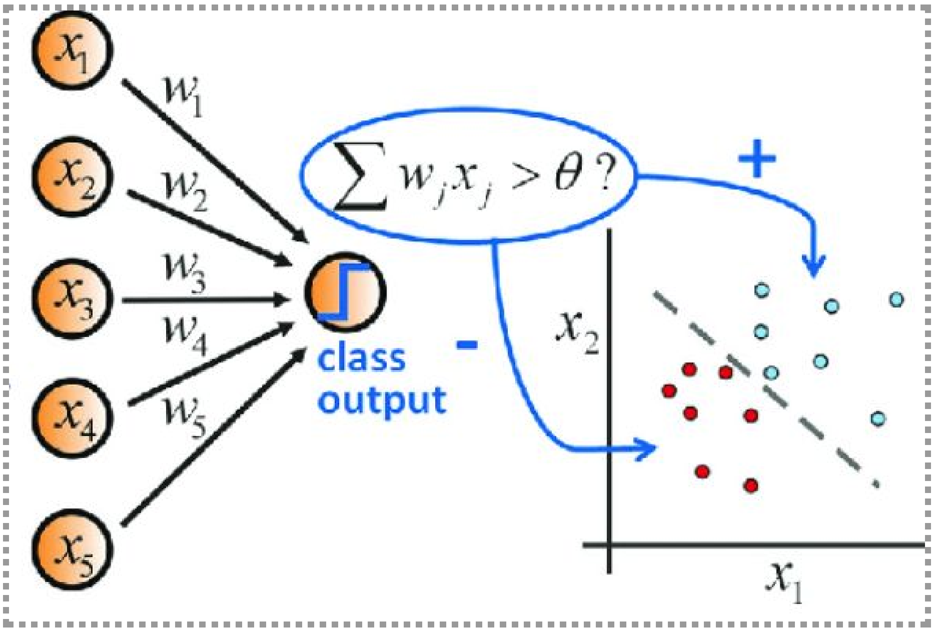
Project 2- EEVAL: BINARY CLASSIFIER



STUDENT: Xián García Nogueira 35578885R

LECTURER: Bogdan Grzegorz

14/01/2021

INDEX

1. Objective and implementation general steps

In this laboratory we propose the implementation of a **binary classifier** in R2 (R2 = R × R = {(x, y): x where y are real numbers) with learning algorithm.

- Language of implementation: MATLAB.
- Project structure: 2 scripts with .m extension
  - training.m
  - pro2.m

➢ STEP 1

The dataset for training is generated from the provided script called training.m and by executing this, the dataset is saved in data.mat.

```
rng(6077);
N=20;
A=[randn(N/2,1) rand(N/2,1)+0.5;randn(N/2,1) -rand(N/2,1)-0.5]
angle=randn(1);
x(:,1)=A(:,1)*cos(angle)-A(:,2)*sin(angle);
x(:,2)=A(:,1)*sin(angle)+A(:,2)*cos(angle);
x=x+randn(1,2);
y=sign(A(:,2));

save data.mat;
```

training.m

➢ STEP 2

To implement the training algorithm, the related pseudocode present in chapter 5.1 of the book “**Knowledge discovery with support vector machines**” has been followed.

let $D = \{(\overline{x}_1, y_1), (\overline{x}_2, y_2), \dots, (\overline{x}_l, y_l)\} \subset \mathbb{R}^n \times \{+1, -1\}$
let $0 < \eta < 1$
$\overline{w} \leftarrow \overline{0}$
$b \leftarrow 0$
$r \leftarrow \max\{|\overline{x}| \mid (\overline{x}, y) \in D\}$
repeat
  for $i = 1$ **to** l
    if $\operatorname{sgn}(\overline{w} \bullet \overline{x}_i - b) \neq y_i$ **then**
      $\overline{w} \leftarrow \overline{w} + \eta y_i \overline{x}_i$
      $b \leftarrow b - \eta y_i r^2$
    end if
  end for
until $\operatorname{sgn}(\overline{w} \bullet \overline{x}_j - b) = y_j$ with $j = 1, \dots, l$
return $(\overline{w}, b)$

TRAINING PSEUDOCODE

```
[l,p]=size(x);
w=[max(max(x));0]; %I start with the max value from w equeal to the max x in the data given.
b=0;
n=0.1;          %LEARNING RATE
r=max(sqrt(sum(x)));
iteration=1;
TotalError=1;

    %TRAIN CODE
while TotalError~=0 %Repeat while the algorith is not correctly trained.

    for i=1:l  %repeat for each element(20)


       if sign(x(i,:)*w - b)~= y(i) %if the perceptron is wrong, then change value from w and b
           w=w'+n*y(i)*(x(i,:));
           b=b-n*y(i)*(r^2);
           w=w';
       end
    end
```

TRAINING MATLAB CODE

- ## STEP 3

In order to test all that is implemented, the following aspects are calculated:

- **FP *-False Positives-***
- **FN *-False Negatives-***
- **TP *-True Positives-***
- **TN *-True Negatives-***
- **Total of errors by cycle**

- ## STEP 4

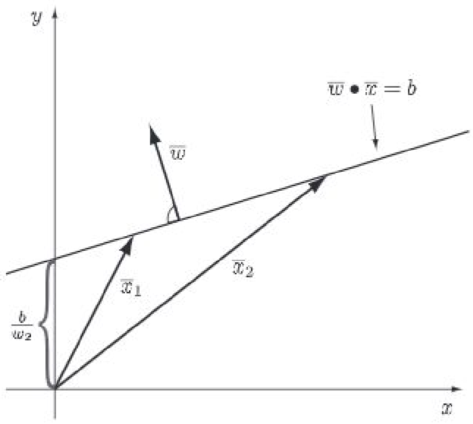
To visualize how the algorithm works it is plotted using subplots the following values for each cycle:

- **Weight(x)**
- **Weight(y)**
- **Bias**

Besides, a figure is generated for each cycle showing the binary classification where we can appreciate changes in the false positives, false negatives, true positives and true negatives represented as:

To calculate and represent decision boundary is necessary to use **y = -mx +b** where:

- m = -(b / w2) / (b / w1)
  - knowing the x and y intercepts (0,-b/w2) and (-b/w1,0) respectively.
  - we can use m = y2-y1/x2-x1 so m = -(b / w2) / (b / w1)
- **y = (-(b / w2) / (b / w1))x + (-b / w2)**
  - following the structure of y = -mx +b
  - knowing x is initialized to -15:15

