ECE421: Lab 1 Q&A

Bryan Yan (yanbryan) and Lyu Wang (wanglyu)

Part 1b

Answer the following question(s), write and save your answer in a separate `PA1_qa.pdf` file. Remember to submit this file together with your code.

1. Refer to the documentation, what is the functionality of the `tol` parameter in the Perceptron class? (2 marks)
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)
3. How can we set the weights of the model to a certain value? (2 marks)
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)

1. Tol is the tolerance parameter, if it is not None, then training will stop once the improvement in error (loss function) is smaller than the tol value. I.e., updates to the weight vector will stop on the iteration where loss > previous_loss - tol
2. It does not guarantee the algorithm will pass over the training data 5000 times. The max_iter makes training stop at or before 5000 epochs, and tol also is a condition to end training earlier if the loss isn't improving by more than tol.
3. To manually set the model's weights, we can simply assign numpy arrays to the model.coef_ and model.intercept_ (the former sets the weights, the latter sets the bias terms). To force the Perceptron to always train for exactly 5000 iterations, we should disable early stopping by setting: Perceptron(max_iter=5000, tol=None)
   Setting tol=None ensures that the model never stops early based on convergence criteria.
   Now, the Perceptron will always iterate through the training data for the full max_iter=5000.
4. It varies, but generally the results from our implementation are fairly close to the scikit-learn implementation

```
(.venv) C:\UofT\ece421\ece421-labs\PA1>python PerceptronImp.py
--------------Test Result-------------------
Confusion Matrix is from Part 1a is:  [[6. 0.]
 [5. 9.]]
Confusion Matrix from Part 1b is: [[ 6  0]
 [ 2 12]]

(.venv) C:\UofT\ece421\ece421-labs\PA1>python PerceptronImp.py
--------------Test Result-------------------
Confusion Matrix is from Part 1a is:  [[12.  0.]
 [ 0.  8.]]
Confusion Matrix from Part 1b is: [[12  0]
 [ 0  8]]

(.venv) C:\UofT\ece421\ece421-labs\PA1>python PerceptronImp.py
--------------Test Result-------------------
Confusion Matrix is from Part 1a is:  [[ 8.  0.]
 [ 1. 11.]]
Confusion Matrix from Part 1b is: [[ 8  0]
 [ 0 12]]
```

Part 2a

Answer the following question(s), write and save your answer in a separate PA1_qa.pdf file. Remember to submit this file together with your code.

- 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)
- 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. (1 marks)
- 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)

1. The input to linalg.inv (which we later changed to linalg.pinv) is X_train.T @ X_train, which may be a singular matrix (specifically it will be singular if the columns of X_train are not linearly independent).
2. I get "ERROR", because linalg.inv can't compute the inverse of a singular matrix, as its inverse just doesn't exist.But sometimes the parameter could be invertible, so linalg.inv does not report an error.
3. I now get "NO ERROR", with the weight shown below. Unlike inv, pinv computes the pseudoinverse, which for nonsingular matrices matches the normal inverse, but for singular matrices, also exists.
   linalg.inv(A): Computes the exact inverse of a matrix only if it is nonsingular (full-rank). If A is singular, an error occurs.
   linalg.pinv(A): Computes the Moore-Penrose pseudo-inverse, which is defined for all matrices, including singular and non-square matrices.
   For nonsingular matrices, pinv gives the same result as inv.
   For singular matrices, pinv provides the closest possible solution, making it useful in least squares regression.

```
(.venv) C:\UofT\ece421\ece421-labs\PA1>python LinearRegressionImp.py
------------------subtestFn---------------------
weights:  [1.04360964e-14 2.00000000e-01 4.00000000e-01]
NO ERROR
------------------testFn_Part2-------------------
Mean squared error from Part 2a is  3046.582974183576
Mean squared error from Part 2b is  3046.5829741835773
```