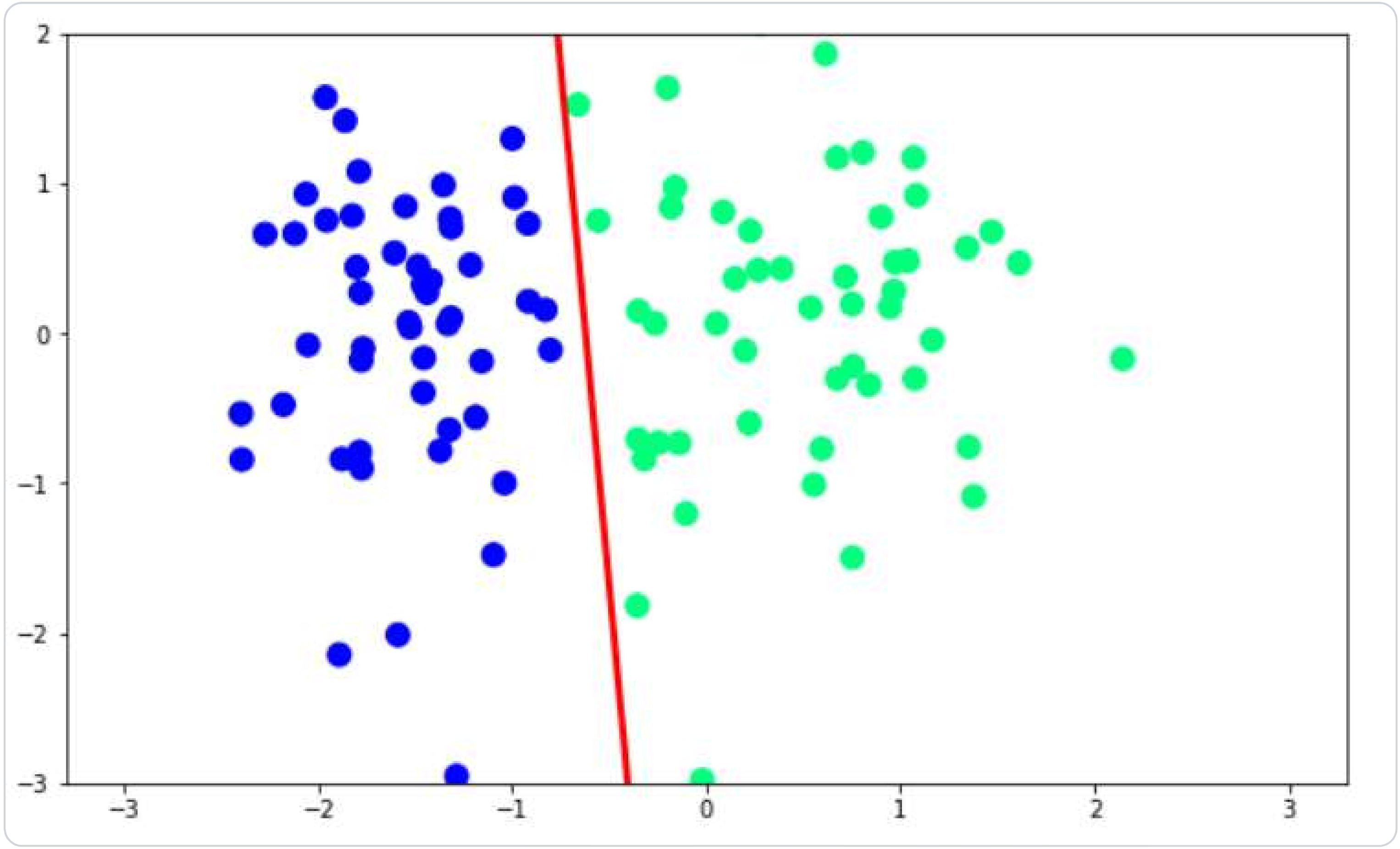


The line in this case is represented by the following equation:

