In conclusion I was able to achieve about 95% accuracy. This led to about 98.5% accuracy in the full data set. I think this is a high probably off mission success. They could definitely plot a course that makes them approach the least amount of objects and then their chance of success would be higher. The chances aren't perfect but they are looking good for success. The confusion matrix shows that we misclassified 2 out of 27 rocks as mines, and it only misclassified 1 out 33 mines as a rock, , which is the preferred miss. If we look at it from another perspective, if we avoided all the predicted mines, there would be really a 98.3% percent chance on the test set that we would survive (up from 95%). When doing this training I used both test set metrics and full data set metrics but focused more on the full data set results. Knowing if the full data set test is good accuracy, and the test set accuracy is good we are not over fitting. When observing the plot one might notice that right off the bat, 1 component has pretty good results about 89%. This is because I realized that the model does not completely start over every time you call it but only when you rerun your script. So, it already had some knowledge when going to fit because I looped through twice. This improved my overall results from about 94% to about 95% accuracy. The number of components that achieved the highest accuracy was 19. This is an interesting result because the graph trends somewhat upward until about 19. Meaning the accuracy was getting better and starts to actually trend downward after. This would point at the first 19 components being useful in predicting and the rest just causing noise in the prediction. Finally, regarding the parameters chosen for the classifier, some of these were chosen by trial and error but some had some reasoning behind them. For the hidden layers, I did some research on how to pick those. 66-90% of the number of input features should be sufficient for a lot of neural networks but after some trial and error, I chose 60 nodes for the first hidden layer and then did 30 and 20 and so on. I wouldn't say that I got a huge gain in accuracy from using more inputs, but I did get some gain by having more nodes and more hidden layers. The activation function I chose was 'relu' and that just had the best performance. The solver I chose was the lbfgs, and not only did this perform better but it converges much faster too. This was stated in the documentation for smaller data sets and was observably true. The other parameter that is worth noting is the random state. I tried random states between 0 and 10 but when I set it to 2 and it just performs better. If you don't define the random state it will change on each call of the classifier and I don't think its as reliable because the reason 2 works is because you are setting the internal parameters to a place where they find their minimums in the best manner. Tolerance and alpha were chosen initially based on the lecture but didn't find any improvements by adjusting them. Max iteration was chosen initially based on the example too but when I increased because of warnings I received that it wasn't converging, I didn't see any change in the outcome.