

Introduction

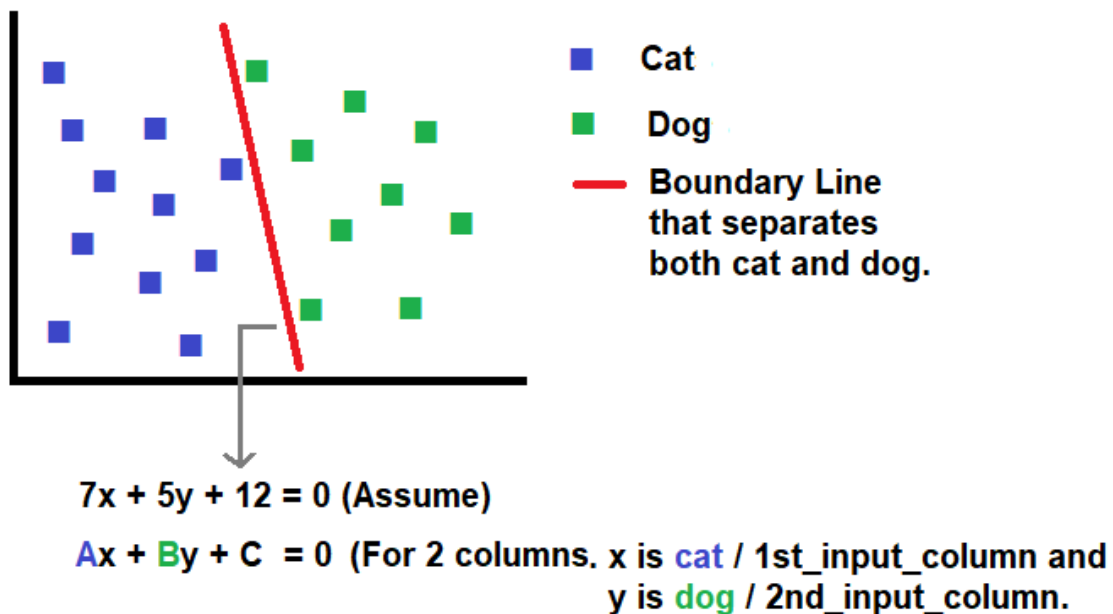
Logistic Regression : When the output column is Yes/No, 1/0, Positive/Negative.

The equation for a straight line is $Ax + By + C = 0$ and from this we can derive $y = mx + b$, where $m = -A/B$, $b = -C/B$

In Linear Regression we worked with $y = mx + b$ but here in Logistic Regression we'll work with the General Equation $Ax + By + C = 0$ and find the correct A, B and C. From the below picture we can see that we need to find the **Boundary Line** for $A = 7$, $B = 5$ and $C = 0$ that separates both classes(cat and dog) efficiently.

In linear regression we find a line that goes through all the points with minimized Error/Loss Function. But here we need to find a line (for 2D its line, for 3D its Plane and for 4 to nD its Hyperplane) that **separates** all the input columns with minimized error/loss.

Assume, we've two input columns, cat and dog. Now we've to classify them. (The Left of **Boundary Line is all Cat and Right is all Dog)**