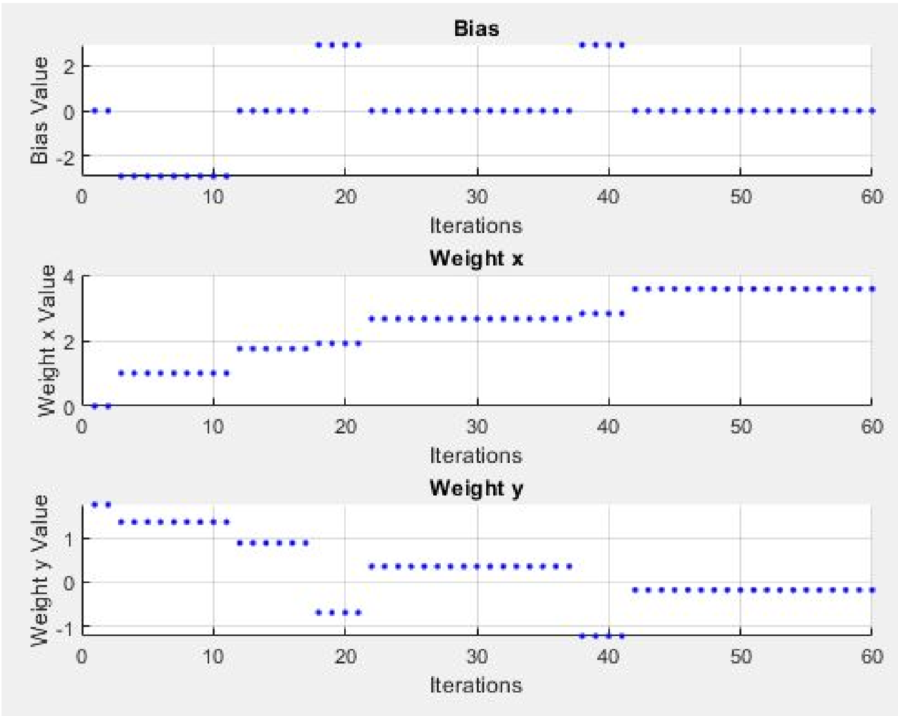


2. Learning Rate Variation

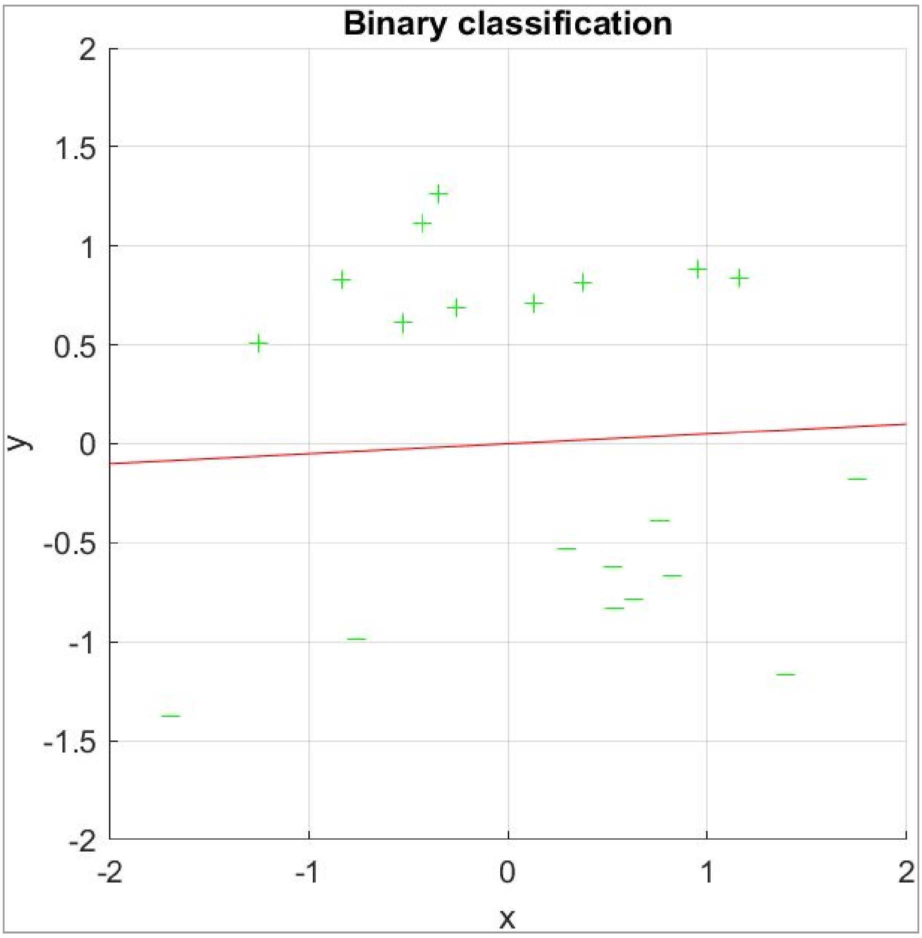
- For this data I used the **initial classifier with a vertical line**.
- Learning rate is represented in code as **n**.
- Following the pseudocode instruction, n may have values **between 0 and 1**.

Learning rate = 0.9:

- **Chart of Bias and Weight:**



- **Final cycle plot:**

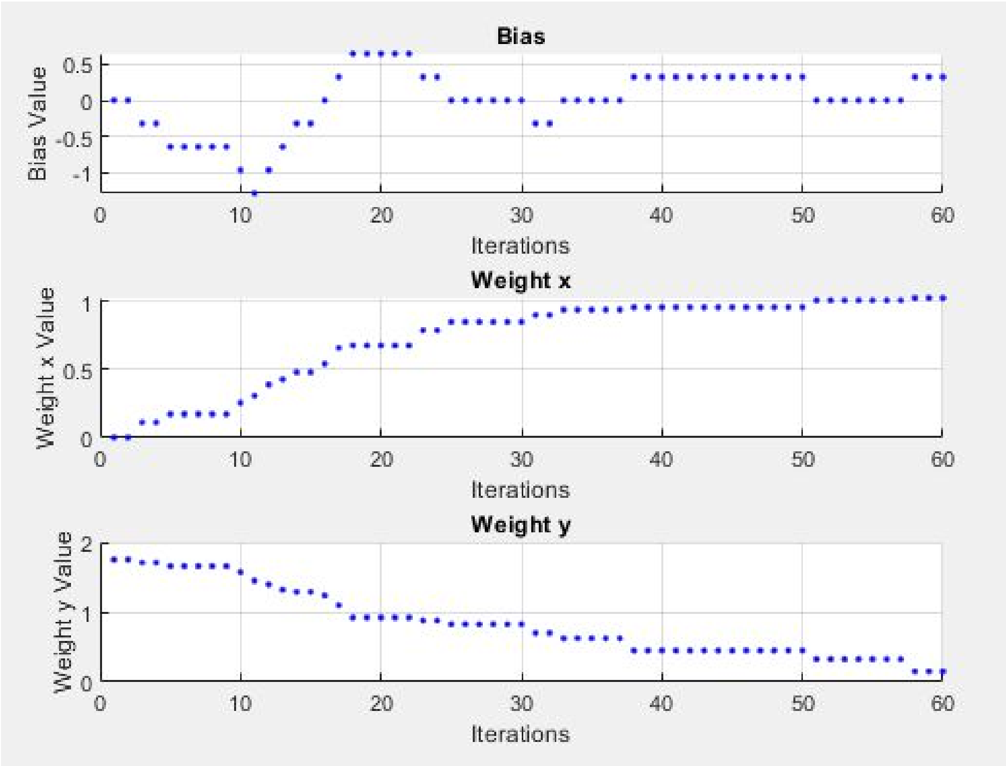


- **Final Values:**

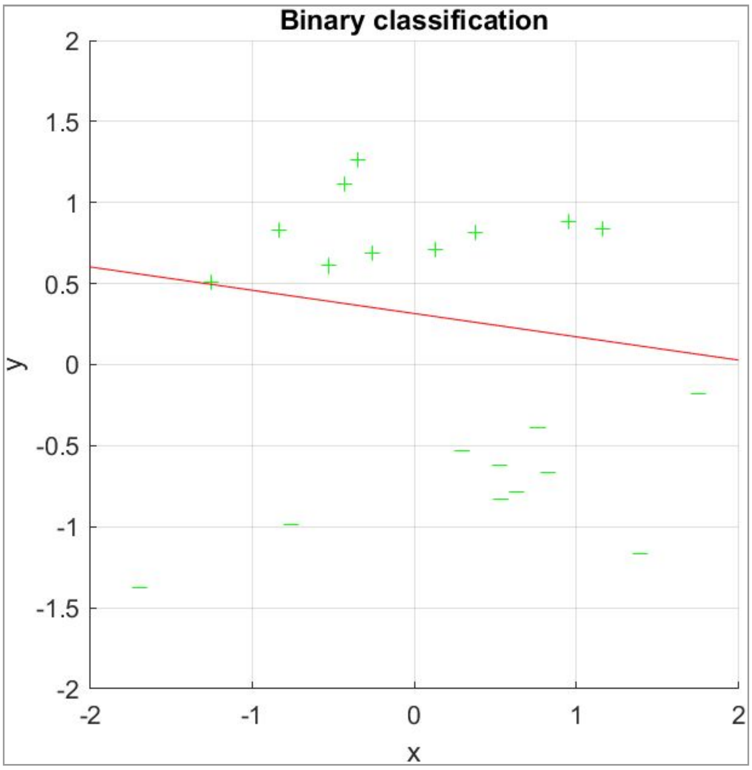
Line	Vertical
Learning Rate	0,9
Bias	0
Weight x	3,5758022951
Weight y	0,1785101821
Cycles	3
Errors	16
Time	4,586988

