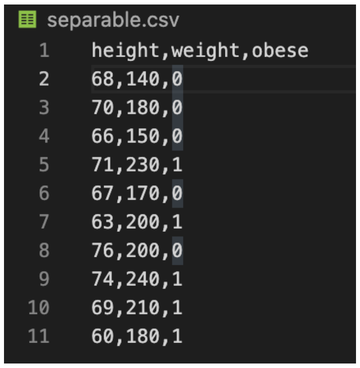
Daniel Reedy
Homework 2 Report

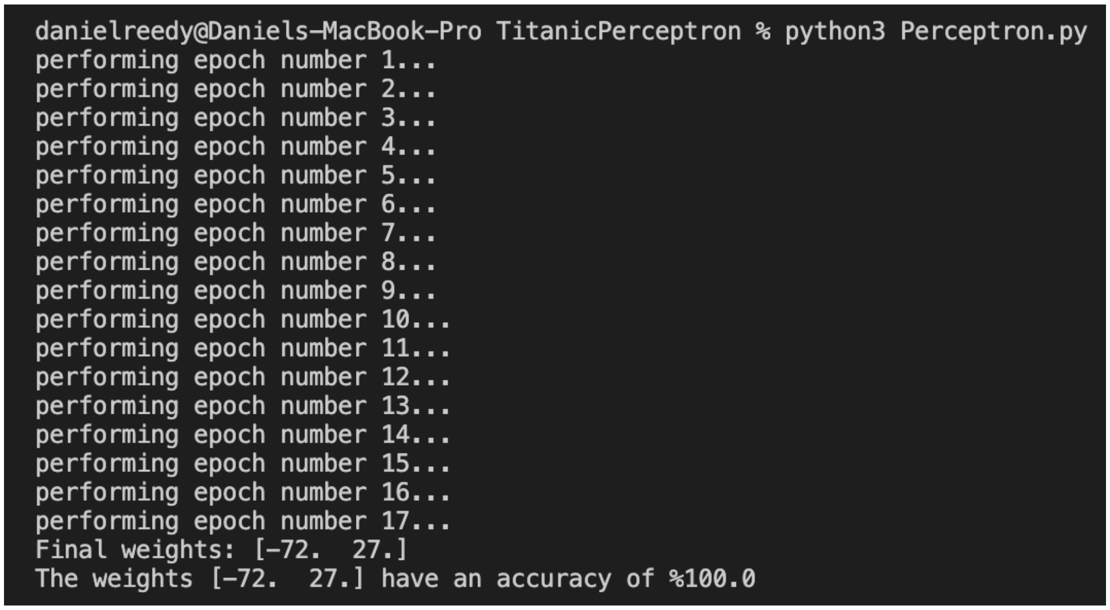
Problem 1

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:



```
separable.csv
height,weight,obese
68,140,0
70,180,0
66,150,0
71,230,1
67,170,0
63,200,1
76,200,0
74,240,1
69,210,1
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:

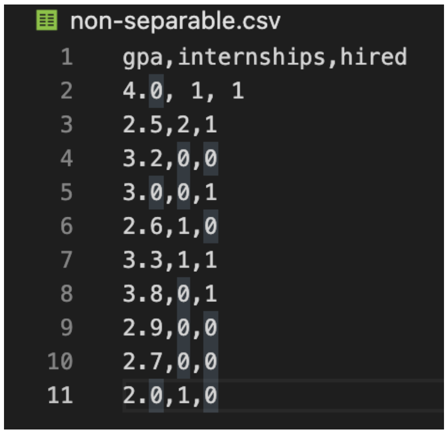


```
danielreedy@Daniels-MacBook-Pro TitanicPerceptron % python3 Perceptron.py
performing epoch number 1...
performing epoch number 2...
performing epoch number 3...
performing epoch number 4...
performing epoch number 5...
performing epoch number 6...
performing epoch number 7...
performing epoch number 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

