Coursework 1: Perceptron Algorithm

1.) (15 marks) Explain the Perceptron algorithm (both the training and the test procedures) for the binary classification case. Provide the pseudo code of the algorithm. It should be the most basic version of the Perceptron algorithm, i.e., the one that was discussed in the lectures.

For binary classification tasks, the Perceptron algorithm is a supervised learning technique. It is based on a linear model with a hyperplane as the decision boundary. Training and testing are the two phases of the algorithm.

It is made up of a single node or neuron that predicts the class given a row of data as input. By computing a weighted sum of the input and bias, this is accomplished (set to 1). The activation of a model is the weighted sum of its inputs: Activation = Weights * Inputs + Bias, where $a = w_n * x_n + b$.

Training Procedure:

1. The model's weights are initialised to zero values at the start of the training procedure.
2. Calculate the weighted sum of the inputs and the current weights for each training sample. The output of the Perceptron is then created by passing this weighted sum through an activation function.
3. The weights are left alone if the Perceptron's projected output matches the actual output. However, the weights are changed in a way that will lessen the mistake if the expected output is inaccurate. The magnitude of the weight modification is done using a learning rate that establishes the learning rate.
4. Up until the 20 iterations, steps 2 and 3 are repeated for each training sample.

Test Procedure:

1. Initialize the weights: At the beginning of the test process, the weights for the model are initialized to the values obtained after the training process.
2. For each test sample, calculate the weighted sum of the inputs and the current weights. This weighted sum is then passed through an activation function, which produces the output of the Perceptron.
3. Use the output of the Perceptron to classify the input sample into one of two categories.
4. Steps 2 and 3 are repeated for all test samples. The performance of the Perceptron is evaluated based on the accuracy of its classification of the test data.

Pseudo Code

Training Function:

1. Create the perceptron train function, which accepts the parameters features X, labels Y, and num_iter, the maximum number of iterations.
2. Create an array of zeros in the weight vector w that has the same dimensions as X.
3. Put the bias b to zero.
4. Iterate over the num_iter iterations in a loop.
5. Loop through each training example in X once for each iteration.
6. Calculate the weights' dot product with the input example X[i] plus the bias b, then add it to it, and save the result in variable a.
7. Determine the sign of a and store it in the variable y_pred to predict the class label for the input example X[i].
8. Verify that the projected label y_pred and the actual label Y[i] are equivalent. If not, change the bias b and the weight vector w as follows: 1.) w = w + Y[i] * X[i] 1.) b = b + Y[i]
9. Return the learnt weight vector w as well as the bias b.

Test Function:

1. Define a function named perceptron_test that takes in input features X, labels Y, learned weight vector w, and bias b.
2. Compute the dot product of the weights and the input X plus the bias b, and store the result in variable a.
3. Predict the class label for the input examples X by taking the sign of a and store it in variable y_pred.

4. Compute the accuracy of the prediction by counting the number of correct predictions and dividing it by the total number of predictions.
5. Return the predicted class labels y_pred.

Accuracy Function:

1. Define a function named get_accuracy that takes in predicted class labels y_pred and true labels y_test.
2. Initialize acc

2.) (30 marks) Implement a binary perceptron. The implementation should be consistent with the pseudo code in the answer to Question 1.

The implementation of the binary perceptron is in the attached python file.

3.) (15 marks) Use the binary perceptron to train classifiers to discriminate between
• class 1 and class 2,
• class 2 and class 3, and
• class 1 and class 3.

Report the train and test classification accuracies for each of the three classifiers after training for 20 iterations. Which pair of classes is most difficult to separate?

Here is a table presented showing the training and testing accuracies between each class:

Class	Training Accuracy (%)	Testing Accuracy (%)
Class 1 vs Class 2	99.4	100.0
Class 2 vs Class 3	84.5	92.5
Class 1 vs Class 3	95.0	94.9

Based on the table provided, it appears that the Class 2 vs Class 3 comparison has the lowest testing accuracy, which may indicate that this pair of classes is the most difficult to separate.

4.) (30 marks) Explain in your own words what the 1-vs-rest approach consist of. Extend the binary perceptron that you implemented in part 3 above to perform multi-class classification using the 1-vs-rest approach. Report the train and test classification accuracies for the multi-class classifier after training for 20 iterations:

The 1-vs-rest approach, also known as the one-vs-all approach, is a strategy used in multi-class classification problems to classify into one of several possible classes. This approach entails training multiple binary classifiers, one for each class, on a subset of the data that includes instances of the target class and instances of all other classes put together during the training process. The 1-vs-rest approach can be applied to a wide range of classification algorithms, including linear classifiers such as logistic regression and support vector machines, as well as non-linear classifiers such as decision trees and neural networks. It is particularly useful when the number of classes is large, and it becomes computationally expensive to train a classifier for each unique combination of classes.

Furthermore, the 1-vs-rest approach can handle imbalanced datasets where some classes have significantly fewer samples than others. In such cases, we can adjust the class weights in the binary classifiers to give higher importance to the minority classes, thus improving their classification accuracy. However, this approach has some limitations. First, it assumes that the classes are mutually exclusive, meaning that a data point can belong to only one class. If there are overlapping classes, this approach may not be appropriate. Second, the performance of the 1-vs-rest approach heavily depends

on the quality of the binary classifiers, and if any one of the classifiers is poor, the overall performance of the multi-class classifier can suffer. While the 1-vs-rest approach is a quick and efficient method for applying binary classifiers to multi-class issues, it may have a class imbalance and be less accurate than other, more advanced techniques.

*The extension of the binary perceptron that can perform multi-class classification using the 1-vs-rest approach is in the attached python file.

Here is the report for the train and test classification accuracies for the multi-class classifier after training for 20 iterations presented in a table:

Class	Training Accuracy (%)	Testing Accuracy (%)
1 vs Rest	69.3	72.9

5.) (10 marks) Add an regularisation term to your multi-class classifier implemented in part 4. Set the regularisation coefficient to 0.01, 0.1, 1.0, 10.0, 100.0 and compare the train and test classification accuracies. What can you conclude from the results?

This table displays the train and test classification accuracies:

1 vs 1 (Lambda Values)	Training Accuracy (%)	Testing Accuracy (%)
0.01	59.8	54.2
0.1	33.2	33.9
1.0	33.2	33.9
10.0	33.2	33.9
100.0	33.2	33.9

Introducing a regularisation term to the multi-class classifier can assist prevent overfitting and improve its generalisation performance on unseen data. We can vary the strength of regularisation and discover the ideal value that balances model complexity and accuracy by tuning the regularisation coefficient.

We may see the following while comparing the train and test classification accuracies with various regularisation coefficients (0.01, 0.1, 1.0, 10.0, and 100.0), from the results:

- The model gets simpler and more regularised as the regularisation coefficient rises, which may result in a drop in training accuracy but a gain in testing accuracy. This is because the model is more likely to identify the underlying patterns in the data and less likely to fit noise.

- On the other hand, the model becomes more complex and less regularised as the regularisation coefficient falls, which may result in an improvement in training accuracy but a drop in testing accuracy. This is due to the possibility that the model overfit the training set and failed to generalise to updated data.

As a result, we must select the regularisation coefficient that maximises testing accuracy while minimising training accuracy loss.