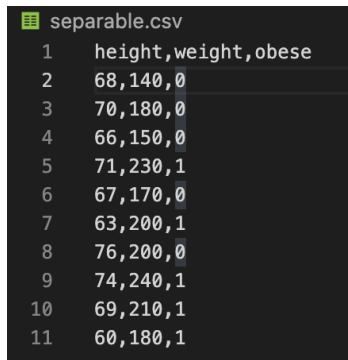


Daniel Reedy Homework 2 Report

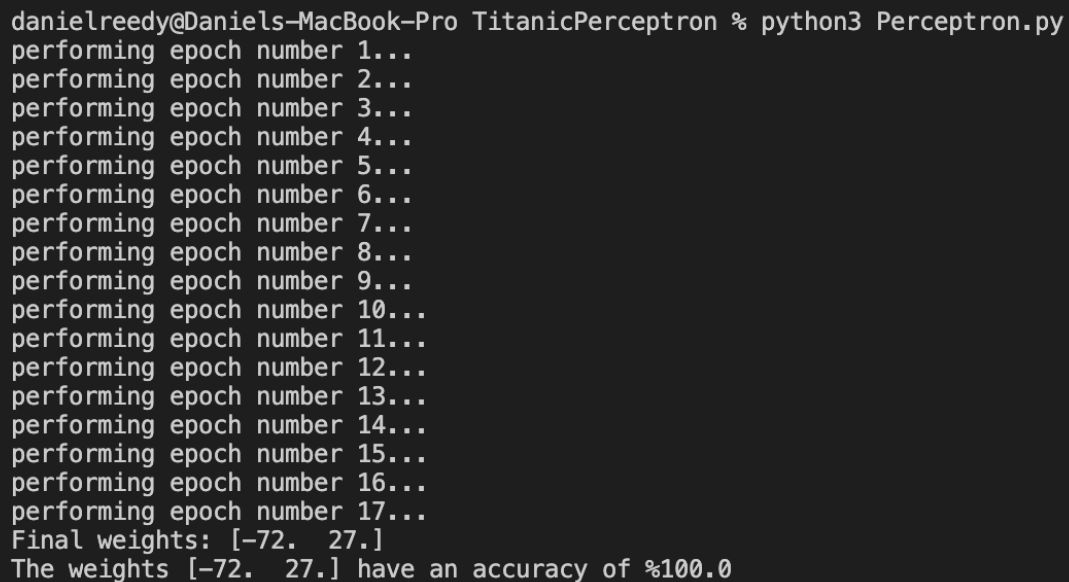
Problem 1

For this problem I created a two-dimensional data set that holds 10 peoples' height and weight, and classifies them as obese or not obese. This data set is clearly linearly separable because the definition of obesity uses a linear relationship between height and weight. The dataset looks like this:



	height,weight,obese
1	68,140,0
2	70,180,0
3	66,150,0
4	71,230,1
5	67,170,0
6	63,200,1
7	76,200,0
8	74,240,1
9	69,210,1
10	60,180,1

When running my Perceptron algorithm, it took 17 epochs to reach convergence. We know that the model reached convergence because it correctly classified every item in the dataset. The results of running my algorithm on this separable data is this:

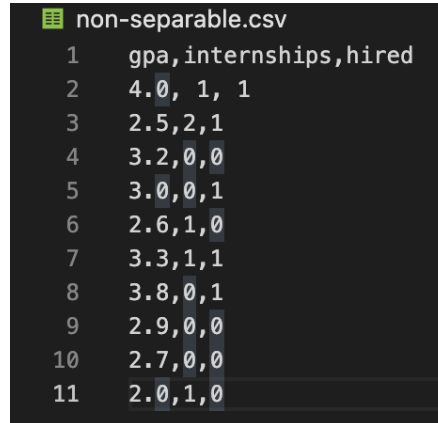


```
danielreedy@Daniels-MacBook-Pro TitanicPerceptron % python3 Perceptron.py
performing epoch number 1...
performing epoch number 2...
performing epoch number 3...
performing epoch number 4...
performing epoch number 5...
performing epoch number 6...
performing epoch number 7...
performing epoch number 8...
performing epoch number 9...
performing epoch number 10...
performing epoch number 11...
performing epoch number 12...
performing epoch number 13...
performing epoch number 14...
performing epoch number 15...
performing epoch number 16...
performing epoch number 17...
Final weights: [-72. 27.]
The weights [-72. 27.] have an accuracy of %100.0
```

This result of 100 percent accuracy is expected when we use a perceptron model on a linearly separable data set.

