Questions for Part 1

Question 1: Refer to the documentation, what is the functionality of the tol parameter in the Perceptron class? (2 marks)

The "tol" defines the stopping criterion for the Perceptron algorithm. If it is not None, the training stops when the loss for the current iteration exceed the loss for the previous iteration by the value specified in “tol” i.e. when current_loss > previous_loss - tol

Question 2: If we set max iter=5000 and tol=1e-3 (the rest as default), does this guarantee that the algorithm will pass over the training data 5000 times? If not, which parameters (and values) should we set to ensure that the algorithm will pass over the training data 5000 times? (2 marks)

When setting the Perceptron to given specifications, the actual iterations performed over the training data is nowhere close to 5000. It is around 20 most of the times. To guarantee that the algorithm performs all 5000 iterations, we must set the tol parameter to None, in which case it will force the algorithm to do "max_iter" number of iterations over the training data.

Question 3: How can we set the weights of the model to a certain value? (2 marks)

When calling the .fit() or .partial_fit() method on a Perceptron instance, we can specify the initial weight used using the sample_weight parameter. Suppose, the weight vector we want to use is W. We make a new tensor containing W repeated for Number_of_samples times, and pass it into the sample_weight parameter.

Question 4: How close is the performance (through confusion matrix) of your NumPy implementation in comparison to the existing modules in the scikit-learn library? (2 marks)

The performance of our NumPy implementation is mostly on-par with the modules of scikit-learn library. Sometimes, our implementation exceeds the scikit-implementation.

Questions for Part 2

Question 1: When we input a singular matrix, the function linalg.inv often returns an error message. In your fit train, y train) implementation, is your input to the function linalg.inv a singular matrix? Explain why. (2 marks)

Although it is unlikely to have an example that causes a singular matrix to formed, this is what is tested in the subtestFn() where each subsequent example is a linear combination of the first example. This causes the determinant of the product of the matrix and its transpose to be zero, which means an inverse does not exist for the matrix.

Question 2: As you are using linalg.inv for matrix inversion, report the output message when running the function subtestFn(). We note that inputting a singular matrix to linalg.inv sometimes does not yield an error due to numerical issue.

The subtestFn() prints "ERROR" to the console when using linalg.inv() because of the presence of the try-except block. In its absence, a **LinAlgError: Singular matrix** is raised.

Question 3: Replace the function linalg.inv with linalg.pinv, you should get the model's weight and the "NO ERROR" message after running the function subtestFn(). Explain the difference between linalg.inv and linalg.pinv, and report the model's weight. (2 marks)

The linalg.inv() function returns the inverse of the input matrix when it exists, but raises an error when a matrix whose determinant is zero i.e. a singular matrix is passed as input. However, the linalg.pinv() returns the original inverse for invertible matrices, whereas, returns the Moore-Penrose pseudo-inverse for singular matrices. The latter is preferred because it handles the case of both invertible and singular matrices.