$$Ax_1 + Bx_2 + C = 0$$

In general, it could be considered as

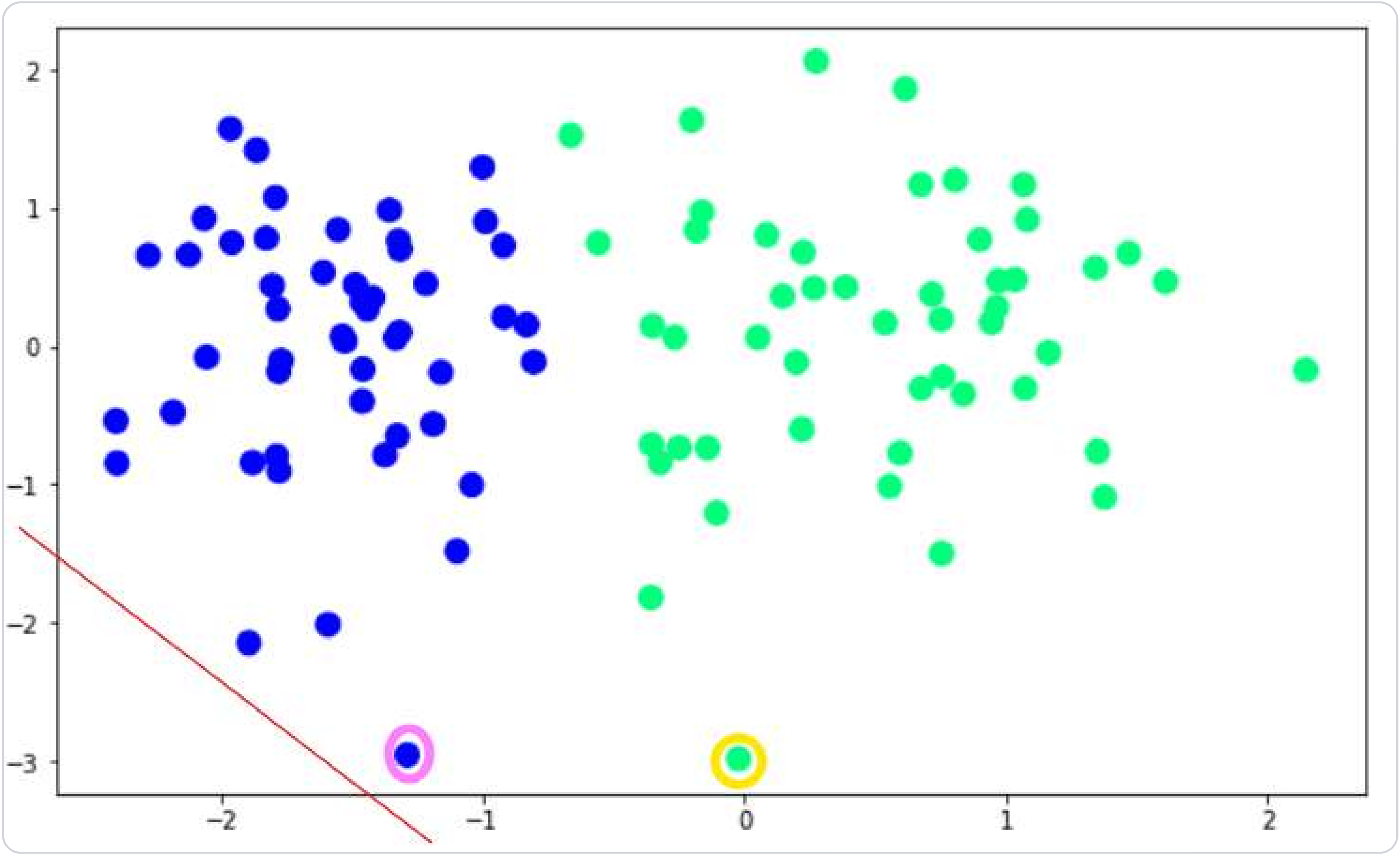
$$Ax_1 + Bx_2 + Cx_3 + Dx_4 + \ldots = 0$$

where,

- $A, B, C...$ are coefficients or weights of the features
- $x_1, x_2, x_3...$ are independent features

Perceptron trick works in the most simple way imaginable. Firstly, you start with random values of A, B and C. You can start with A=B=C=1

For example, we initially start with the red line shown in the figure. Then, we pick a random point and check if the point lies on the correct side of the line or not.



There are two scenarios to consider after we pick a point:

