1.Accuracy = [0.99]
Final Weights = [[-3.80657242] [-2.840821] [-3.0719334] [-1.4003906] [4.9]]

A perceptron is basically a function. It takes in inputs and spits out outputs. It attaches weights to each input and uses them in the calculation to get an output. If the output is incorrect it changes the weights until the output is correct. The weights you see above is the attached weights to each input and the bias that manages to create an accuracy of 99%. My code update weights after each training instance not the batch updates. The Predict function uses the current weights to predict the calculate the output. The Fit function run through the data continuously, changing the weights until the model's accuracy has reach 100% or has not improved enough. The score function get the predicted output and compares it against the real output, it keeps tracks of how many the model got wrong. Get_weights returns the weights of each input at the time of it being called.

Data Set 1

X	Y	Label
0.4	-0.3	1
0.5	-0.5	1
0.9	-1.6	1
1.2	-2.3	1
-0.3	0.4	0
-0.7	0.9	0
-1.2	1.4	0
-1.5	2.3	0

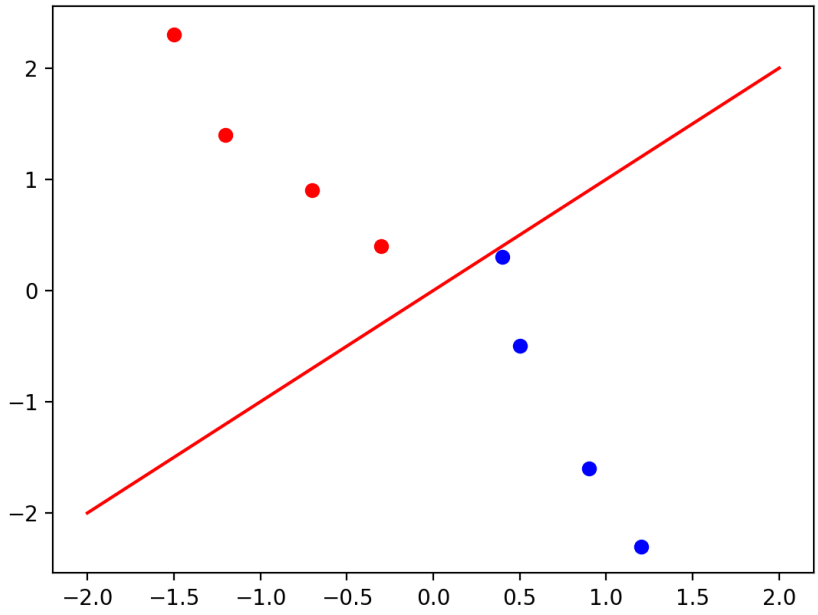


Figure 1: Data Set 1 Instance and Decision Line

Data Set 2.

X	Y	Label
0.4	0.3	1
0.5	-0.5	1
-0.9	-1.6	1
1.2	-2.3	1
-0.3	0.4	0
0.7	-0.9	0
-1.2	-1.4	0
-1.5	2.3	0

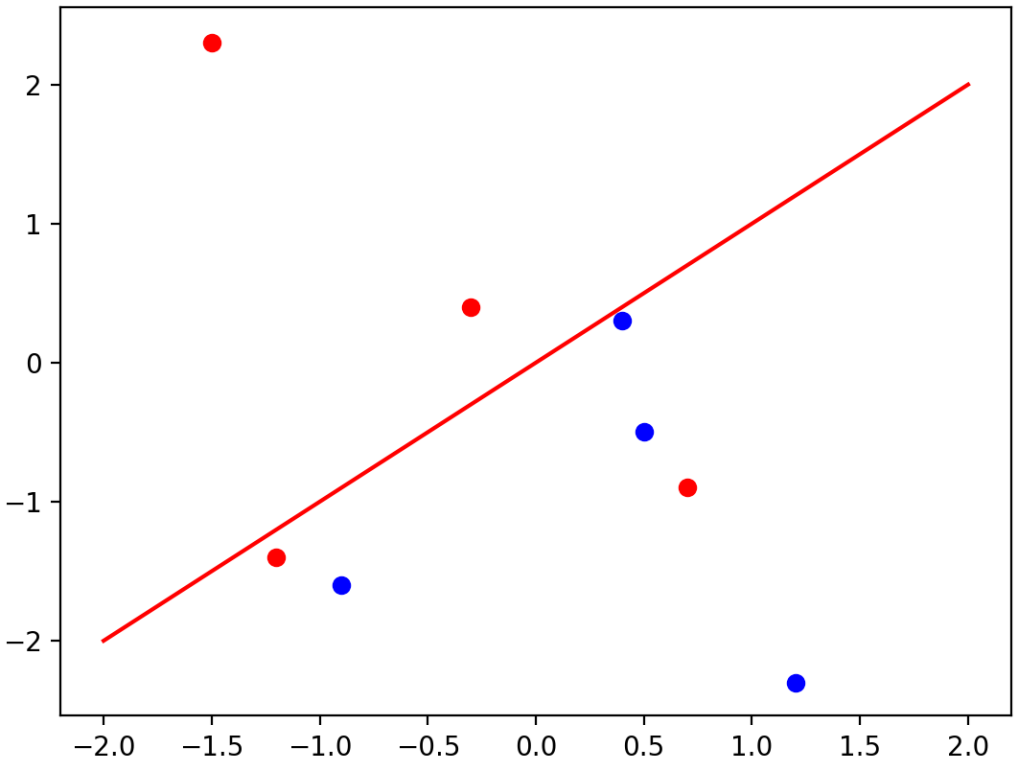


Figure 2: Data Set 2: Instance and Decision Line

2. Data set 1 is linearly separable, with 4 instances of Red and 4 instances of Blue Data set 2 is not linearly separable, with 4 instances of Red and 4 instances of Blue.