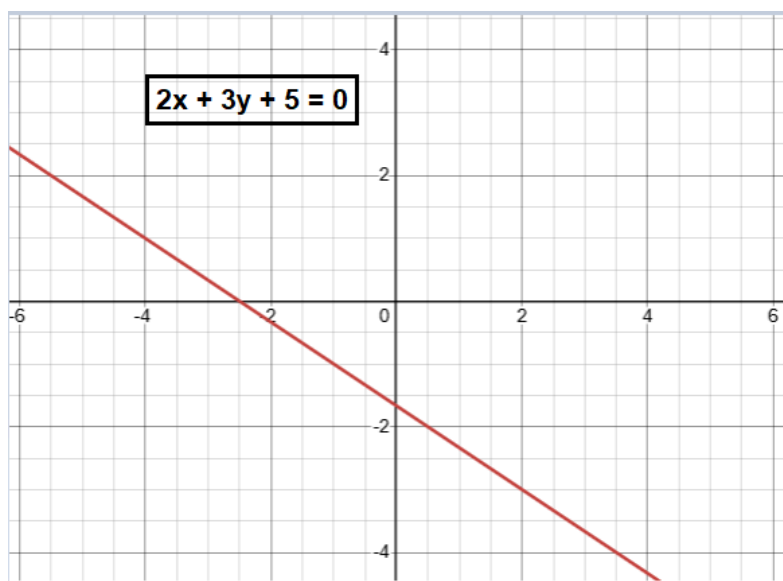





So for 3 input columns : $Ax + By + Cz + D = 0$ (x, y, z are **input columns** i.e. x_1, x_2, x_3)

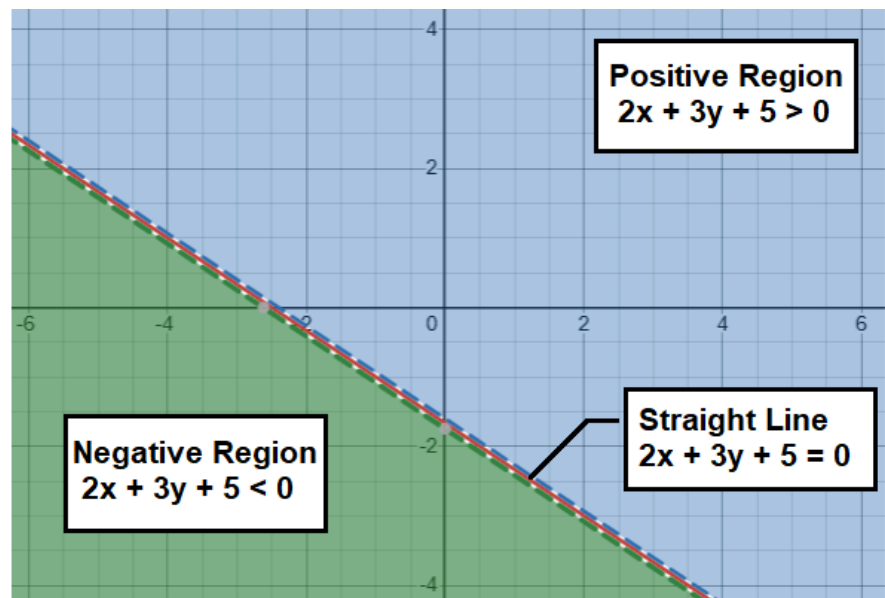
Boundary Line

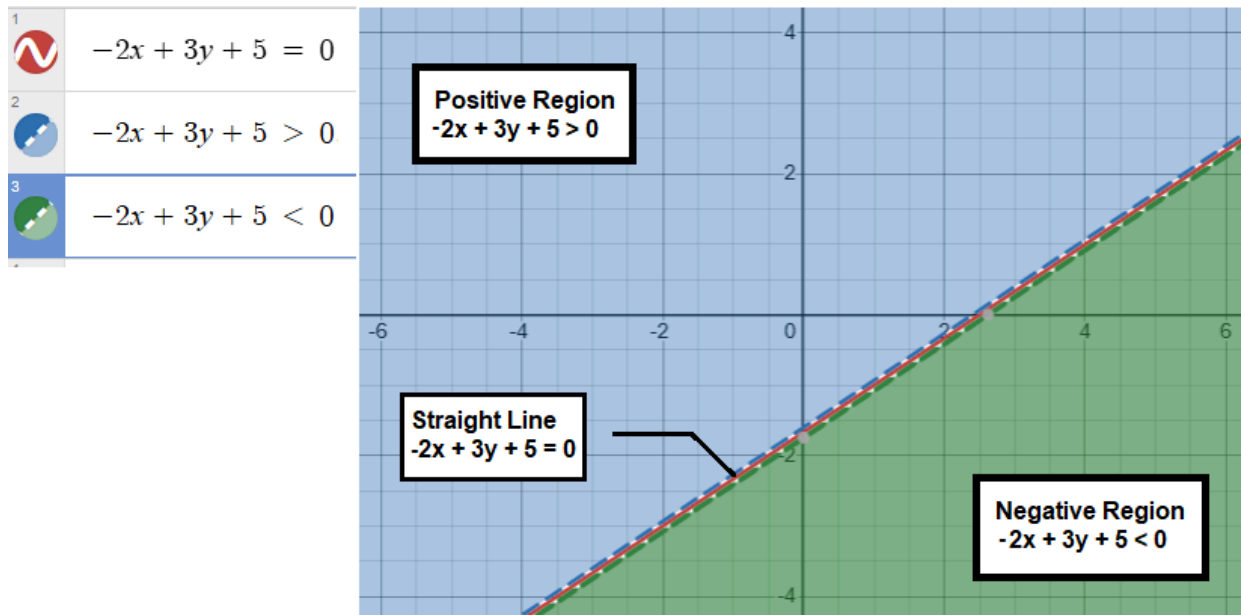
1) **Positive** and **Negative** region of a straight line :

