COMP527
Assignment 1

Student Name: Alice Varley
Student ID: 201685457

Question 1

Algorithm 1 `train_perceptron`(Training data: $\{\bar{X}_1, \cdots, \bar{X}_d\}$, **`MaxIter`**)

```
1: w_i ← 0 for all i = 1, ..., d;
2: b ← 0
3: for iteration = 1, ..., MaxIter do
4:    Shuffle data
5:    for all (X̄, y) ∈ D do
6:       a ← Σ_{i=1}^{d} w_i x_i + b
7:       if y · a ≤ 0 then
8:          w_i ← w_i + y · x_i for all i = 1, ..., d
9:          b ← b + y
10:      end if
11:   end for
12: end for
```

The perceptron algorithm is a binary linear classifier, meaning it learns a decision boundary which separates two classes using a hyperplane in the feature space. Here, we refer to a positive class with a label +1 and a negative class with label -1. The aim is to determine a weight vector that causes the perceptron to produce the correct output for the samples. It is split into a training and testing (or online) mode.

Perceptron algorithm: training procedure

In the training mode, the perceptron takes as input the training data $\{\bar{X}_1, \cdots, \bar{X}_d\}$ of dimension d (having d features), where each object $\bar{X}_i$ consists of a series of feature values $x_1, \ldots, x_d$ and a label $\bar{Y}_i$. Ideally, the training data would be linearly separable to allow the perceptron to properly learn the differences between the classes. The other input is the hyperparameter **`MaxIter`**, which represents the maximum number of times that the algorithm iterates through the dataset (the epochs). This is necessary in order to terminate the algorithm; another method to measure converge would be stipulating some small change in weights which, when reached, the algorithm terminates. The algorithm uses the perceptron update rule to repeatedly update the weights and biases until they reach optimal values.

The weights (one for each dimension) and the bias are initialised as 0. Note that it is also possible to treat the bias as the 0th weight, that is $w_0 = b$. They could also be initialised as random numbers, but this is irrelevant as the algorithm will adjust them as appropriate through the various iterations. The algorithm then begins the outer for loop, which defines how many times the algorithm loops through the data. In each iteration, the data should be shuffled to ensure that certain classes aren't all grouped together (which would mean that the weights aren't adjusted very much and the model will probably learn to predict samples as from one class) leading to a poor model. For this assignment, the data is shuffled once with the same random seed to ensure consistent results.

Then the inner loop begins, which iterates over each sample. It calculates the activation score, which is equivalent to the weight vector multiplied by a vector of feature values for that object plus the bias: $\bar{W}^T \cdot \bar{X} + b$. The sign of this score represents the predicted class for the object (recall that we have defined the two classes as +1 and -1). Therefore, if the activation score a has a different sign to the label, this will mean that their product is negative and the instance has

been misclassified. In this case, the weights and bias are updated to try to improve classification (though there is no guarantee that this adjustment will be sufficient to correctly classify the same object in the next iteration). For each weight, the product of that instance's true label y and the corresponding feature value x is added to its existing value. In other words, a scalar y multiplies the vector of features and is added to the weight vector: $\bar{W}^T = \bar{W}^T + y \cdot \bar{X}$. This is the part that serves to minimise the loss function by adjusting the parameters.

Algorithm 2 `test_perceptron`$(w_1, \ldots w_d, b, x_1, \ldots, x_d)$

```
1: a ← Σ_{i=1}^{d} w_i x_i + b
2: return SIGN(a)
```

Perceptron algorithm: testing procedure

For a given sample object $\bar{X}$ from the test data with feature values $x_i, i = 1, \ldots, d$, the test algorithm is given as input the bias, the weights for each feature (which in practise, would be a weight vector) and the object's feature value for each dimension. As aforementioned, the bias b could alternatively be included in the weight vector as w_0. It computes the activation score discussed above. It returns the sign of the activation signal, which serves as the class label prediction. We can then verify whether the test produced the correct output by comparing the output to the sign associated with the true class label. In practice, we may want to produce a function that iterates through each object inside the function, rather than calling the function inside a loop.

Question 3

Pair of classes	Train accuracy	Test accuracy
Classes 1 & 2	98.75%	100.00%
Classes 2 & 3	79.38%	87.18%
Classes 1 & 3	89.38%	84.21%

Interestingly, the classes with the lowest training accuracy (2 & 3) aren't the ones with the lowest testing accuracy (1 & 3). One possible reason for the testing accuracy in 1 & 2 and 2 & 3 being higher than the training accuracy could is that 20 epochs was not sufficient for perceptron to optimise for the training data, and the testing data being smaller could mean that there is a better chance for a higher accuracy. These figures would suggest that it was harder for the perceptron to adjust the weights for 2 & 3 and that the perceptron overfitted the data for 1 & 3. Therefore, the data suggests that classes 2 & 3 were the hardest to separate (in the training phase).

Question 4

The one-vs-rest approach is a heuristic which applies multiple binary classification algorithms to a multi-class classification problem. The multi-class dataset is split into multiple binary classifiers (one per class). For each classifier, the samples of the class it is trained to detect are classified as positive with an output of +1 and all other samples are classed as negative with an output of -1. Then, for each object, each binary classifier is applied, and the classifier (i.e. class) giving the highest activation score is selected as the predicted class label for that object.

After training for 20 epochs, the multi-class classifier produced an accuracy of **75.00%** on the training data and **72.41%** on the testing data. These are poorer accuracies for both training and testing when compared to those for the binary classifiers. This could be in part due to the fact that perceptron is not overly suited to multiclass classification, being a binary classification algorithm, and could also just be down to the way that the data was shuffled.

Question 5

For both the training and testing data, every regularisation coefficient λ produced a significantly lower accuracy than the classifiers without regularisation. Furthermore, there was no variation in accuracy for either the training or testing data as λ changed. Finally, for $\lambda = 10.0$ and $\lambda = 100.0$, there is a `RunTimeWarning` due to the extreme size of the resulting weights after regularisation.

Regularisation coefficient, λ	Train accuracy	Test accuracy
0.01	33.4%	32.8%
0.1	33.4%	32.8%
1.0	33.4%	32.8%
10.0	33.4%	32.8%
100.0	33.4%	32.8%