Learning rate = 0.1:

- **Chart of Bias and Weight:**



- **Final cycle plot:**

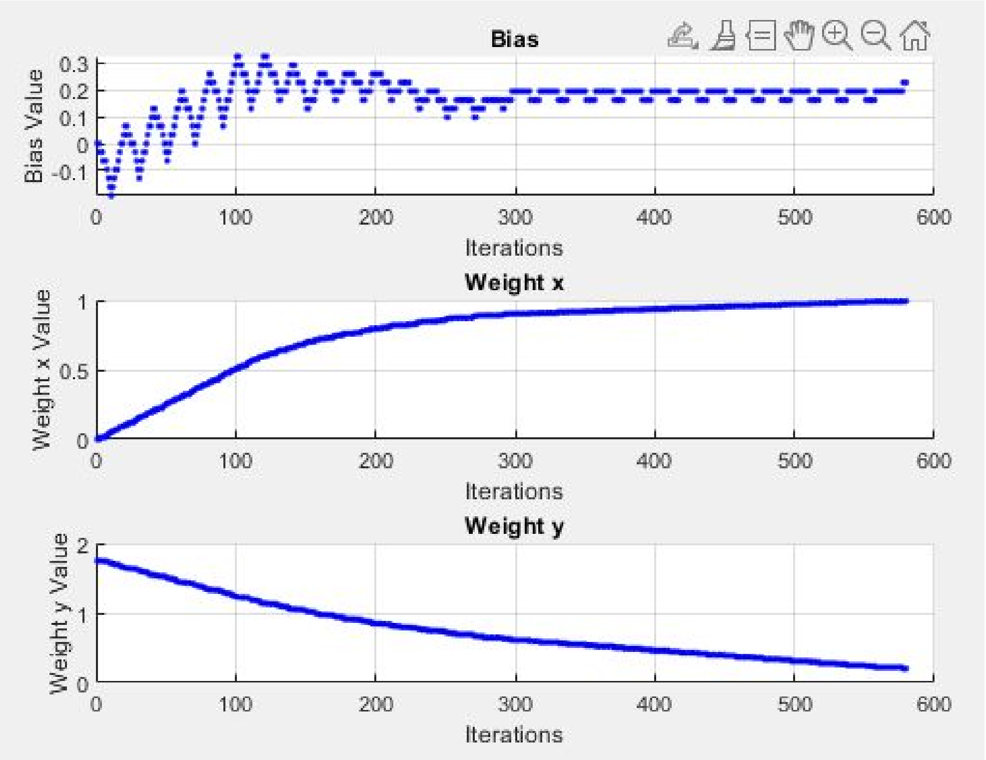


- **Final Values:**

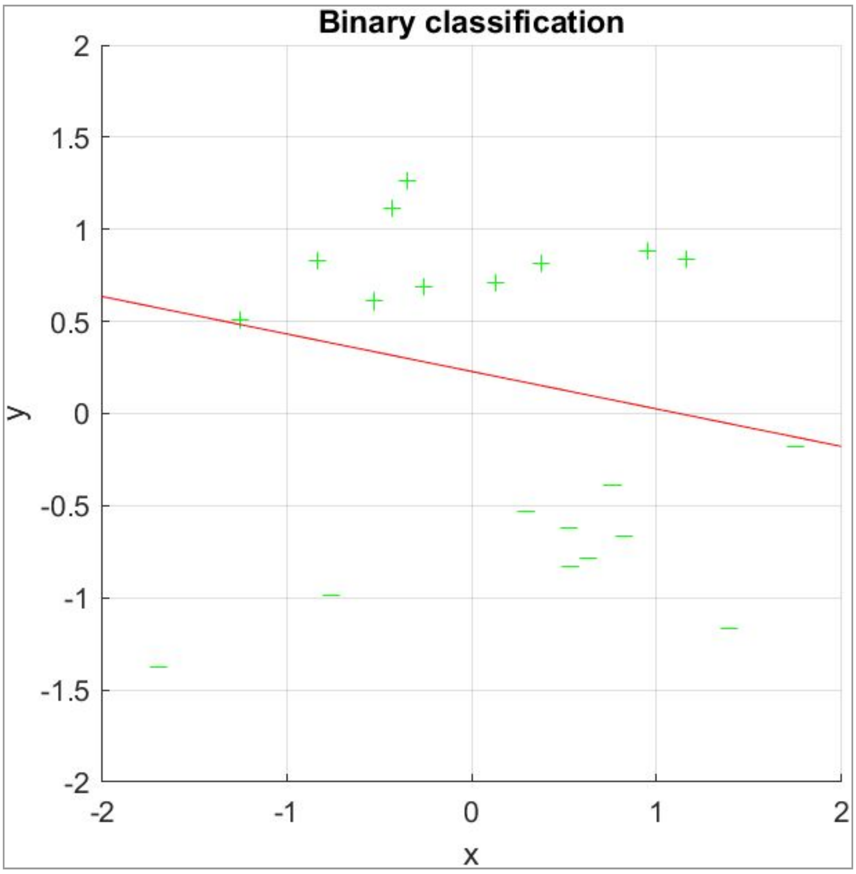
Line	Vertical
Learning Rate	0,1
Bias	0,3241140492
Weight x	1,0275765096
Weight y	0,1476190586
Cycles	3
Errors	10
Time	3,444646

Learning rate = 0.01:

- **Chart of Bias and Weight:**



- **Final cycle plot:**



- **Final Values:**

Line	Vertical
Learning Rate	0,01
Bias	0,2268798345
Weight x	0,9916939343
Weight y	0,2016442710
Cycles	29
Errors	147
Time	62,765667

3.Results with different initial settings for the classifier

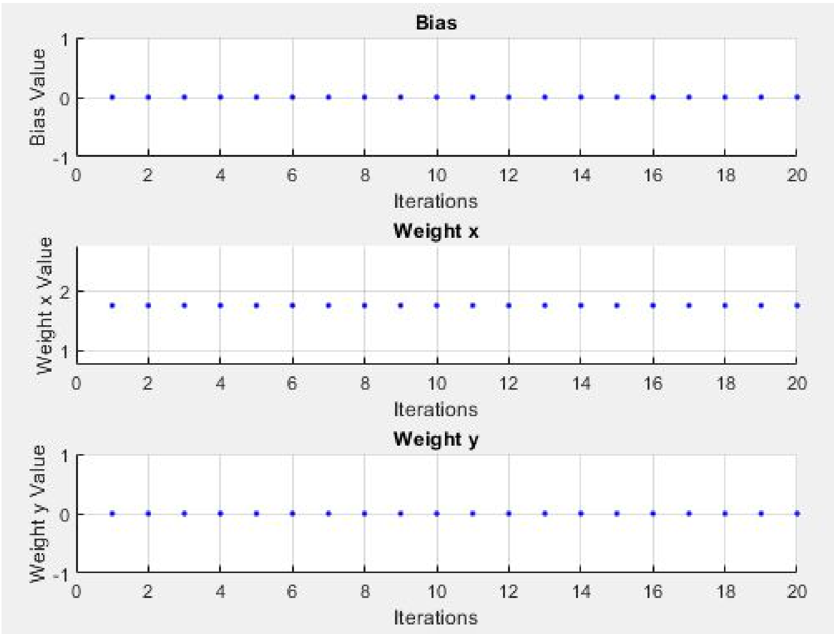
I check the different results by changing the initial line of weights with the learning rate = 0.1 since it is the one that has given me the best results.

