1.0 Pocket Algorithm

1.1 What is the functionality of the `tol` parameter in the Perceptron class?

The tolerance to declare convergence. It represents the stopping criterion for the training algorithm. If the change in the mean squared loss between consecutive iterations is less than tol, the algorithm considers that the model has converged and stops training.

1.2 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?

Setting max_iter=5000 in the Perceptron class does ensure that the algorithm will iterate over the training data for a maximum of 5000 passes. However, the tol parameter is related to the stopping criterion for convergence, and it does not directly control the number of iterations.

To ensure that the algorithm goes through the training data exactly 5000 times we can use the early_stopping parameter, if we set it to False then the algorithm will never stop before it reaches 5000 iterations

1.3 How can we set the weights of the model to a certain value?

In scikit-learn, you can set the weights of a trained model in the Perceptron class using the coef_ and intercept_ attributes. These attributes represent the weights and bias term of the linear model, respectively.

1.4 How close is the performance (through confusion matrix) of your NumPy implementation in comparison to the existing modules in the `scikit-learn` library?

Confusion Matrix is from Part 1a is: [[11, 0], [0, 9]]
Confusion Matrix from Part 1b is: [[11 0], [6 3]]

The performance is close but I believe that the algorithm that we implemented is more accurate than the existing module in the `scikit-learn` library.

2.0 Linear Regression

2.1 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.

In our `fit_LinRegr` function when we call `linalg.inv` the input to this function is the dot product of `X_train.T` and `X_Train`. The determinant of a dot product of a matrix and its transpose is always zero. So, our input to the function `linalg.inv` always a singular matrix.

2.2 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 output message when using `linalg.inv` is `ERROR`.

2.3 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.

The `linalg.inv` function returns the inverse of the matrix. The `linalg.pinv` function is useful when your matrix is non-invertible (singular matrix) or determinant of that matrix = 0. If the input is a singular matrix then it `linalg.pinv` will return the pseudo inverse.