Part 1:

Q1: Refer to the documentation, what is the functionality of the tol parameter in the Perceptron class? (2 marks)

The tol parameter is the stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss – tol). The default value is 1e-3.

Q2: 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)

It doesn't guarantee that the algorithm would run for 5000 iterations when we set max_iter = 5000 and tol = 1e-3. To ensure that the algorithm runs for 5000 iterations, we need to set max_iter = 5000 and set tol = None. This way, the algorithm will pass over the training data 5000 times.

Q3: How can we set the weights of the model to a certain value? (2 marks)

We set the weights of a model by calculating it using the least square solution. We can also set the weights of the model to a certain value in the fit_perceptron() function

Q4: 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)

We can judge that using the precision scores. The performance of both the methods yields almost the same precision score. And the confusion matrix also yields almost the same results. The difference in its performance is negligible at almost every iteration. But it is worthy to note that a few iterations did yield a noticeable difference in performance with the NumPy implementation giving a better result.

Part 2:

Q1: When we input a singular matrix, the function linalg.inv often returns an error message. In your fit_LinRegr(X train, y train) implementation, is your input to the function linalg.inv a singular matrix? Explain why.

The inputs from subtestFn() is a singular matrix because its determinant is zero. And we cannot compute the inverse of a singular matrix. That is the reason we receive an error

message when we call fit_LinRegr() function. But the inputs from testFn_Part2() computes the inv value, which implies the input matrix is not singular.

Q2. 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.

When we run subtestFn(), the input is a singular matrix which can be verified by computing its determinant. We will receive an ERROR message as the output when we run subtestFn(). In this case we can instead use linalg.pinv to compute its inverse.

Q3. 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.

If the determinant of the matrix is zero it will not have an inverse and your inv function will not work. This happens when the matrix is singular. But the pinv function will work. The pinv function computes the (Moore-Penrose) pseudo-inverse of a matrix. So, when we use the pinv function it returns the inverse of a matrix when it is available and the pseudo inverse when it isn't.

Weights: [-3.33066907e-16 2.00000000e-01 4.00000000e-01]