Student ID – 201669264 Name: Rahul Nawale

Question 1: Answer

Explain the Perceptron algorithm (both the training and the test procedures) for the binary classification case

The perceptron algorithm is a linear classification algorithm that is used for binary classification problems, where the goal is to separate two classes by finding an optimal hyperplane that divides them. In this algorithm, the input features are multiplied by their corresponding weights, and the resulting values are added together to form a weighted sum. This sum is then passed through an activation function that produces a binary output, indicating which class the input belongs to.

The perceptron algorithm is based on the concept of a single artificial neuron or perceptron, which takes multiple inputs, applies weights to them, and produces a single output. The algorithm can be represented by the following steps:

Training Procedure:

1. **Initialize the weights and bias**: Start by setting the initial weights and bias to small random values or zeros.
2. **Compute the weighted sum**: For a given input vector x, compute the weighted sum of the inputs and the weights, as follows:
   z = w.T * x + b
   where w is the weight vector, b is the bias term, and T denotes the transpose operator.
3. **Apply the activation function**: Pass the weighted sum z through an activation function to obtain the output y, which is the predicted class label. In the binary case, the activation function is typically a step function or a sign function, as follows:
   y = sign(z) = sign(w.T * x + b), where sign is the signum or step function.
4. **Update the weights**: If the predicted output y does not match the true output t, update the weights and bias term by adding the product of the learning rate and the error to them, as follows:
   w = w + α * (t - y) * x
   b = b + α * (t - y), where α is the learning rate, t is the true class label, and x is the input vector.
5. **Repeat**: Repeat steps 2-4 for each input vector in the training set until the algorithm converges or a maximum number of iterations is reached.

Testing procedure:

1. **Given a new data point x**, calculate its predicted label y_pred using the learned weights w and bias b as follows:
   y_pred = sign(w.T*x + b)
2. **If y_pred is greater than or equal to 0**, classify the input as belonging **to class 1**.
3. **If y_pred is less than 0**, classify the input as belonging **to class 0**.

In summary, the perceptron algorithm is a simple yet powerful linear classifier that is suitable for binary classification problems. The algorithm works by iteratively adjusting the weights and bias term based on the difference between the predicted and true class labels, until it converges or reaches a maximum number of iterations. The algorithm can be extended to handle multiclass classification problems by using one-vs-all or one-vs-one approaches.
Important: Perceptron algorithm only converges if the input data is linearly separable. If the data is not linearly separable, the algorithm may never converge or converge to a suboptimal solution.

Pseudo code:

Train

```
PerceptronTrain(Training data: D, MaxIter)
1. Initialize weights to 0 for all i = 1, 2, ..., d
2. Initialize bias to 0
3. for iter = 1 to MaxIter do
4.    for all (X, y) in D do
5.       a = dot product of weights and X + bias
6.       if y * a <= 0 then
7.          for i = 1 to d do
8.             weights[i] = weights[i] + y * X[i]
9.          bias = bias + y
10. return bias and weights
```

Test

```
PerceptronTest(bias, w1, w2, ….., wd, X)
1. Input of the test algorithm is Bias, learned weights and new(unseen) input vector X
2. a = dot product of weights and X + bias
3. return sign(a)
```

In this algorithm, D is the training dataset, MaxIter is the maximum number of iterations, X is the input vector, y is the true label of the input, d is the number of features in X, and a is the activation function. The algorithm initializes the weights and bias to 0, and then iterates through the training data for MaxIter number of iterations. For each input X and label y, the algorithm calculates the activation function a and updates the weights and bias if the prediction is incorrect. Finally, the algorithm returns the learned weights and bias. For testing dataset, for each input data X, it calculates activation function using learned weights and bias during training and returns the sign(a) i.e classifies the input X as class 0 or 1.

Question 2: Answer

Explaination of the algorithm in Source Code (.py file) step by step:

1. perceptron_train(data, max_iter) takes two arguments: the training data data, which is a list of tuples (X, y) where X is a list of features and y is the target label (either 1 or -1), and max_iter which is the maximum number of iterations to run the algorithm for.

2. First, the algorithm initializes the weights and bias to zero.
Then, it runs a loop for max_iter iterations.

3. In each iteration, the algorithm loops over all the training examples (X, y) in data.
For each training example, it calculates the activation function a as the dot product of weights and X, plus the bias.

4. If the training example is misclassified (i.e., y * a <= 0), then the algorithm updates the weights and bias according to the perceptron update rule: wi = wi + y*xi for all i, and bias = bias + y.

