The `l2_rls_predict` function takes two arguments:

- `w`: The model parameters, including the bias term.
- `data`: The input data matrix for which the output is to be predicted.

The steps in the function are:

1. Expanding the input data with a column of ones to account for the bias term.
2. Computing the predicted output by multiplying the expanded input data with the model parameters.
3. Returning the predicted output.

1.3 Role of Additional Arguments

The regularization parameter lambda (`lmbd`) is an important argument that controls the trade-off between model complexity and overfitting. By tuning the lambda value, we can find the optimal balance between fitting the training data and generalizing to unseen data.

2 Face Recognition

Classification steps:

a. Load the dataset and preprocess the data by splitting the images into left and right halves.

b. Partition the data into training and test sets.

c. Perform hyperparameter tuning by selecting the best regularization parameter lambda using k-fold cross-validation.

d. Train the L2-regularized least squares model on the training data with the chosen lambda.

e. Predict the output labels for the test data using the trained model.

f. Evaluate the model performance by computing the confusion matrix and classification accuracy.

Chosen hyper-parameter: The best regularization parameter lambda selected using k-fold cross-validation is 0.0001.

Results on the test set: The mean classification accuracy across different lambda values during cross-validation is around 0.86. The confusion matrix provides a more detailed view of the model's performance on individual subjects.

Analysis: It seems the images that was the easiest to classify were taken from the same angle and had the subject with relatively same facial expressions. The images that were the hardest to classify were the ones taken from different angles and had pose variations. This difficulty in classifying them could be because the model is not complex enough to capture these variations.

3 Face Completion

The L2-regularized least squares face completion model yielded a Mean Absolute Percentage Error (MAPE) of 0.27. Upon visual inspection of the ground truth and completed faces, it was observed that the model produced blurry images. Despite these limitations, the model did manage to generate a rough approximation of the face structure. To improve the performance of the face completion model, a deep learning-based approach could be employed, such as convolutional neural networks (CNNs), Variational Autoencoders (VAEs), or

Generative Adversarial Networks (GANs). These models have the ability to learn more complex features and patterns in the data, which could potentially result in better face completion performance and more accurate reconstructions of the right side of the face based on the left side.

4 Gradient Descent

In experiment 6.2, the impact of changing the learning rate on the cost function and testing accuracies over each iteration can be analyzed as follows:

When the learning rate is too low (e.g., 1e-3), the model converges slowly, and it takes more iterations to achieve optimal or near-optimal weights. However, the training process is stable, and the model is less likely to overshoot the optimal weights. This results in a smooth decrease in the cost function and stable training and testing accuracies over each iteration.

When the learning rate is too high (e.g., 0.01), the model converges faster, but it is likely to overshoot the optimal weights. This causes oscillations in the training and testing accuracy graphs, and instability in the learning process is observed. The cost function may also show sudden spikes, indicating that the model has moved away from the optimal weights.

Setting the learning rate too low can cause the the model to converge slowly, which may require more iterations to achieve good performance. The training process may take longer, increasing computational time.

Similarly, setting the iteration number too low can lead to an undertrained model, which may not have reached its optimal performance. On the other hand, setting the iteration number too high can lead to overfitting or increased computational time without significant performance improvement.

5 Advanced Gradient based training

The hinge loss function is defined as $\max(0, 1-y*(w^T@x))$. The `hinge_gd_train` function trains a linear classifier by minimizing the hinge loss using the gradient descent approach. The function takes as input the data, labels, learning rate, number of iterations, regularization hyperparameter (C), and optional test data and test labels.

In the function, the input data is first normalized by dividing by 255. Then, the bias term is added to the data by appending a column of ones. The weight vector w is initialized to zeros. The cost and weight arrays are also initialized to store the values for each iteration.

The gradient descent algorithm is run for the specified number of iterations. In each iteration, the margin is calculated as $y * (X_{tilde}@w)$. The gradient of the hinge loss is computed as $-np.mean((margin < 1) * y * X_{tilde}, axis = 0).reshape(-1, 1)+w/C$. The weight vector w is updated as $w = w-\eta*gd$. The hinge loss is computed as $np.mean(np.maximum(0, 1-margin))+np.sum(w* *2)/(2 * C)$. The cost and weight values are stored for each iteration. If test data and labels are provided, the test loss is computed and stored as well.

The low training and test accuracies for both models suggest that the linear classifiers are not able to effectively fit the given dataset. The choice of hyperparameters (learning rate and number of iterations) might be suboptimal for

both models, and experimenting with different values might help improve the performance.

The hinge loss's constant value at 0 indicates that the model is perfectly classifying all the training samples which maybe due to overfitting.