# Linear Classification (PLA) and Regression

## Implementation report for PLA and Pseudoinverse Algorithm

Perceptron Leaning Algorithm is implemented on a data generated programmatically. As the generated data is linearly separable, algorithm as a guaranteed convergence. The performance of algorithm is compared by giving 0 initial weights against the weights calculated by linear regression.

# Linear Classification (PLA) and Regression

Implementation report for PLA and Pseudoinverse Algorithm

## Data Formatting

Data is generated by an internal function in the program. Data is 2 dimensional and within the interval of -1 to +1. This can be imagined as x-coordinates or y-coordinates. We pick random 2 points and generate a line (target function). This line classifies all the data points either as +1 or -1. Labels are stored in Y and data points are stored in X. Dimensions of X are [N x 2] and that of Y are [N x 1] where N is the number of data points. The function implemented for this is

[X,Y] = generateData(N)

## Algorithm

### PLA

Perceptron learning algorithm has given a set of input data and corresponding labels. Initially, weights are zeros i.e. W = [0 0 0].

**Note: Weights W = [w0 w1 w2] has additional parameter w0. For this, I have adjusted the X matrix as [N x 3] where 1st column is all 1s.**

Below is the algorithm:

1. Make an adjustment for w0.

2. Calculate the $W^TX$ and store the result.

3. Compare the result with Y by multiplying result and Y.

4. If multiplication is negative, the point is misclassified.

## Perceptron Learning

Perceptron Leaning Algorithm is implemented on a data generated programmatically. As the generated data is linearly separable, algorithm as a guaranteed convergence. The performance of algorithm is compared by giving 0 initial weights against the weights calculated by linear regression.

5. Make a Boolean array where Misclassified point index has a value= True.

7. Choose a random misclassified point from that array.

8. Update the weight W0 = W0 +$y_n x_n$.

9. Increment the iteration count.

10. Go to step 2 till Boolean array has all False element i.e. No- Misclassified point

11. Return the final weights calculated.

Please note that I have used an infinite loop condition for PLA. In case, if data does not converge the program will keep on executing.

### *Pseudoinverse*

In this algorithm I have used a function w = np.linalg.pinv(X_new). This function takes a pseudoinverse and calculates the new weights. We use these weights as initial weights for our perceptron algorithm and we start from step 2.

## *Results of PLA:*

**Note: The results are highly fluctuating and may not generate the same numbers again. As the experience are conducted on a random numbers. For every experiment I have generated random samples and random target function and 100 such experiments are carried out. The average number of iteration may vary significantly.**

Below tables shows number of iteration taken for convergence. The iteration number is rounded up to nearest integer.

| Without pseudoinverse | | | | | | |
|---|---|---|---|---|---|---|
| #experiments/#data points | 10 | 50 | 100 | 200 | 500 | 1000 |
| 100 | 13 | 53 | 125 | 180 | 546 | 863 |

| With Pseudoinverse | | | | | | |
|---|---|---|---|---|---|---|
| #experiments/#data points | 10 | 50 | 100 | 200 | 500 | 1000 |
| 100 | 6 | 38 | 74 | 169 | 458 | 891 |