We can't directly say, the right side of a straight line is actually right, because it can be left for that equation of straight line. So we mathematically say, **positive** and **negative** region of a straight line e.g. for $2x + 3y + 5 = 0$ we get :

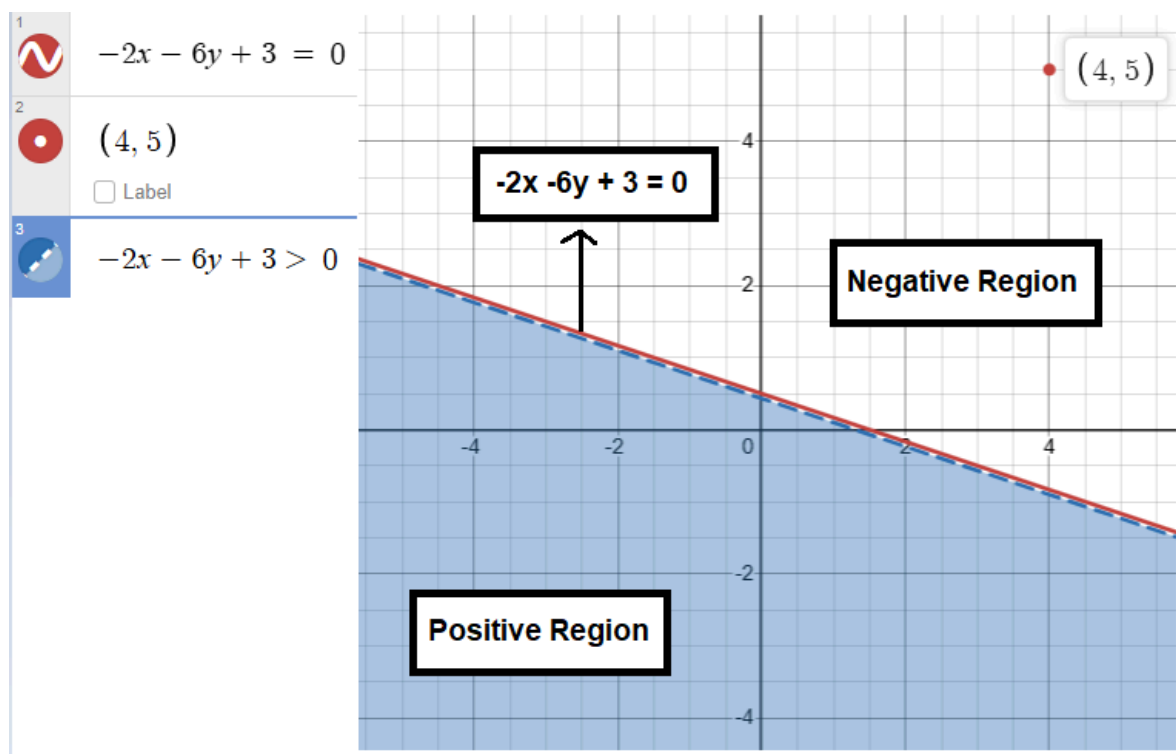


1		$2x + 3y + 5 = 0$
2		$2x + 3y + 5 > 0$
3		$2x + 3y + 5 < 0$





2) Is the point (4,5) in the Positive or Negative Region of the line/equation $-2x - 6y + 3 = 0$?