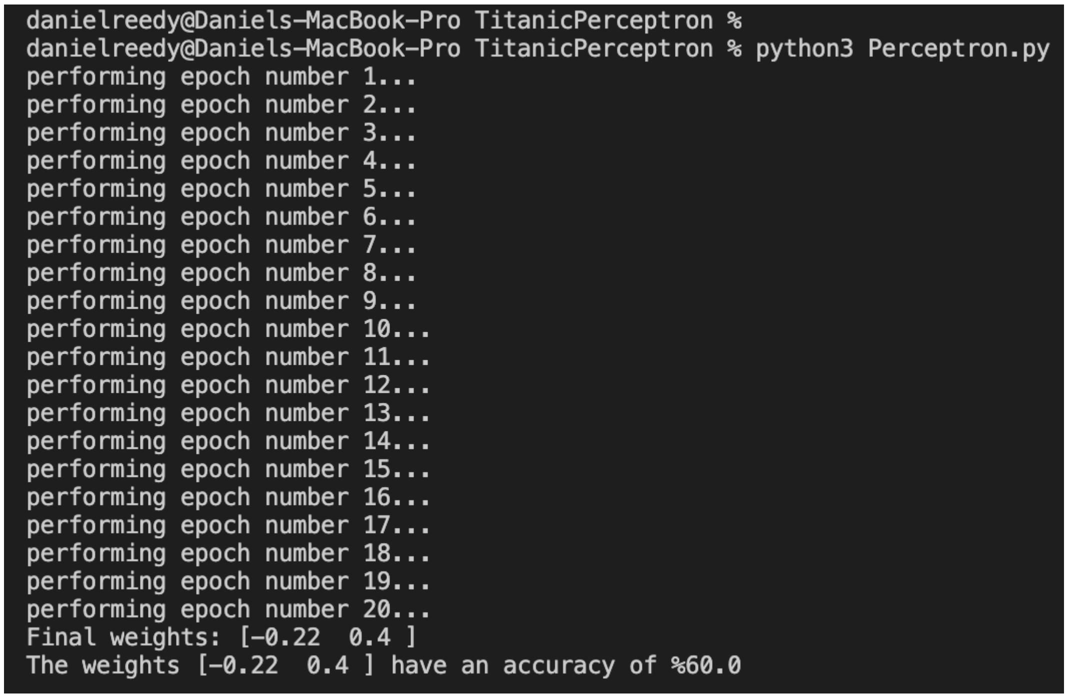
For this problem, I created a two-dimensional data set that holds a student's GPA, number of internships, and whether or not they were hired out of graduation. This data set is not linearly separable, and it looks like this:



non-separable.csv

```
gpa,internships,hired
4.0, 1, 1
2.5,2,1
3.2,0,0
3.0,0,1
2.6,1,0
3.3,1,1
3.8,0,1
2.9,0,0
2.7,0,0
2.0,1,0
```

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:

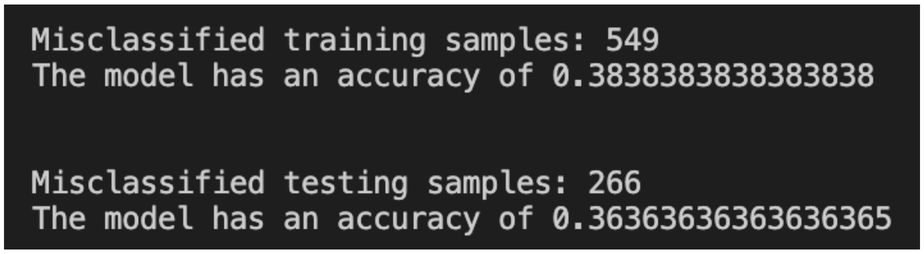


```
danielreedy@Daniels-MacBook-Pro TitanicPerceptron %
danielreedy@Daniels-MacBook-Pro TitanicPerceptron % python3 Perceptron.py
performing epoch number 1...
performing epoch number 2...
performing epoch number 3...
performing epoch number 4...
performing epoch number 5...
performing epoch number 6...
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. 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 0.00001 in order for the algorithm to work properly (otherwise the cost function trended upwards rather than downwards). Eventually, after trying many different numbers of enums and learning rates, I was able to bring the sum of squared errors down from 171.0 to 103.209. Unfortunately I was not able to achieve a high accuracy on my data. Here is the results of the algorithm:



```
Misclassified training samples: 549
The model has an accuracy of 0.3838383838383838


Misclassified testing samples: 266
The model has an accuracy of 0.36363636363636365
```

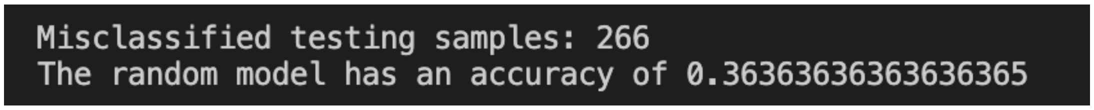
There is a small decrease in accuracy in the testing data, which is unsurprising because the model was fit specifically around the training data. However, both were so inaccurate that it is difficult to make meaningful comparisons. Maybe there was some fault that I made in calculating the accuracy? Or in my data preparation? I am still unsure after lots of digging into the issue, but I hope to find out in time.

Problem 4

The most predictive feature on my model by far was Sex. I believe this because its weight value is the highest of all the features (0.0183) and the range of possible values from this range was tied for the smallest of all of the features (from -1 to 1).

Problem 5

I created random weights and ran a prediction of the test data using them and got these results:



```
Misclassified testing samples: 266
The random model has an accuracy of 0.36363636363636365
```

The fact that this accuracy is the same as the actual model might have something to do with how the predictions are made in this algorithm. The predictions are made uniformly for all of the examples in a data set. So, both my model and the random model predicted all 1's for the test data, leading to the same accuracy on both. I believe I must be using this method of prediction incorrectly because the way that I am using it makes it very difficult to draw meaningful conclusions about the models.