CSE353 Assignment3 Perceptron Report

Chengzhi Dong
ID 112890166

Due October 28 2021

1 Introduction

For assignment 3, I need to implement the Perceptron Learning Algorithm (PLA) to estimate a decision boundary for classifying linear separable data. I need to implement PLA's variant, the Pocket Algorithm, to estimate a decision boundary with least number of mistakes for classifying non-linear separable data. I implement the algorithm on different size of data. Also, I need to estimate a decision boundary with a set of training data, then apply the decision boundary I found on a set of testing data and check its error rate.

2 Method

My algorithm mainly implemented the the Perceptron Learning Algorithm (PLA) to estimate a decision boundary for classifying linear separable data and the Pocket Algorithm to estimate a decision boundary with least number of mistakes for classifying non-linear separable data.
First, I imported the numpy, matplotlib.pyplot, and random libraries.
Then, I defined two functions. The sign function takes two parameters: w (a decision boundary) and x (a data point array). The sign function will return a sign, positive or negative, of the data point relative to the decision boundary. The sign function will calculate the result of transpose of w times x. If the result is negative, it returns a negative 1; else if the result is positive, it returns a positive 1; else it returns zero.
The count_error takes takes three parameters: w (a decision boundary), x_data (x data matrix), and y_data (y data array). The count_error counts the mistakes that the decision boundary has over the x and y data. It loops over the x data matrix and y data array. For each point of x data, it calculate the sign of the point and the w, and compare the sign with its corresponding y data. It counts one mistake if the sign and corresponding y data are unequal. After the loop, the functions the total of mistakes it found.

For Part 1, the algorithm first load the data from the two text files, "X_Linear-Separable.txt" and "Y_LinearSeparable.txt". Then, it found the size of the samples and each x point. The algorithm loops over the x, and store every x point into a 2-d array with each inner array being a x point. Also, it store positive and negative x1 and x2 value into separate array for plotting each points. It found the min and max of x1 and x2 values for plot the line of decision boundary. For the part of PLA, the algorithm initialize an array of w_{ls} (decision boundary) to zeros, and an order for iterate through the data. I used a while loop to keep the iteration going. For every new iteration, the algorithm shuffled the order. The algorithm check each point with the current decision boundary w_{ls} to see if it has any mistake. When encounter a mistake, the algorithm correct it by adding $y[i] \times x[i]$ to w_{ls}. The algorithm iterate through the data again and again, until it found no mistake after a full cycle of each data. Then the algorithm found the optimal decision boundary for this set of data. For visualization, using the positive and negative arrays found earlier, the algorithm plot each data point with positive being blue and negative being red. Using the decision boundary $w_{ls}[2]$ is not zero, the algorithm uses the min and max of x1 values found earlier and plug into w_{ls} to found the x2 values of the two end points and draw the line of decision boundary. Else if $w_{ls}[2]$ is zero, it means the line of decision boundary is a vertical line and the algorithm uses the min and max of x2 values found earlier and w_{ls} to draw the vertical decision boundary line. The algorithm also calculated and print the error rate of the decision boundary.

For part 2, the data in non linear separable, so instead of using PLA, I implements its variant, the Pocket Algorithm. Same as part 1, I load the data and store them into arrays. For the Pocket Algorithm, the algorithm initial two decision boundaries, one is a temporary w called w_{temp} and another is the pocket w called w_{nd}. Each w has its corresponding number of error called min_error and current_error. Instead using a while loop, this time I used a for loop with 1000 iterations max. The order is also randomized for each new iteration. The algorithm loops over the data, when it encounter a mistake with w_{temp}, it corrects the w_{temp}. Then it compare the number of error of w_{temp} to the number of error of w_{nd}. If w_{temp} is better, it updated w_{nd} to w_{temp} and the min_error. After the 1000 iterations, the algorithm found an optimal decision boundary, w_{nd} with least number of errors. Then, same as part 1, the algorithm plot the points and the line of decision boundary for visualization. Also, the algorithm calculated and print the error rate of the decision boundary.