So (4,5) in the Negative Region of the line $-2x - 6y + 3 = 0$.

- 3) From the above step 2, move the Boundary Line ($-2x - 6y + 3 = 0$) such that **Negative Region's Point (4, 5)** lies in the Positive Region.
-

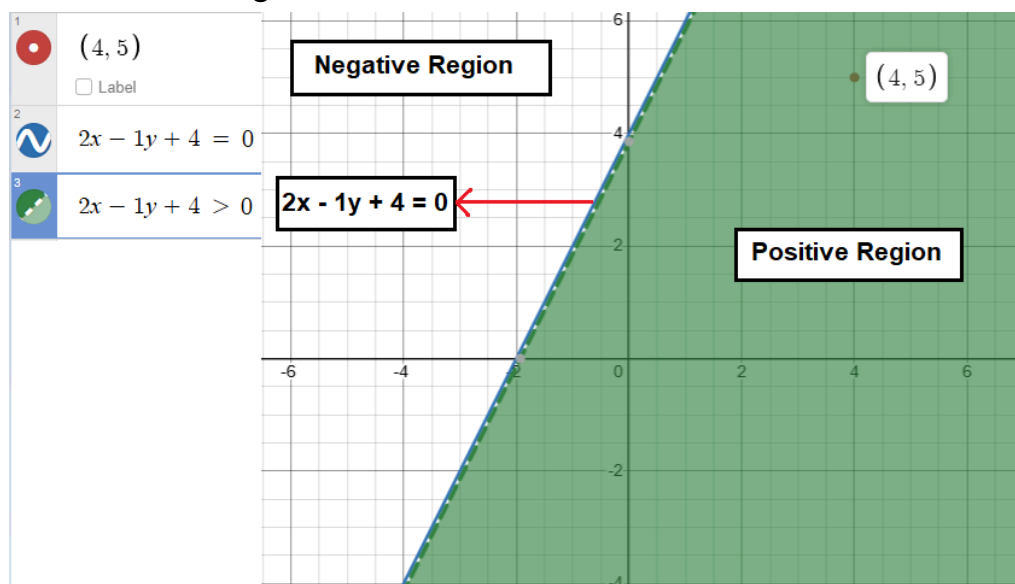
Assuming the Point (4, 5) actually belongs to the Positive Region but that stupid line said “NO, it belongs to the negative region”. So now we’ve to correct the mistake i.e. make sure the point lies in the Positive Region of that line. Obviously we can’t change the position of the point (4, 5) but we can change the equation/line in such a way so that the **New Line’s** positive region has the point (4, 5).

So Actual Output is Positive (region) but the Line saying it is Negative, in that case : (We need to move towards **Positive** region to make sure the Point lies in the **Positive** region, so we do + below)

New Line = Old Line + Point

	-2	-6	3	(Old Line's A, B, C)
(+)	4	5	1	(Must add 1 as the Perceptron rules)
<hr/>				
	2	-1	4	

So our new line equation is $2x - 1y + 4 = 0$. Lets see if the point (4, 5) lies in the Positive region or not :



