COMP337 Assignment 1 Project Report

Wuwei Zhang, 201522671, sgwzha23@liverpool.ac.uk

1. Task 1: Perceptron Algorithm for Binary Classification

Algorithm 1 shows the pseudo-code of the training stage of single perceptron to perform binary classification. This algorithm initialize the weights W and bias b with 0, the length of weight is same as the dimension for each element in train_sample X. Then we use our train_sample to predict the corresponded score pred, if both the true label y and predicted score pred are positive or negative, we will look at next element. Otherwise, we update the weights and bias. Once we finish look through all element, we will repeat this procedure for n_epoch times. Algorithm2 shows the procedure for predict the class of the sample. If the pred score smaller than 0, we label it as 1 otherwise -1.

Algorithm 1 Train:

Input: train_sample X, train_label Y, epoch num_epoch, learning_rate lr
$w = 0$ for all w in W and i = $1...len(x)$
$b = 0$
for $epoch$ **to** num_epoch **do**
  SHUFFLE X
  for x in X **do**
    $pred = W.T.dot(x) + b$
    if $y * pred < 0$ **then**
      $W = W + y * x_i * lr$
      $b = b + y * lr$
    end if
  end for
end for
return W, b

Algorithm 2 Predict:

Input: weights W, bias b, sample x
$pred = W.T.dot(x) + b$
if $pred > 0$ **then**
  Label x as 1
else
  label x as -1
end if

2. Task2: Implement binary perceptron

The program uses load_data() to load train and test set and label class 1, 2, 3 to 0, 1, 2. For binary classification, I use np.where(_train_y != i) to remove the class that doesn't to be classified, then, I use set_to_target() to clip label for remaining classes to {-1, 1}.
The forward propagation algorithm is implemented in single_perceptron() and the training algorithm is implemented in train_perceptron(). For binary classification, the predicted class is 1 if the output is greater than 0 and -1 otherwise. For multiclass classification, the predicted class is the class with the highest output by using np.argmax().

3. Task3: Train and test binary perceptron

With random seed equal to 1 and shuffle training data before each epoch. The number of weight/bias updates for classification 'class 2 and 3' is 340 which is much greater than 'class 1 and 3' and 'class 1 and 2'. Moreover, the training and testing accuracy for 'class 2 and 3' is 53.75% and 50.00%, the others' actuaries are all 100%. Moreover, by modifying random seed, learning rate and epoch to different value, this result is still similar. Therefore, class 2 and 3 is most difficult to separate.

4. Task4: Multi-class classification with perceptron

In this task, I trained 'class 1 vs rest', 'class 2 vs rest' and 'class 3 vs rest' three weights-bias pair in total. In testing stage, unlike the binary classification, this classification strategy directly compute the W.T.dot(x) + b for each weights-bias pair for each sample as the score, then label the sample by applying np.argmax() to score, the higher score means higher probability to be classified as this class. However, the train and test accuracy both are only 66.67%.

5. Task5: L2 Regularisation

By changing weight update function to W += lr * y * x - lr * l2_reg * W in train_perceptron(), this project can apply the L2 regularisation easily. With random seed 1 to shuffle the data before each epoch and setting learning_rate to 1e-2, this step uses 6 different regularisation coefficients. to train the perceptron. When l2_reg equals 1e-2, the train accuracy 91.67% and the test accuracy is 96.67%, which performs best over all different regularisation coefficients. By using larger coefficients, the testing and training accuracy are all instantly decreasing. For l2_reg equals to 10 and 100, the number of weight/bias updates is very high but the testing and training accuracy in these two cases are only 33.33%. However, by using smaller learning_rate, the larger regularisation coefficients may have better performance.

6. Shuffle matters

In the early stage of implementation, I didn't use shuffle to prevent over-fitting therefore the accuracy is very low in that case. However, if removing the random seed from code, the performance of shuffle is not very stable, sometimes it has lower accuracy than training without shuffle.