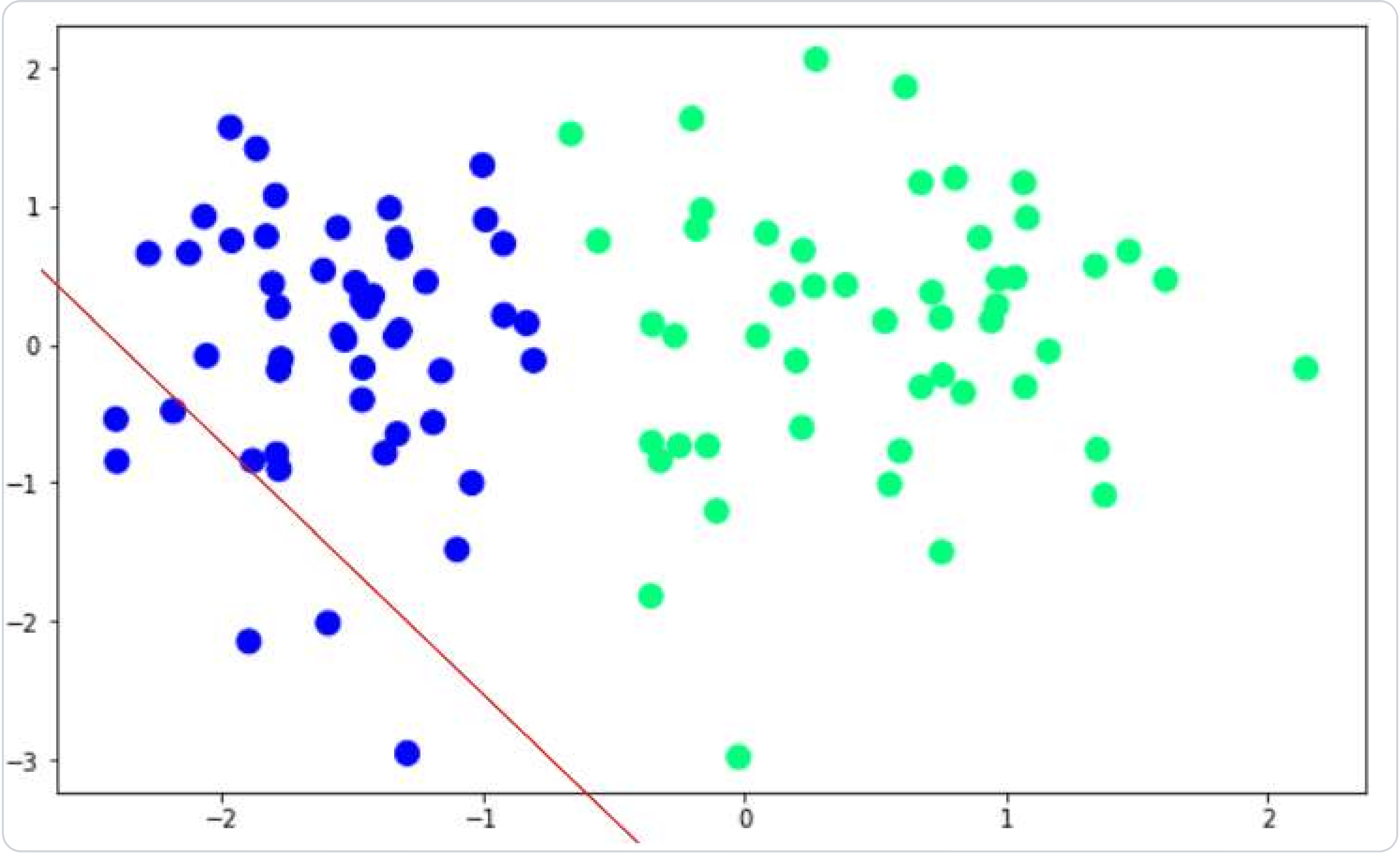
- If a point is **correctly classified**, do nothing
- If a point is **incorrectly classified**, adjust the line in such a way that it is classified.

The above procedure is repeated multiple times in a loop to get the desired line.

> ***As per current example, blue points should be on the left of the line and green must be on the right side of the line.***

Thus, when we pick a point yellow circled in the figure given below, we see that it is correctly classified and we do nothing.

However, when we pick the pink circled point we do some transformations on the line because it is on the wrong side of the line. After applying transformations, the point is correctly classified as shown below:



We apply transformations on line to classify the points correctly. Since how to apply transformations on line is whole another topic I have discussed it in detail in the given article [**Regions & Transformation on Line**]. Therefore, give it a quick read if you aren't aware of how it works.

In short, we add/subtract the coordinates of the data point from the coefficients of the equation. This will make sure the chosen point is in the correct region. However, we do not directly subtract the coordinates from coefficients because it causes the line to change drastically. Thus, to make the changes slow we first multiply coordinated by the learning rate and then add or subtract it according to the situation.

In a nutshell, the process works like this,

- We choose a random data point and check if it is correctly classified
- If it is correctly classified we do nothing
- If it isn't correctly classified, we apply a transformation
- We create a loop and repeat from step 1 for an appropriate number of epochs or until all the data points are picked and correctly classified.