Horizontal line:

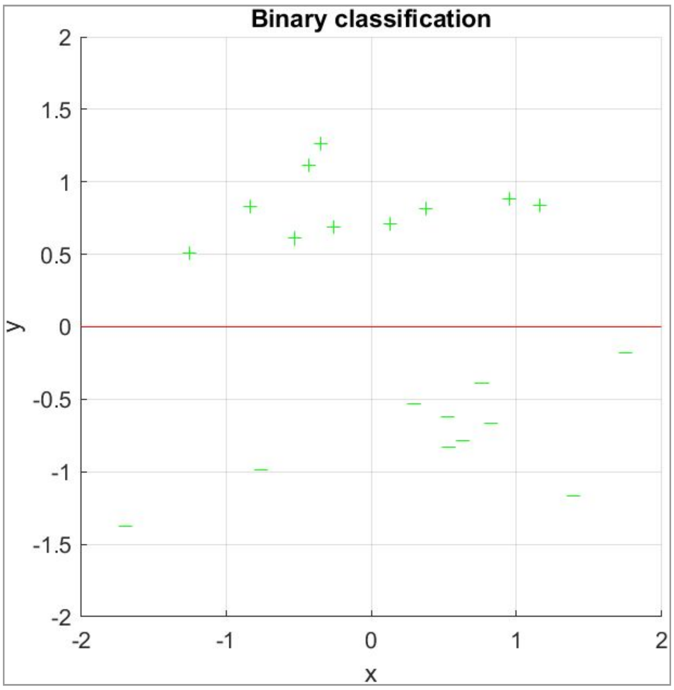
- **Code:**

```
w=[0;max(max(x))];
```

- **Chart of Bias and Weight:**



- **Plot after 1 cycle:**



- **Final Values:**

```
***************** CYCLE 1 ******************

False Negative: 0
False Positive: 0
True Negative: 10
True Positive: 10

Total Errors in this cycle: 0
ACCURACY = 1.00


Errors in TOTAL: 0
Elapsed time is 1.273207 seconds.
```

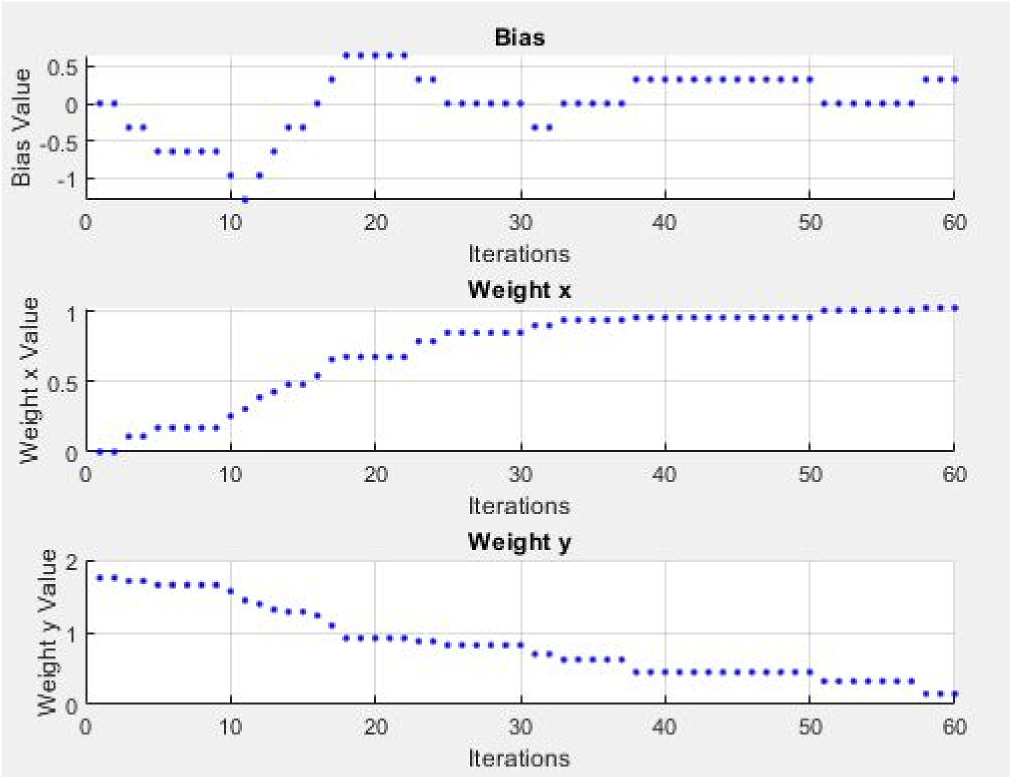
Line	Horizontal
Learning Rate	0,1
Bias	0
Weight x	1,7517495258
Weight y	0
Cycles	1
Errors	0
Time	1,171146

Vertical line:

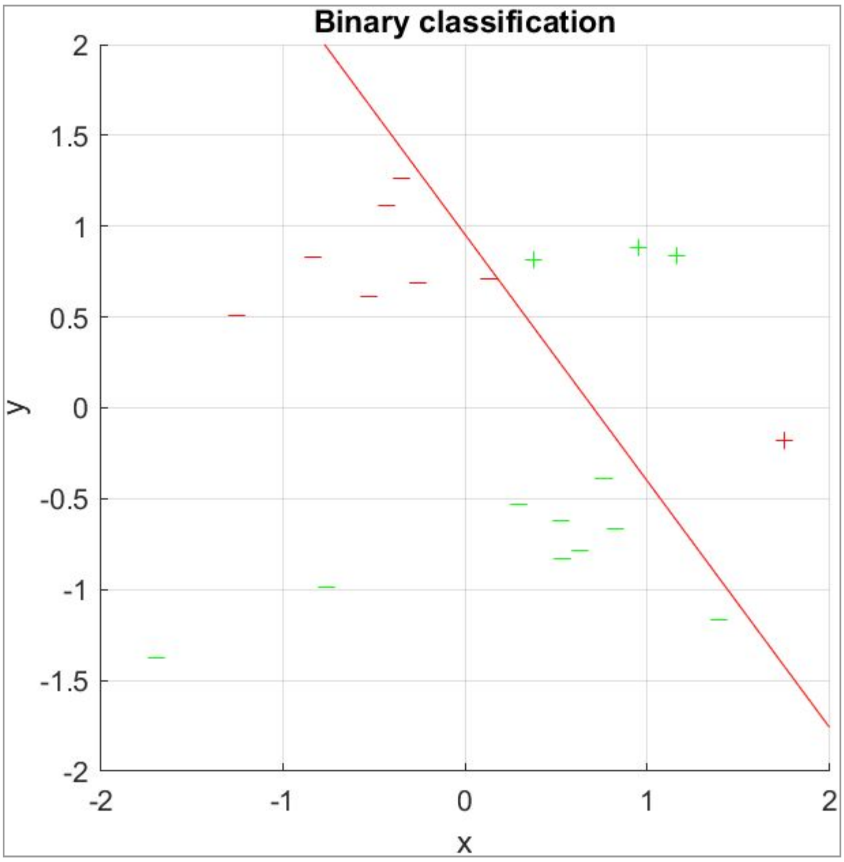
- **Code:**

```
w=[max(max(x));0];
```

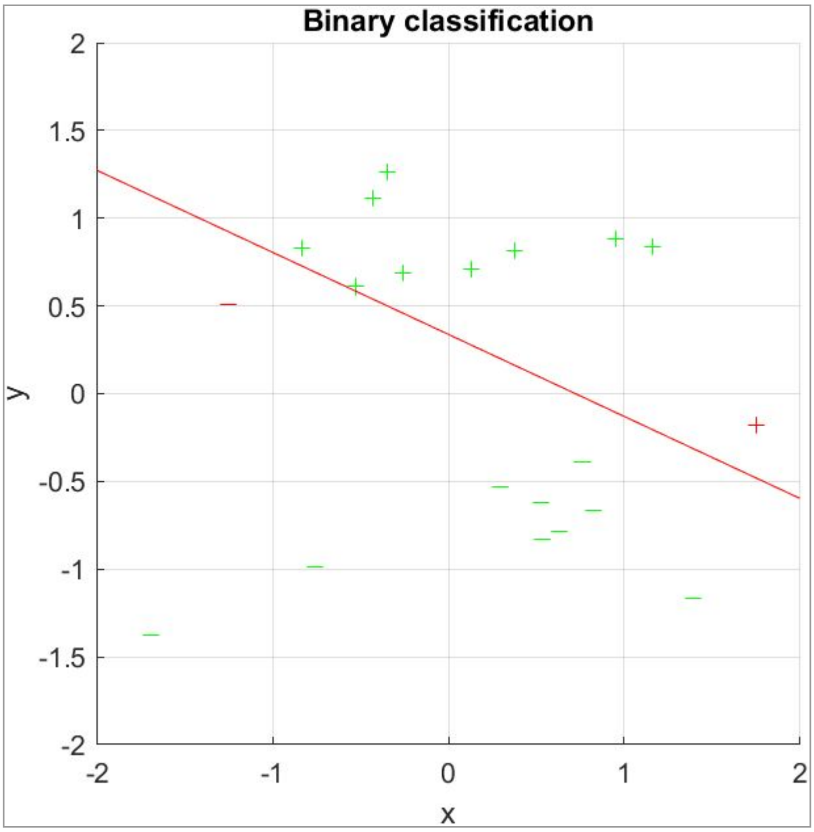
- **Chart of Bias and Weight:**



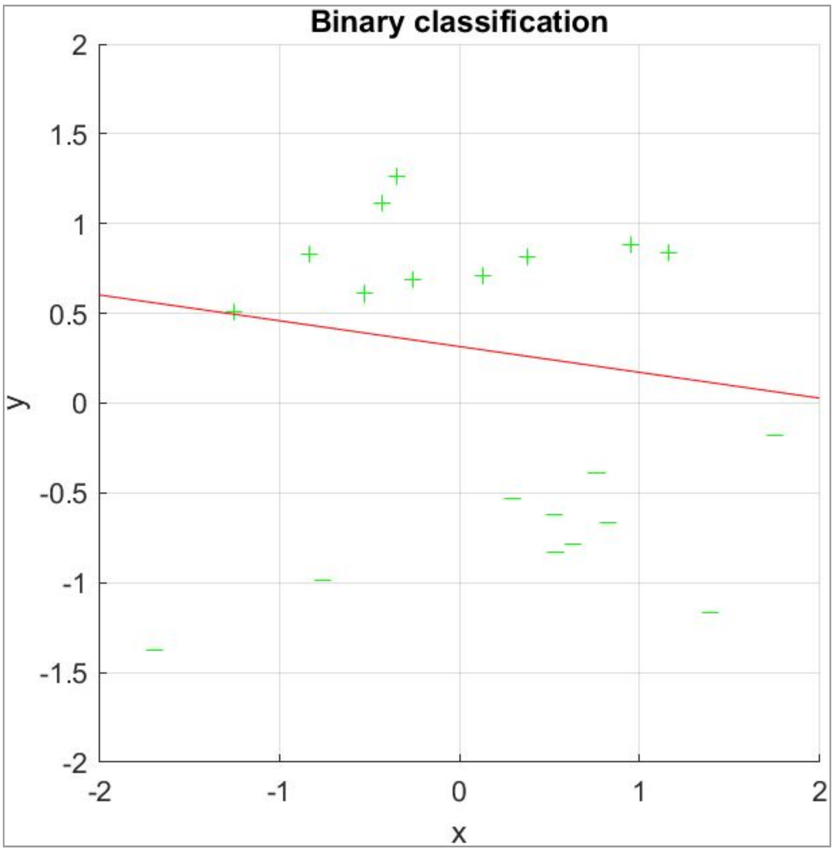
- **Plot after 1 cycle:**



- **Plot after 2 cycle:**



- **Plot after 3 cycle:**



- **Final Values:**

```
****************** CYCLE 1 ******************

False Negative: 7
False Positive: 1
True Negative: 9
True Positive: 3

Total Errors in this cycle: 8
ACCURACY = 0.60


****************** CYCLE 2 ******************

False Negative: 1
False Positive: 1
True Negative: 9
True Positive: 9

Total Errors in this cycle: 2
ACCURACY = 0.90


****************** CYCLE 3 ******************

False Negative: 0
False Positive: 0
True Negative: 10
True Positive: 10

Total Errors in this cycle: 0
ACCURACY = 1.00


Errors in TOTAL: 10
Elapsed time is 3.693055 seconds.
```

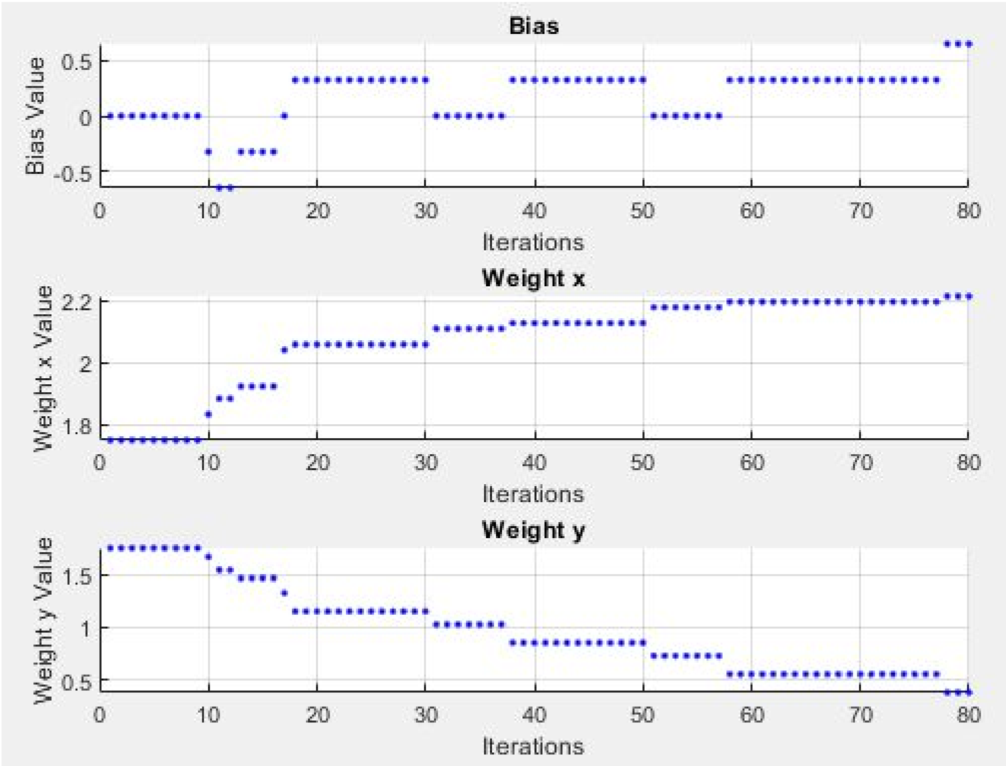
Line	Vertical
Learning Rate	0,1
Bias	0,3241140492
Weight x	1,0275765096
Weight y	0,1476190586
Cycles	3
Errors	10
Time	3,444646