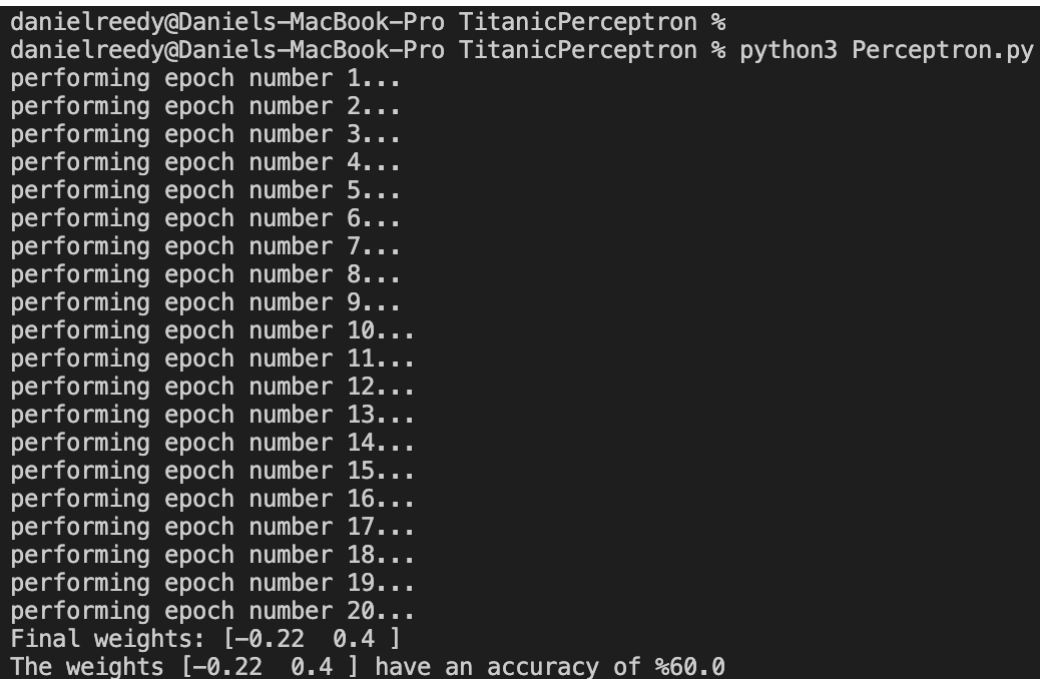
Problem 2

For this problem, I created a two-dimensional data set that holds a student's GPA, number of internships, and whether or not they were hired out of graduation. This data set is not linearly separable, and it looks like this:



	gpa	internships	hired
1	4.0	1	1
2	2.5	2	1
3	3.2	0	0
4	3.0	0	1
5	2.6	1	0
6	3.3	1	1
7	3.8	0	1
8	2.9	0	0
9	2.7	0	0
10	2.0	1	0
11			

When running the Perceptron algorithm, the model never reached convergence. The best prediction accuracy that I was able to achieve on this data was %60. The results of running my algorithm on this data was this:



```
danielreedy@Daniels-MacBook-Pro TitanicPerceptron %  
danielreedy@Daniels-MacBook-Pro TitanicPerceptron % python3 Perceptron.py  
performing epoch number 1...  
performing epoch number 2...  
performing epoch number 3...  
performing epoch number 4...  
performing epoch number 5...  
performing epoch number 6...  
performing epoch number 7...  
performing epoch number 8...  
performing epoch number 9...  
performing epoch number 10...  
performing epoch number 11...  
performing epoch number 12...  
performing epoch number 13...  
performing epoch number 14...  
performing epoch number 15...  
performing epoch number 16...  
performing epoch number 17...  
performing epoch number 18...  
performing epoch number 19...  
performing epoch number 20...  
Final weights: [-0.22  0.4 ]  
The weights [-0.22  0.4 ] have an accuracy of %60.0
```

The model that this algorithm created was not very accurate, but that just shows the limitation of a perceptron model when our data is not close to linearly separable.

Problem 3

For this problem, I used the textbook's implementation of the Adaline gradient descent algorithm. First I transformed the data so that it fit into a numpy array of floats (I also decided to normalize the data between a range of 0 to 1). After doing this, I created a model with the training set using the textbook's Adaline. After lots of tinkering and examining the cost function, I found that the learning step must be less than 0.0003 in order for the algorithm to work properly (otherwise the cost function trended upwards rather than downwards). Eventually, after trying many different numbers of epochs and learning rates, I was able to bring the sum of squared errors down from 171.0 to 63.4 using a learning rate of 0.00028. Unfortunately I was not able to achieve a high accuracy on my data. Here are the results of the algorithm:

```
Correctly classified training samples: 370 out of 891  
The model has an accuracy of 41.53%
```

```
Correctly classified testing samples: 161 out of 418  
The model has an accuracy of 38.52%
```

There is a small decrease in accuracy in the testing data, which is unsurprising because the model was fit specifically around the training data. However, both were so inaccurate that it is difficult to make meaningful comparisons. I tried many things like initializing the weights to random values to avoid local minimums but this did not work for me. An interesting thing that I found while meticulously trying different learning rates is that my model had much better accuracy with the very specific learning rate of 0.0002853. Here are the results with this learning rate:

```
Correctly classified training samples: 616 out of 891  
The model has an accuracy of 69.14%
```

```
Correctly classified testing samples: 306 out of 418  
The model has an accuracy of 73.21%
```

The reason that I did not originally report results using this learning rate is that the cost function does not reach as low values as it did with the previous learning rate (this time it only dropped from 171.0 to 83.0). Therefore, I believe that I should not be using the model with the higher error.

Problem 4

The most predictive feature on my model by far was Sex. I believe this because its weight value has the highest absolute value of all the features (0.508) and the range of possible values for the data was normalized to fit between 0 and 1. Pclass was a close second most predictive feature with a weight absolute value of 0.492. The third most predictive feature was SibSp, with its weight having an absolute value of 0.44.

Problem 5

I created random weights (between -1 and 1) and ran a prediction of the test data using them and got these results:

```
Correctly classified testing samples: 146 out of 418  
The model has an accuracy of 34.93%
```

This random model gave us an accuracy that is worse than the fitted model. This means that the model does bring us a little bit closer to the correct classifications, even if it is not perfect.