Therefore,

$$\text{new\_coeff} = \text{old\_coeff} - n.\,coordinates$$

where,

$$n \text{ is learning rate}$$

Algorithm

Let's say you have the following data,

X_0	Income(X_1)	Age(X_2)	Loan approval(Y)
1	50000	26	1
1	36000	34	0
1	20000	21	1
1	54000	31	1
1	19000	17	0

Thus, the generalized equation that shows the relationship between this data is given by,

$$W_0X_0 + W_1X_1 + W_2X_2 = 0$$

Therefore, we can write it as follows,

$$\sum_{i=1}^{2} W_iX_i = 0$$

Now, we must loop and in each iteration change the value of weights if the point is wrongly classified. Thus, algorithm looks like this,

- for i in range(epochs):
  - randomly select a student
  - if point is in **negative region** and is required in **positive region:**
    - subtract (coordinate multiplied by learning rate) from old coordinate
  - if point is in **positive region** and is required in **negative region:**
    - add (coordinate multiplied by learning rate) to old coordinate

Here, the given conditions look like as given below:

- if point is in **negative region** and is required in **positive region:**

$$\text{if } X_i \in \text{N and } \sum_{i=1}^{2} W_iX_i >= 0$$

then,

$$W_n = W_o - nX_i$$

- if point is in **positive region** and is required in **negative region:**

$$\text{if } X_i \in \text{P and } \sum_{i=1}^{2} W_i X_i < 0$$

then,

$$W_n = W_o + nX_i$$

You can see that we have to check the condition twice to update the weights. Therefore, to simplify further, we multiply the learning rate with (actual value - predicted value)

Thus, in new algorithm, we do not need to check anything using if condition, we simply replace them by the following one equation:

$$W_n = W_o + n(y_i - \hat{y}_i)X_i$$

Let's consider the following four cases:

CASE	Actual value (y_i)	Predicted(hat{y_i})	Explanation	y_i - hat{y_i}
1	0	0	Correct classification, do nothing	0
2	0	1	point is in **negative region,** so subtract	-1
3	1	0	point is in **positive region,** so add	1
4	1	1	Correct classification, do nothing	0

From, above table, we can see that we don't need to do anything in case 1 and 4. However, in case 2 we must subtract, so according to new formula:

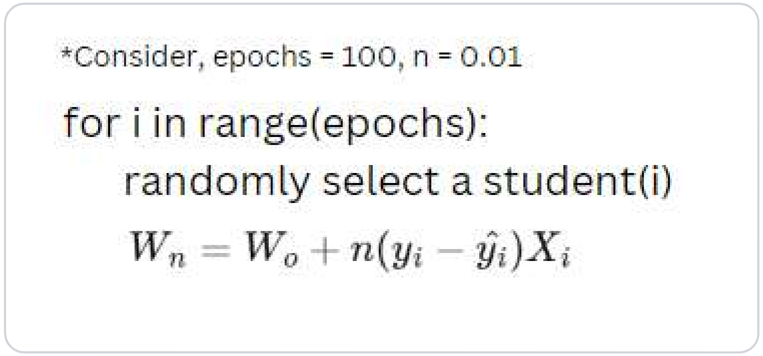
$$W_n = W_o + n(y_i - \hat{y}_i)X_i$$
$$W_n = W_o + n(0 - 1)X_i$$
$$W_n = W_o - nX_i$$

Similarly, for case 3, we should add and according to formula if we put values,

$$W_n = W_o + n(y_i - \hat{y}_i)X_i$$
$$W_n = W_o + n(1 - 0)X_i$$
$$W_n = W_o + nX_i$$

Thus, we have achieved it in a single equation instead of if condition block.

Now the algorithm becomes,



```
*Consider, epochs = 100, n = 0.01
for i in range(epochs):
    randomly select a student(i)
    $W_n = W_o + n(y_i - \hat{y}_i)X_i$
```

Code Implementation:

```
def perceptron(X,y):

    X = np.insert(X,0,1,axis=1)
    weights = np.ones(X.shape[1])
    lr = 0.1

    for i in range(1000):
        j = np.random.randint(0,100)
        y_hat = step(np.dot(X[j],weights))
        weights = weights + lr*(y[j]-y_hat)*X[j]

    return weights[0],weights[1:]

def step(z):
    return 1 if z>0 else 0

intercept_,coef_ = perceptron(X,y)
print(coef_)
print(intercept_)
```

The given Kaggle notebook contains the implementation of the concept, plus I have used it on the data to show results of the algorithm. Run all the cells and you will also see transformation of the line through the epochs.

Kaggle notebook: **https://www.kaggle.com/code/devyanichavan/perceptron-trick**

Final Words

Thanks for making it to the end. I hope you understood the concept of the perceptron trick. If you have any suggestions or corrections, please let me know in the comments.