Line passing through I and III quadrants:

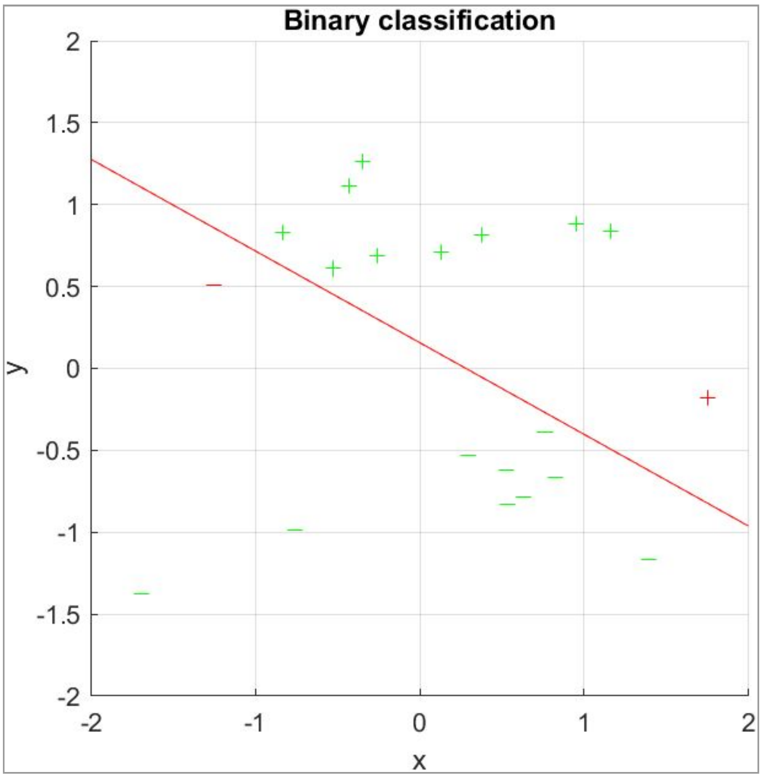
- **Code:**

```
w=[max(max(x));max(max(x))];
```