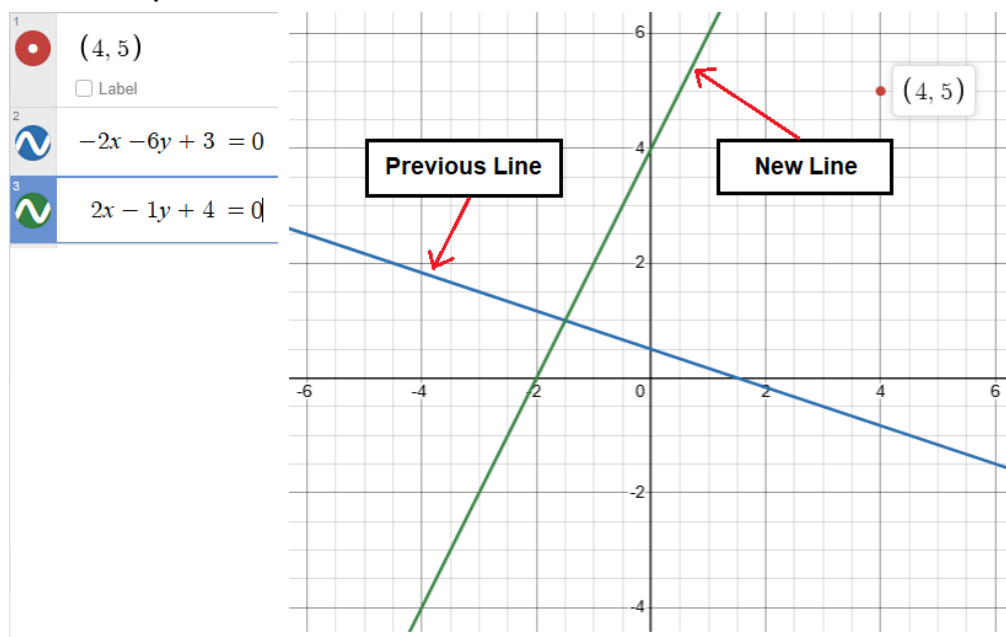
As we can see, now the Point (4, 5) lies in the Positive Region in our New Line.

[But if the Actual Output is Negative (region) but the Line saying it is Positive, in that case :

New Line = Old Line - Point (We need to move towards **Negative** region to make sure the Point lies in the **Negative** region, so we did -)]

4) Learning Rate :

From step 3 and 4 :



As you can see, the difference of the sudden move from Previous Line to New Line is huge. In Machine Learning we don't make such big move in one step. We move slowly from the Previous Line/State with "**Learning Rate**" :

$$\text{New Line} = \text{Old Line} - \text{Learning_Rate} * \text{Point}$$

$$\text{New Line} = \text{Old Line} + \text{Learning_Rate} * \text{Point}$$

Equation for Perceptron

input		output
x1	x2	y
cgpa	iq	placed
3.3	20	1
2.0	30	1
3.5	10	0

For **any row**, e.g. the first red boxed row :

$$Ax + By + C = 0$$

$$\Rightarrow w_1x_1 + w_2x_2 + w_0 = 0$$

$w_0 * 1 = 1$. So let's just add 1 as the first input column :

input			output
x0	x1	x2	y
x0	cgpa	iq	placed
1	3.3	20	1
1	2.0	30	1
1	3.5	10	0

so, $w_0x_0 + w_1x_1 + w_2x_2 = 0$ (Now w_0 here is A, w_1 is B..)

For n input columns : $w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n = 0$

$$\Rightarrow [w_0 \ w_1 \ w_2 \ \dots \ w_n] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = 0$$