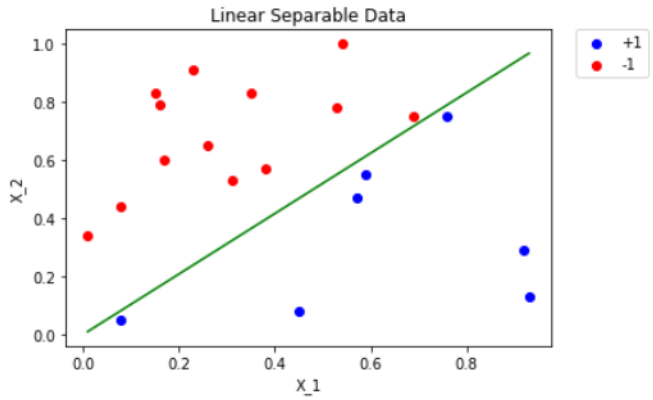
For part 3, there are more sample data and the positive represent '1' and the negative represent '5'. But it is basically the same as part 2. For the training part, the algorithm load the training data, find its sample size, store data into different arrays. Then the algorithm implements the Pocket Algorithm to find the optimal decision with least number of error using the training data. The algorithm plot the points of the training data and the line of decision boundary. It calculated and print the error rate of the decision boundary on the train data. For the testing part, the algorithm plot the test data and the same decision

boundary line it estimated from training phrase. Then it calculated the error rate of the decision boundary on the test data.

For part 4, there are more sample data and each sample has 257 parameters instead of 3. But it has no large difference from part 3. For the training part, the the algorithm load the training data, find the size of samples and each x point, store data into different arrays. The initial temporary and pocket decision bound are arrays of zeros of size of 257. Then the algorithm implements the Pocket Algorithm to find the optimal decision with least number of error using the training data. Since I cannot plot a 257 dimension graph, the algorithm only calculated and print the error rate of the decision boundary on the train data. For the testing part, the algorithm calculated the error rate of the decision boundary it estimated from training phrase on the test data.

3 Experiment

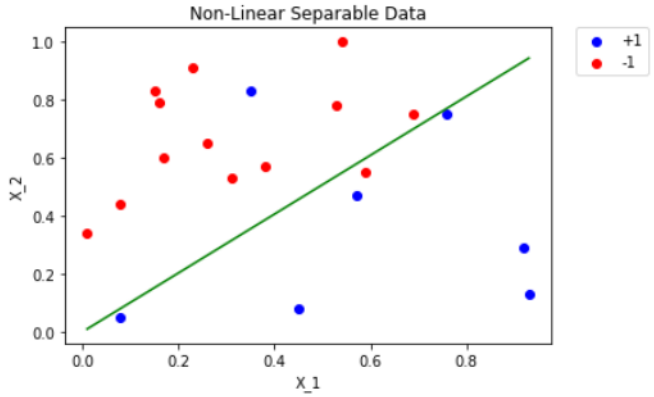
Part 1:
Below are my results for part 1.




```
Decision Boundary for Linear Separable Data = [ 0.    1.53 -1.47]
Number of Error for Linear Separable Data: 0
Error Rate for Linear Separable Data: 0.0
```

The algorithm successfully estimated the optimal decision boundary with error rate of 0 on this set of linear separable data. The result is exactly what I expected.

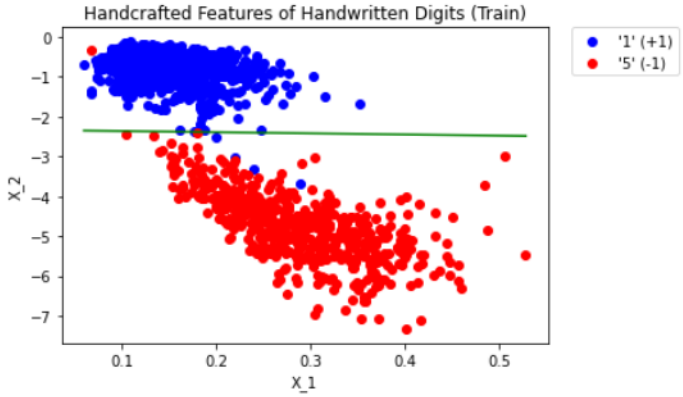
Part 2:
Below are my results for part 2.