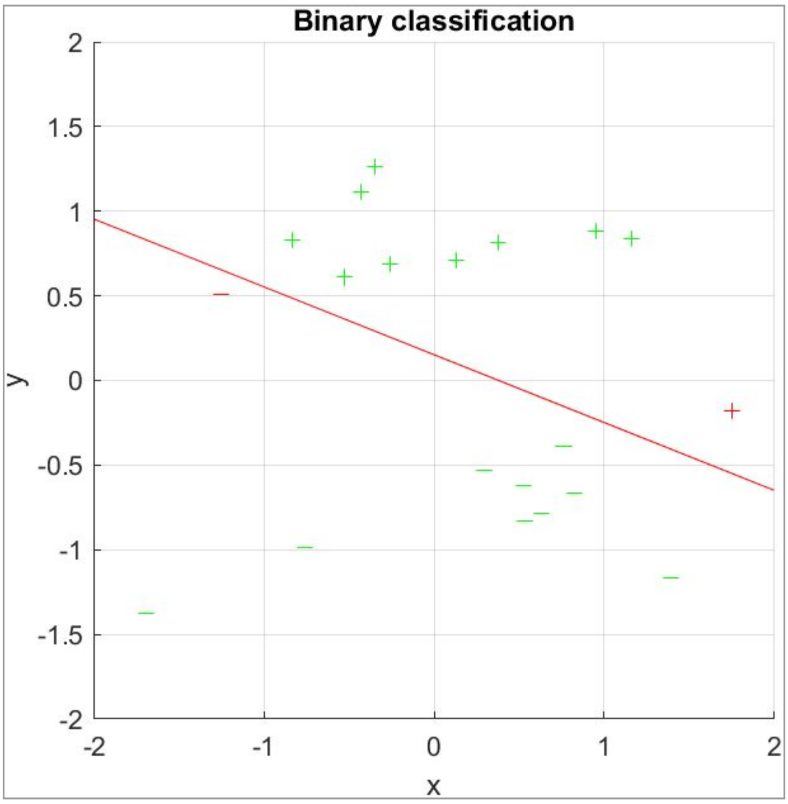
- **Chart of Bias and Weight:**



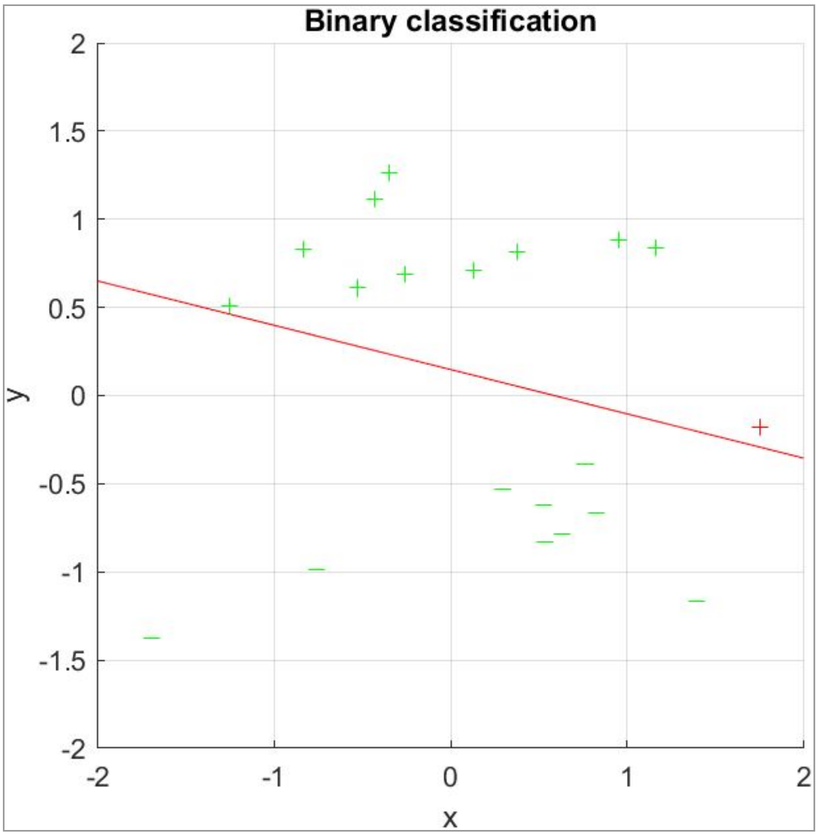
- **Plot after 1 cycle:**



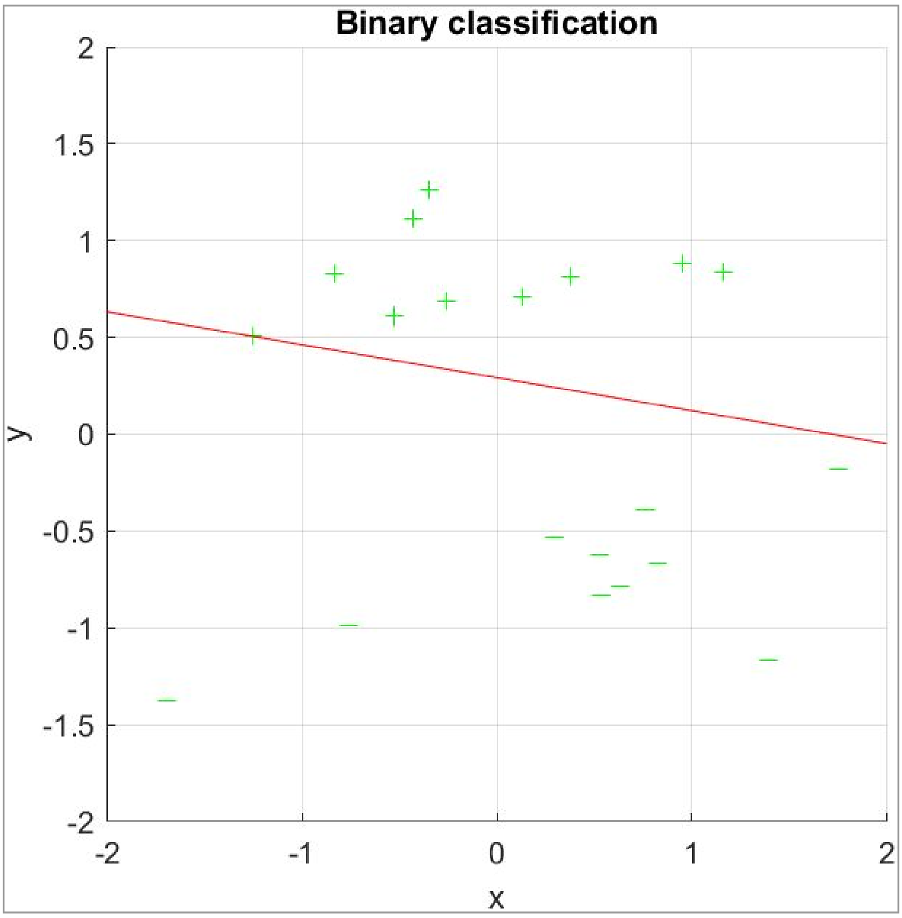
- **Plot after 2 cycle:**



- **Plot after 3 cycle:**



- **Plot after 4 cycle:**



- **Final Values:**

```
***************** CYCLE 1 ******************

False Negative: 1
False Positive: 1
True Negative: 9
True Positive: 9

Total Errors in this cycle: 2
ACCURACY = 0.90


***************** CYCLE 2 ******************

False Negative: 1
False Positive: 1
True Negative: 9
True Positive: 9

Total Errors in this cycle: 2
ACCURACY = 0.90


***************** CYCLE 3 ******************

False Negative: 0
False Positive: 1
True Negative: 9
True Positive: 10

Total Errors in this cycle: 1
ACCURACY = 0.95


***************** CYCLE 4 ******************

False Negative: 0
False Positive: 0
True Negative: 10
True Positive: 10

Total Errors in this cycle: 0
ACCURACY = 1.00


Errors in TOTAL: 5
Elapsed time is 5.124378 seconds.
```

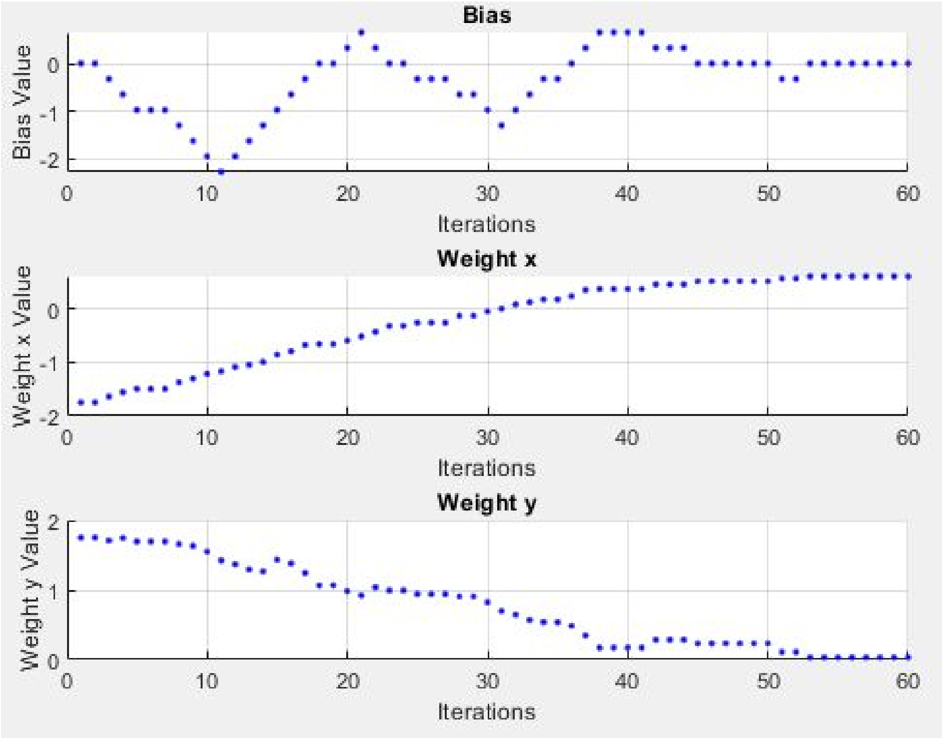
Line	I and III quadrants
Learning Rate	0,1
Bias	0,6482280985
Weight x	2,2144189300
Weight y	0,3765587092
Cycles	4
Errors	5
Time	4,812821

Line passing through II and IV quadrants:

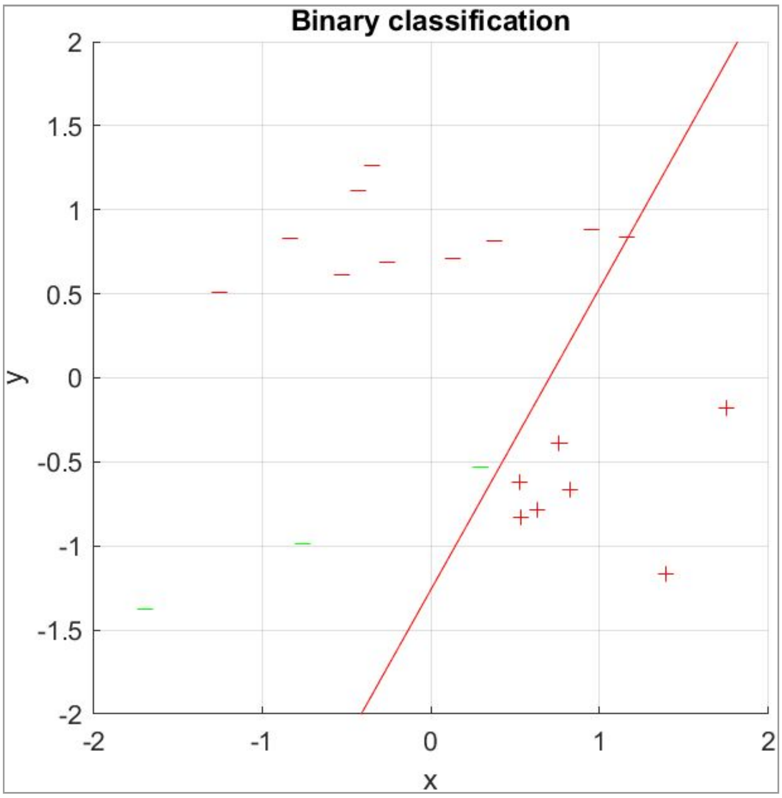
- **Code:**

```
w=[max(max(x));-(max(max(x)))];
```