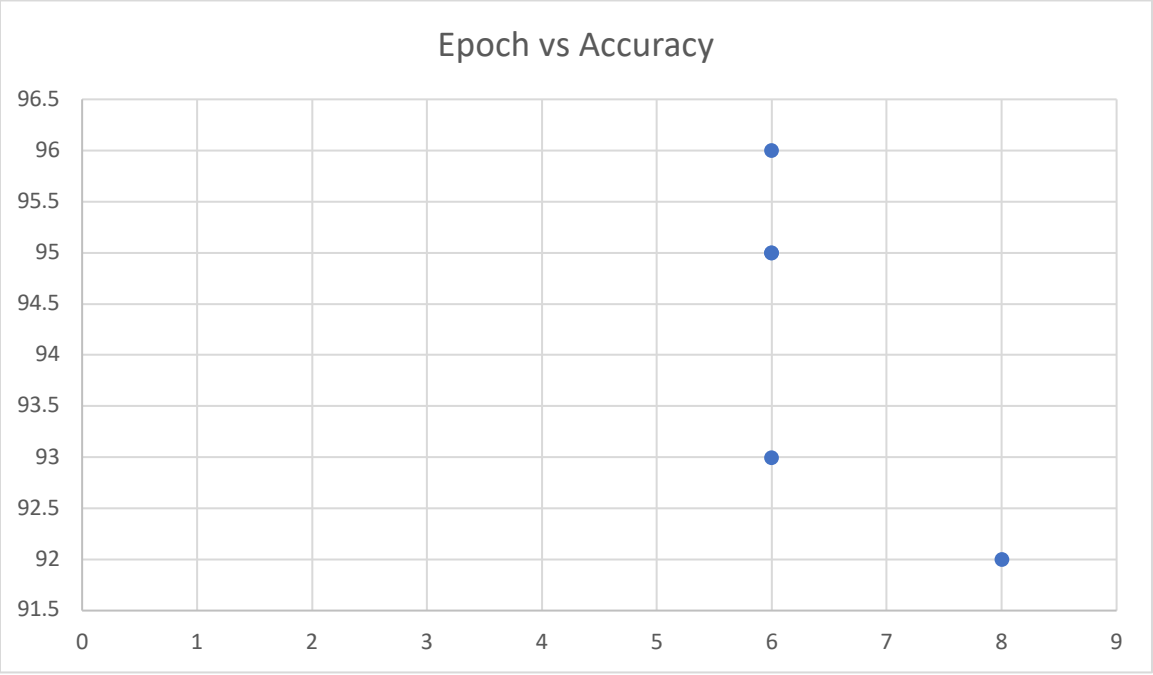
3. The learning rate determine how much the weight changes by. So if my learning rate is very small, it will take more epochs to change the weight a sufficient amount in order to change the output. In my code, learning rate isn't that important, it has minimal effect. Despite the learning rate it still takes 12 epochs to fully train on the linsep2nonorigin.

In the first data set, you can see the line split the two classes. That because the data is linearly separable. One line can divide both classes into different sides of the line. However in the second data set, the line is unable to properly divide each class. There are Red dots in the blue side.

4.

Epochs	Train Accuracy	Test Accuracy
6	0.94	0.93
6	0.96	0.96
6	0.95	0.95
8	0.96	0.92
6	0.98	0.95
Avg		
6.4	0.958	0.942

By analyzing the weights of each input, the best indicators of voting democrat is voting yes on adoption of the budget resolution. And the best indicator of voting republican is voting yes on the water project cost sharing bill. The least critical indicators are voting yes on the el Salvador aid bill. It has a weight of 0 so it does not swing either way.



As you can see with this graph, it looks like there is no correlation between epochs and misclassification. It seems that all of my model seems to stop running around 6 epochs, and the one that ran for 8 epochs did not have an improvement in accuracy. I think this mostly depends on the luck of the draw, because the data is randomly split and shuffled after every epoch. So it depends greatly on how the data is divided.

5. Comparing my results to the sklearn perceptron, my perceptron is worse. The sklearn perceptron is on average 2% more accurate on the testing data than mine. I think the sklearn perceptron works exactly like mine except it probably has a better stopping criteria allowing it to get as close as possible to the theoretically limit accuracy for the data set. I think my stopping criteria is not as precise as theirs and my perceptron stops too early. .