5. Finally, after all iterations are complete, the algorithm returns the final bias and weights.

6. The perceptron_test(bias, weights, X) function takes the bias and weights obtained from perceptron_train and a new input vector X, and returns the predicted label for X.

7. It calculates the activation function a for X using the same dot product formula as in the training algorithm, and then returns the sign of a, which is either 1 or -1 depending on whether the prediction is positive or negative.

Question 3: Answer

Accuracies after 20 iterations:

Class 1 vs Class 2:
Train accuracy: 99.38%
Test accuracy: 100.00%

Class 2 vs Class 3:
Train accuracy: 81.37%
Test accuracy: 77.50%

Class 1 vs Class 3:
Train accuracy: 94.38%
Test accuracy: 89.74%

Which pair of classes is most difficult to separate?

Class 2 vs Class 3 pair is most difficult to separate with a training accuracy of 81.37% and a testing accuracy of 77.50%.

This indicates that the classifier struggles to differentiate between instances belonging to Class 2 and Class 3. In contrast, the other two pairs of classes (Class 1 vs Class 2 and Class 1 vs Class 3) have higher training and testing accuracies, indicating that the classifier is better able to distinguish between those classes.

Question 4: Answer

One vs rest approach

The 1-vs-rest approach is a classification strategy used when dealing with multi-class classification problems. In this approach, a classifier is trained for each class, where each classifier is responsible for distinguishing between one class and the rest of the classes.

The 1-vs-rest approach is a strategy for multi-class classification problems, where we aim to classify an incoming object **X** into one of **k** possible classes. In this approach, we assume that the binary classification algorithm A can output a numeric score representing its "**confidence**" that an object belongs to a particular class. **For example:**

1. To classify an object X using this approach, we first train **k binary classifiers (A1, A2, ..., Ak)**, where each classifier is trained with the objects of **class "i"** treated as positive samples and all other objects as negative samples.

   This results in k prediction models.
2. When an incoming object X needs to be classified, we apply all k prediction models (A1, A2, ..., Ak) and obtain k numeric scores representing the "confidence" that X belongs to each of the k classes. We then output the class label y corresponding to the model with the highest score, **i.e., y = $\text{argmax}_{i\in\{1,2,\ldots,k\}}$ Ai(X).**

3. The choice of the numeric score depends on the classifier at hand. For example,
   for Perceptron, the **activation score "a" = $b + W^T*X$** is used, while for logistic regression, the sigmoid function is applied to the activation score $a = b + W^T*X$ to obtain the "confidence" score.

The 1-vs-rest approach is a simple and effective way to extend binary classification algorithms to multi-class classification problems. However, it may not always be the best approach, as it can suffer from class imbalance and can be sensitive to the choice of hyperparameters.

Multi-class - Train and Test accuracies after training for 20 iterations

Train accuracy: 89.2116182572614

Test accuracy: 89.83050847457628

Question 5: Answer

Results are not reported here as it is not mentioned in the report to do so. Results are output of the python code.

Regularization is used to prevent overfitting of the model by adding a penalty term to the loss function, which encourages the model to have smaller weights.

Comments on the result

1. The initial multi-class perceptron model has a training accuracy of 89.2% and a testing accuracy of 89.8%. After adding L2 regularization to the model with different regularization coefficients, the training and testing accuracies decreased significantly.

2. From the results obtained above, we can see that as the **regularization coefficient increases, both the training and testing accuracies decrease**. This is expected since the penalty term in the loss function is proportional to the size of the weights.

3. At a very low regularization **coefficient of 0.01**, the model has high training and testing accuracies, which suggest that the model is underfitting. On the other hand, as the coefficient increases **to and above 0.1 up to 100**, the model has low training and testing accuracies, which suggest that the model is overfitting. As the regularization coefficient increased, the performance of the model continued to deteriorate, suggesting that the penalty term was becoming too large and the model was becoming too simple.

4. **Increasing the number of iterations** may not necessarily increase the accuracy as the model may start to overfit to the training data. Instead, it is essential to tune the regularization coefficient to find the optimal balance between preventing overfitting and preserving model performance.

Overall, these results suggest that the **initial model without regularization was the most effective for this dataset**, and the addition of the L2 regularization term did not improve the performance of the model. The **optimal value of the regularization coefficient** depends on the specific problem and dataset, and can be found through **hyperparameter tuning techniques such as cross-validation**.