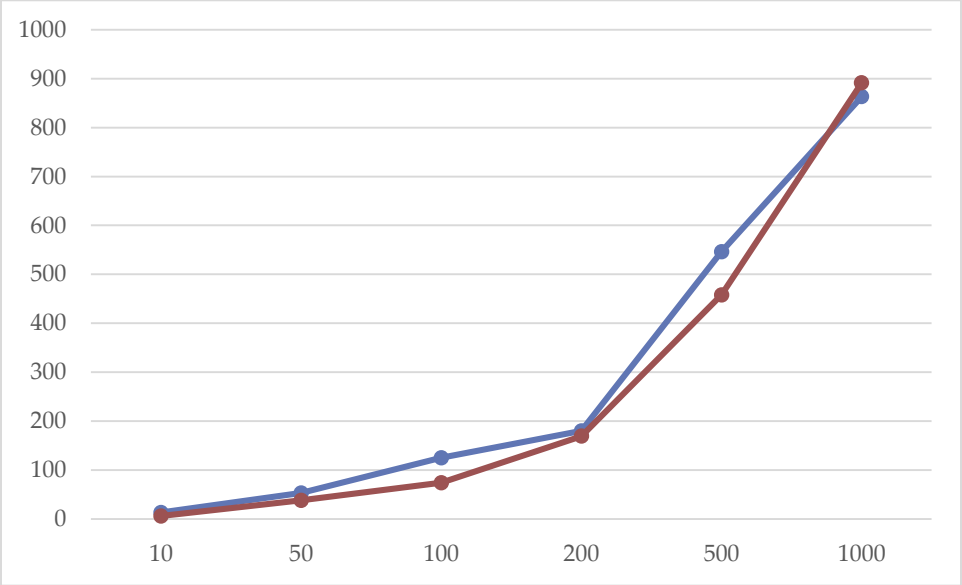


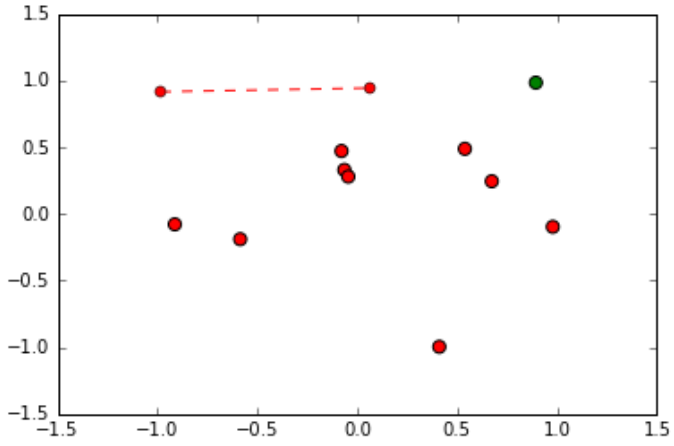

Fig 1: Graphical representation of iteration comparison. Blue line – Iteration taken by PLA, Brown line – Iteration taken by PLA with pseudoinverse.

A simple demonstration of the program is shown below for 10 data points. The plotting functions used are not part of actual program. Plotting functions are used only for this demo.

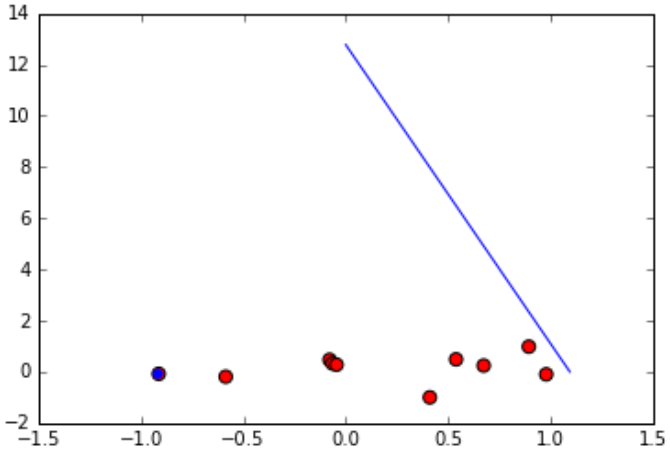
PLA converged in 6 iterations. Please find the below graph for each iteration. The blue point in the graph is the misclassified point chosen to calculate the new weights.