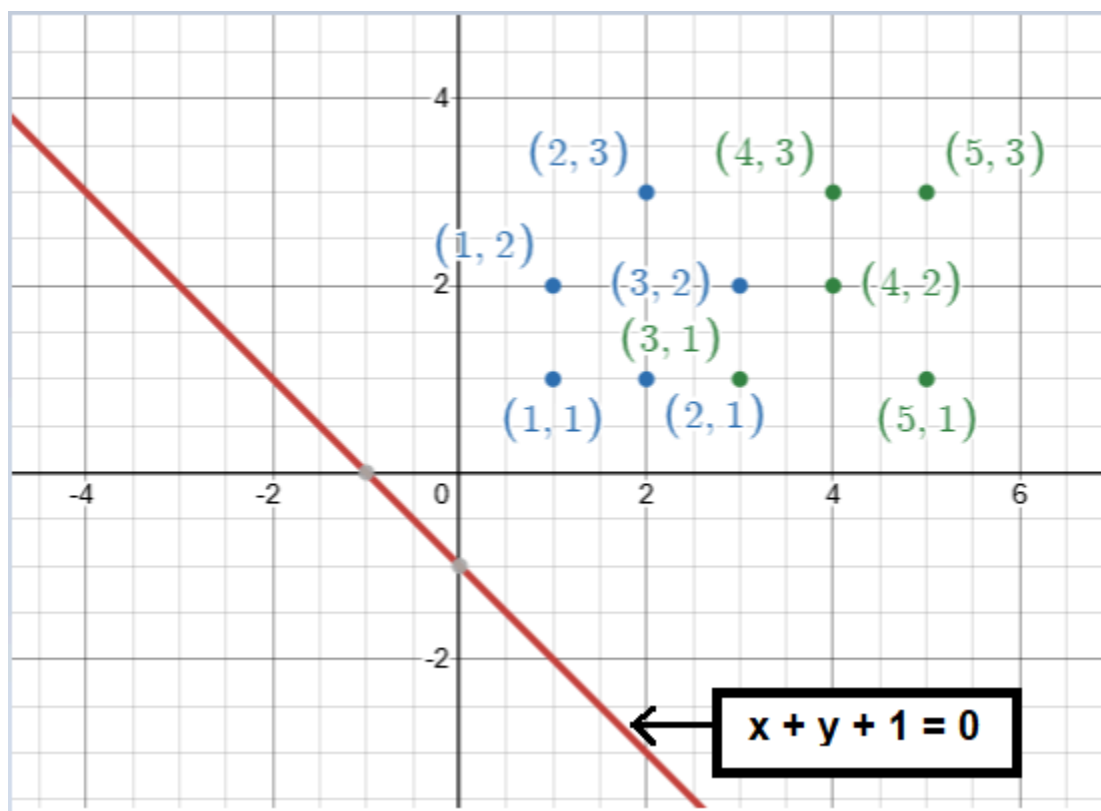
$$\Rightarrow \mathbf{WX} = 0 \quad \text{where } \mathbf{W} = [w_0 \ w_1 \ w_2 \ \dots \ w_n] \text{ and}$$

$$\mathbf{X} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

This $\mathbf{WX} = 0$ is the Line Equation for **one point** i.e. **one row**. And the output of \mathbf{WX} tells us where the Point falls, Positive or Negative region.

WARNING : Here, X is the First Input Row. So x_0 is a SCALAR value here for the first row, not the entire column. Similar for x_1 and x_2 .

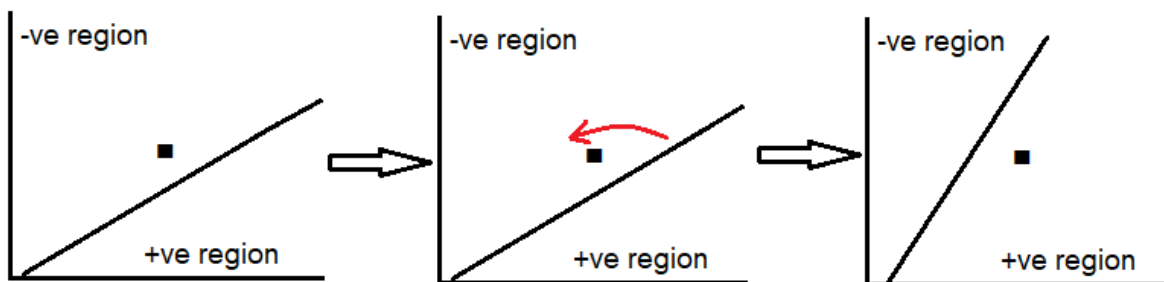
Wait, X is the input column values for one row, then what is the A, B, C ... i.e. W at the beginning? It's 1 i.e. $[1, 1, 1, \dots, 1]$. So initially we draw the line, $x + y + 1 = 0$:



and then pick a random Point(values of Input columns for that **one randomly picked row**) and check if that Point is in the correct region or not by “**what is the actual output(given in the output column) and what is my line saying(after putting the coordinate values of that one row in the initial line equation($x + y + 1 = 0$) and getting a positive/negative value which tells us in which region our Point laying right now?)**”

i) If Actual output(**1**) saying that my Point is in the **Positive Region** but the current line equation that we’ve drawn on our dataset saying its in the **Negative region** :

In that case I’ve to move My (Boundary) Line Equation **towards the Negative region** so My (Boundary) Line can identify that Point as Positive which is the actual output.



$$W_{\text{new}} = W_{\text{old}} + \eta * X[\text{random_index}]$$

η = Learning Rate.

ii) If Actual output(**0**) saying that my Point is in the **Negative region** but the current line equation saying its in the **Positive Region** :

In that case I’ve to move My (Boundary) Line Equation towards the Positive region so My (Boundary) Line can identify that Point as Negative which is the actual output.



$$W_{\text{new}} = W_{\text{old}} - \eta * X[\text{random_index}]$$

η = Learning Rate.

iii) if the Actual Output and My Line saying the same region which is either both Positive or Negative :

In that case we won't bring any changes to our current line.

$$W_{\text{new}} = W_{\text{old}}$$

We can write the above 3 conditions in one equation :

y = Actual Binary Output by the dataset which is either 1 or 0.

\hat{y} = Binary Output by our Line. What **My Current Line(WX)** drawn on the **dataset** saying Where is the Point? If in the Positive region, then 1. If in the Negative region, then 0.

$$W_{\text{new}} = W_{\text{old}} + (y - \hat{y}) * \eta * X[\text{random_index}]$$

$y - \hat{y} = 0$, when y and \hat{y} are same which is the **(iii)** condition.

$y - \hat{y} = 1$ (actual output is Positive Region) - 0 (line saying Negative Region) = 1, which is the **(i)** condition.

$y - \hat{y} = 0$ (actual output is Negative Region) - 1 (line saying Positive Region) = -1, which is the **(ii)** condition.

So, each time for a randomly picked Point, we will calculate :

$$W_{\text{new}} = W_{\text{old}} + (y - \hat{y}) * \eta * X[\text{random_index}]$$

Perceptron Algorithm

- 1) Initialize W to [1, 1, 1]
- 2) Usually we do 1000 epoch and for each epoch, we randomly pick a Point i.e. Input Row from the dataset. Then do the below calculation

where the Current Drawn Line may move towards the Positive Region or Negative Region or be as before if the Point is already in the correct region by both actual and line's output.

$$W_{\text{new}} = W_{\text{old}} + (y - \hat{y}) * \eta * X[\text{random_index}]$$

Code Implementation

(Assume x_train and y_train are given)

x_train = x_train but with inserting a new column of 1 as the 1st column
since $W = w_0x_0 + w_1x_1 + w_2x_2 \dots + w_nx_n$

n, m = row_numbers, column_numbers

W = [1] * m # W = [w0, w1, w2, ..., wm] = [intercept, coefficients]

for epoch in range(1000):

```
    # Below all the calculation is for the SINGLE ROW,
    x_train[random_index], y_train[random_index].
    random_index = np.random.randint(low=0, high=n)
    x, y = x_train[random_index], y_train[random_index]
     $\hat{y} = 1$  if np.dot(W, x) > 0 else 0
    W = W + (y -  $\hat{y}$ ) *  $\eta$  * x
```

Summary

- 1) Draw the Boundary Line anywhere over the dataset assigning the initial value of W(A, B, C,).
- 2) Inside a loop, for every epoch, pick a random Point(input rows) and move the Boundary Line for that point if needed, which we discussed in "Perceptron Algorithm".