Part 1b)

1. Refer to the documentation, what is the functionality of the tol parameter in the Perceptron class?
   The tol parameter is the stopping criterion. The iterations will stop when loss > previous_loss - tol, so that the algorithm will terminate when the new loss does not improve or improve very little compared to the previous loss.

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?
   This does not guarantee that the algorithm will pass over the training data 5000 times because the algorithm will terminate if loss < previous_loss - 1e-3 even when it has not iterated 5000 times yet.

   To ensure that the algorithm will pass over the training data 5000 times, we should set the tol parameter to None to get rid of the stopping criterion.

3. How can we set the weights of the model to a certain value?
   We can set the weights of the model by setting the class_weight parameter of the perceptron class, the coef_ attribute of the perceptron class, and the sample_weight parameter of the fit function. coef_ is an ndarray of shape (1, n_features), which is the weights assigned to the features. sample_weight is an ndarray of shape (n_samples,) that serves as the weights applied to individual samples. Sample weights will be multiplied with class_weight (passed through the constructor) if class_weight is specified.

4. How close is the performance (through confusion matrix) of your NumPy implementation in comparison to the existing modules in the scikit-learn library?
   We ran the test function 30 times. For 26 of 30 trials,either our implementation produces the same matrix as the scikit-learn modules or the largest difference between the corresponding elements in the two matrices is less than or equal to two. For the remaining 4 trials, the largest difference between the corresponding elements in the two matrices is greater than two.

Part 2a)

1. 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. (2 marks)
   Since we do not know how data samples are formalized, we cannot guarantee the input matrix of linalg.inv is always invertible. Sometimes the input to the function linalg.inv is singular, sometimes it is not. We cannot make a conclusion because the data are generated randomly.

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.
   Running subtestFn() using linalg.inv resulted in the terminal printing “ERROR”. This always happens because the data matrix provided in subtestFn() is not full column rank. Therefore, it cannot be inverted and linalg.inv always gives error.

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.
   After replacing linalg.inv with linalg.pinv, subtestFn() can be run with no error. The output of subtestFn() is shown below.
   ```
   ------------------subtestFn----------------------
   weights:  [-1.55431223e-14  2.00000000e-01  4.00000000e-01]
   NO ERROR
   ```
   The difference between linalg.inv and linalg.pinv is that the latter produces the pseudo-inverse of the input matrix. It is an “approximation” of the inverse of the input matrix even if it is singular.