**Note: The scale of graph is changing in every iteration but the data points are same.**

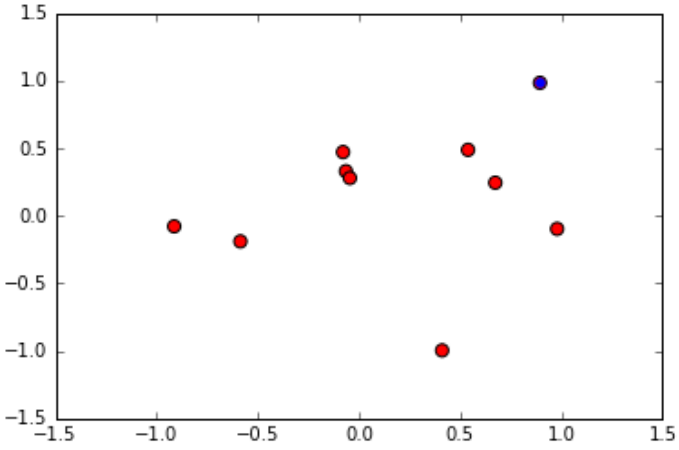
Below is the random line for data generation (Target Function) Blue point is misclassified point that is picked to update the weights.



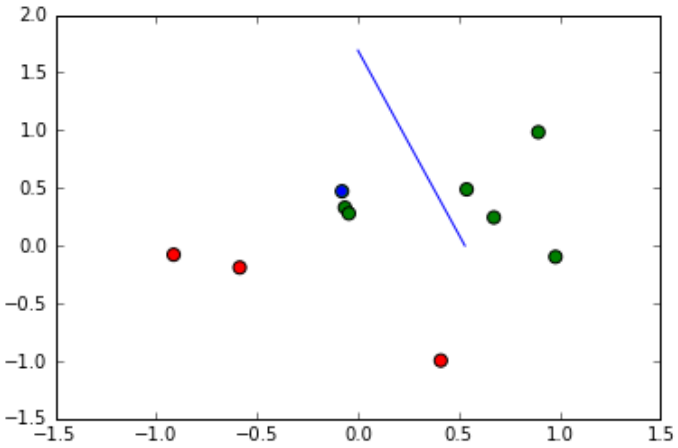
Iteration 1:



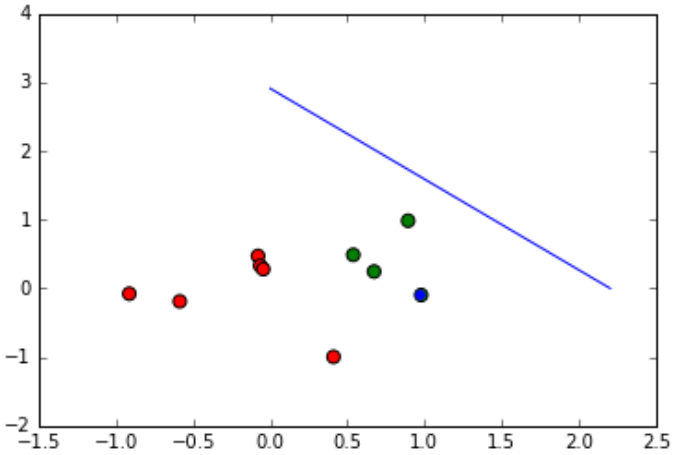

Iteration 2:



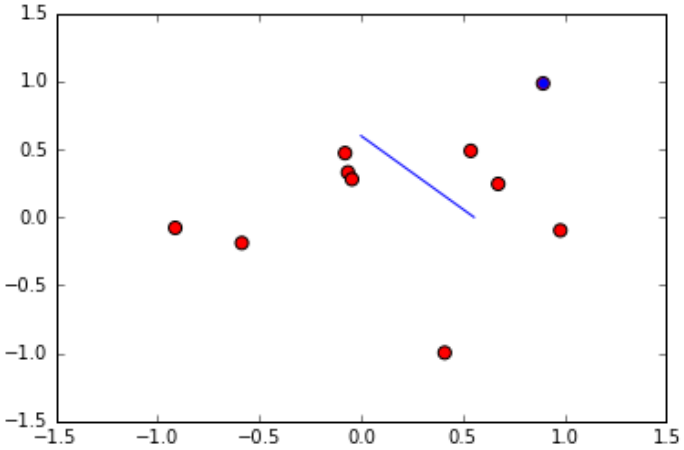
Iteration 3:



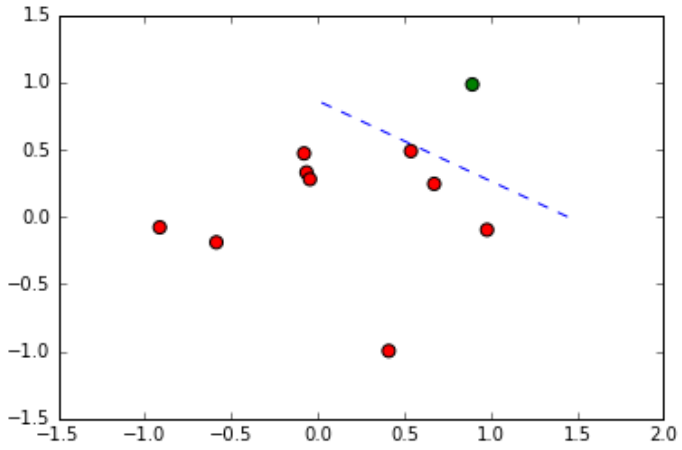

Iteration 4:




Iteration 5:



Iteration 6:

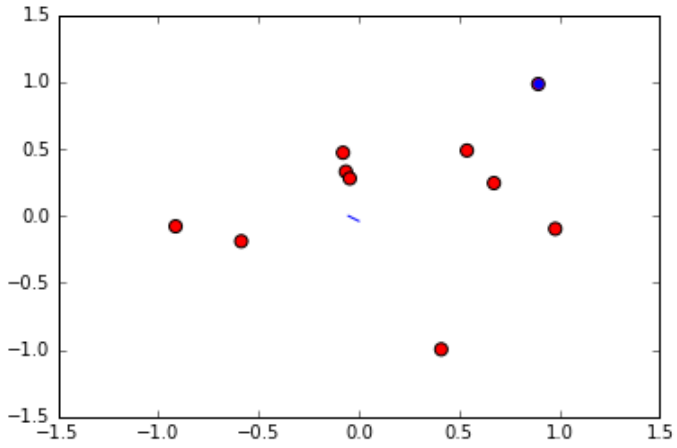



Here in iteration 6 – Data is classified correctly. Weights calculated are W = [-2.000 8.223 1.7681]

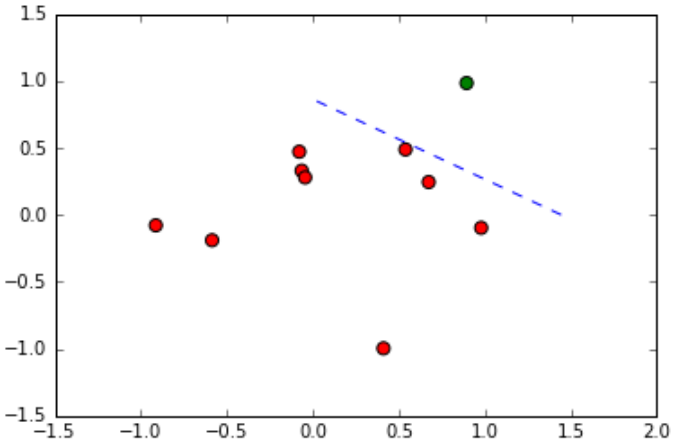
## *After Regression:*

This data is then given to pseudoinverse algorithm to calculate the weights. The weights given by regression are W1 = [-0.938 0.294 0.592]. Now, we give these weights as initial weights to PLA. Here, converges in 2 iteration as below.

Iteration 1:




Iteration 2:




All the points in this iteration are classified correctly.

## *Miscellaneous:*

Code is using some built in functions for task like taking average, calculating pseudoinverse and plotting graph. Reference for these functions is taken from python official web-sites. These are only supportive functions and hence used in implementation.

Another aspect: When perceptron updates the weights W= [w0 w1 w2]. Here, w0 will always be an **integer** and has no decimal precision. However, weights from linear regression has a decimal precision for every element in a vector. This can be one of the reason that number of iteration in pseudoinverse PLA sometimes outnumber general case of PLA. There may be slight change in notations in the program and that is just for sake of simplicity.