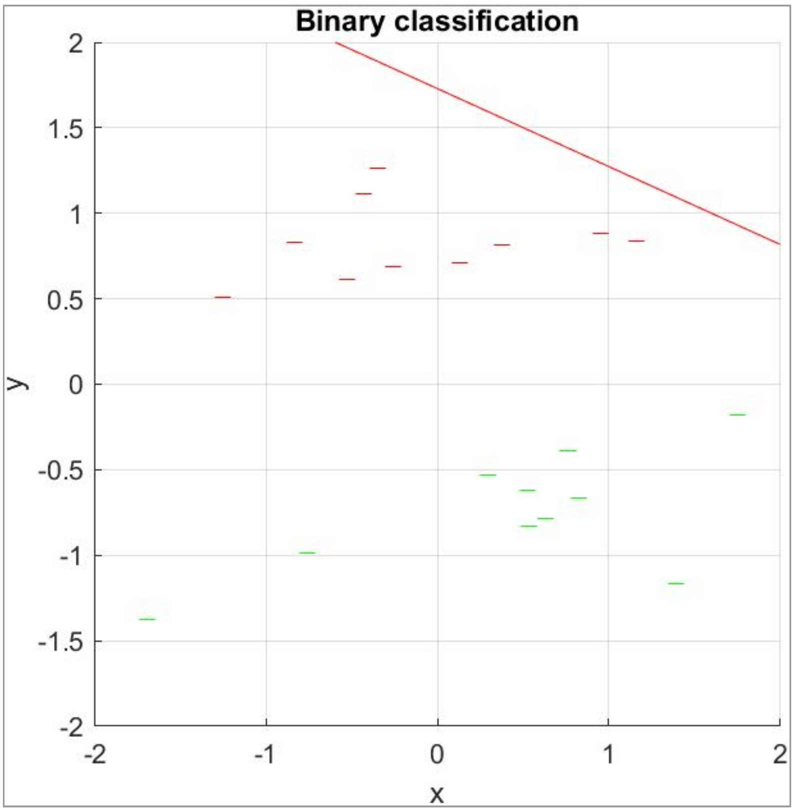
- **Chart of Bias and Weight:**



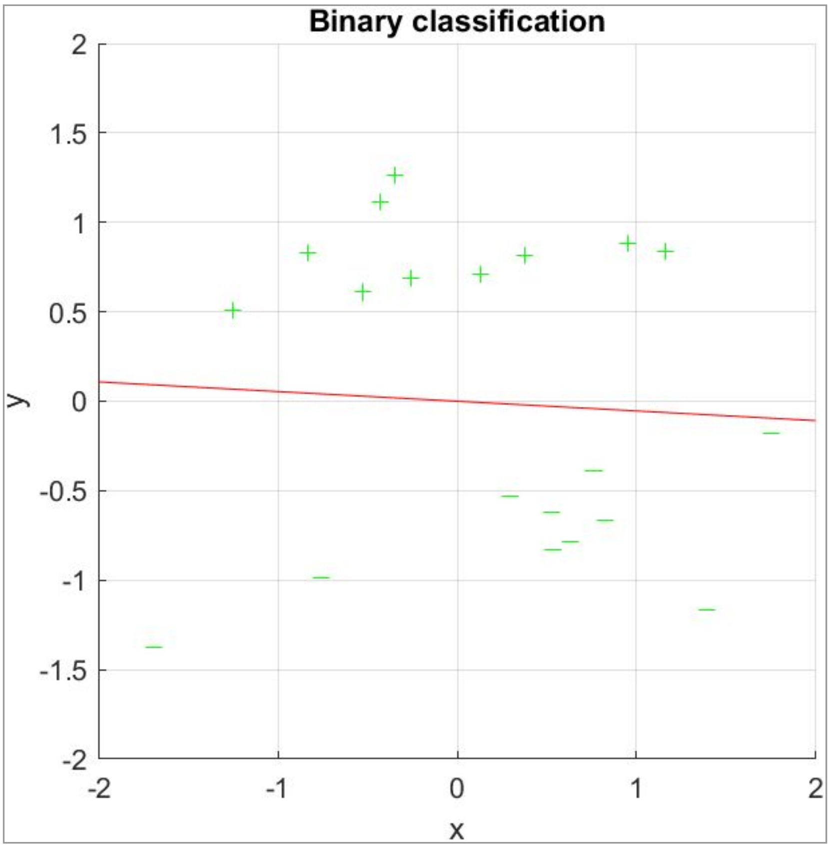
- **Plot after 1 cycle:**



- **Plot after 2 cycle:**



- **Plot after 3 cycle:**



- **Final Values:**

```
****************** CYCLE 1 ******************

False Negative: 10
False Positive: 7
True Negative: 3
True Positive: 0

Total Errors in this cycle: 17
ACCURACY = 0.15


****************** CYCLE 2 ******************

False Negative: 10
False Positive: 0
True Negative: 10
True Positive: 0

Total Errors in this cycle: 10
ACCURACY = 0.50


****************** CYCLE 3 ******************

False Negative: 0
False Positive: 0
True Negative: 10
True Positive: 10

Total Errors in this cycle: 0
ACCURACY = 1.00


Errors in TOTAL: 27
Elapsed time is 3.815598 seconds.
```

Line	II and IV quadrants
Learning Rate	0,1
Bias	0
Weight x	0,6096012385
Weight y	0,0329673457
Cycles	3
Errors	27
Time	3,510294

4. Final observations

- The **learning rate** is a parameter that greatly influences how our algorithm gets to classify correctly in the shortest possible time. In this way we have verified:

  - ➔ **Higher learning rates**:
    The step size is bigger so it may be faster or maybe not because the step is too high that accurate weights might have not been found and maybe is necessary to check again.

  - ➔ **Lower learning rates:**
    The execution time may be higher because of the small step but it is more secure to not have skipped the accurate weights there won't be needed to check again.

So by trying 0.9,0.1 and 0.01,for this experiment, **the most optimal of them is 0.1** because it takes the lowest time with the lowest errors.

Line	Vertical	Vertical	Vertical
Learning Rate	0,9	0,1	0,01
Bias	0	0,3241140492	0,2268798345
Weight x	3,5758022951	1,0275765096	0,9916939343
Weight y	0,1785101821	0,1476190586	0,2016442710
Cycles	3	3	29
Errors	16	10	147
Time	4,586988	3,444646	62,765667

- Different **initial settings for the classifier** also affects the time to correctly classify the data, for example, **the optimal one for this case is to initialize the weights vector to an horizontal line** that passes through the origin of coordinates. It occurs because it only takes a cycle to the perceptron to realize that it is the optimal one to classify between positive and negative data.

Line	Horizontal	Vertical	I and III quadrants	II and IV quadrants
Learning Rate	0,1	0,1	0,1	0,1
Bias	0	0,3241140492	0,6482280985	0
Weight x	1,7517495258	1,0275765096	2,2144189300	0,6096012385
Weight y	0	0,1476190586	0,3765587092	0,0329673457
Cycles	1	3	4	3
Errors	0	10	5	27
Time	1,171146	3,444646	4,812821	3,510294

With this experiment, we have been able to appreciate how our binary classification algorithm learns and turns out to achieve its objective successfully, as we can see for example in the following image, where it has **finally classified the data correctly** as well as all the executions, independently of the learning rate and initialization of the classifier.