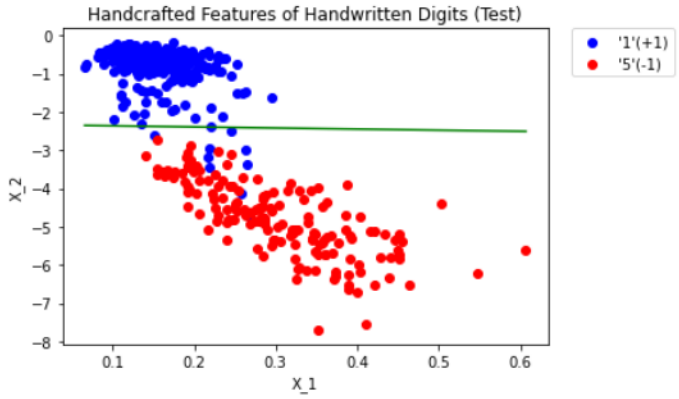

```
Decision Boundary for Non-Linear Separable Data = [ 0.    1.33 -1.31]
Number of Error for Non-Linear Separable Data: 2
Error Rate for Non-Linear Separable Data: 0.1
```

The algorithm successfully estimated the optimal decision boundary with error rate of 0.1 on this set of non-linear separable data. After I examined the data, I think 2 is the least number of error I can get with a linear decision boundary, so the algorithm did a well job.

Part 3:
Below are my results for part 3.




```
Decision Boundary from Training Data of Digit Handcrafted Features = [9.     1.1094 3.8603]
Number of Error for Digit Handcrafted Features Data (Train): 5
Error Rate for Digit Handcrafted Features Data (Train): 0.0032030749519538757
```




```
Decision Boundary from Training Data of Digit Handcrafted Features = [9.     1.1094 3.8603]
Number of Error for Digit Handcrafted Features Data (Test): 9
Error Rate for Digit Handcrafted Features Data (Test): 0.02122641509433962
```

From the training phrase, the algorithm generated an optimal decision boundary which has an error rate of 0.0032 on the training data set. The same decision boundary has an error rate of 0.0212 on the testing data set. Consider part 3 has very large sample size, the decision boundary having only 5 errors on the training data set and 9 errors on the testing data set is a decent result.

Part 4:
Below are my results for part 4.

```
Number of Error for Raw Data (Train): 0
Error Rate for Raw Data (Train): 0.0

Number of Error for Raw Data (Test): 4
Error Rate for Raw Data (Test): 0.009433962264150943
```

Based on my results, I am surprised that the train data set is linear separable. The optimal decision boundary that the algorithm estimated using the training data set has an error rate of 0 on the training data set. For the test phrase, the same decision boundary has an error rate of 0.0094 on the testing data set. It is interesting to see that the algorithm can estimate a decision boundary with only 4 mistakes on this large size of samples.

4 Discussion

Originally, I set a large number of iteration for the Pocket Algorithm, approximately 100000. It takes a very long time to run. Soon, I realize the pocket w only updated for the first maybe 200 iterations. So, I decrease the number of iteration to 1000. It works perfectly fine and reduce the running time.

For the order of iteration, I have tried a naive way, which iterate the data

every time in the same original order, and the random order, which I shuffled the order every time for each iteration. Both order of iteration have similar results, same number of errors but different w. However, the result is always the same for the naive way. Therefore, I prefer the random way which results a slightly different w every time I run the program.

To have better results, I could try to estimate the decision boundary with other variants of PLA, such as Adaptive Linear Neuron algorithm, its batch version, and variants of the cyclic strategies. Also, using linear regression to initialize the w_0 would be a better start than initialize it with zeros.