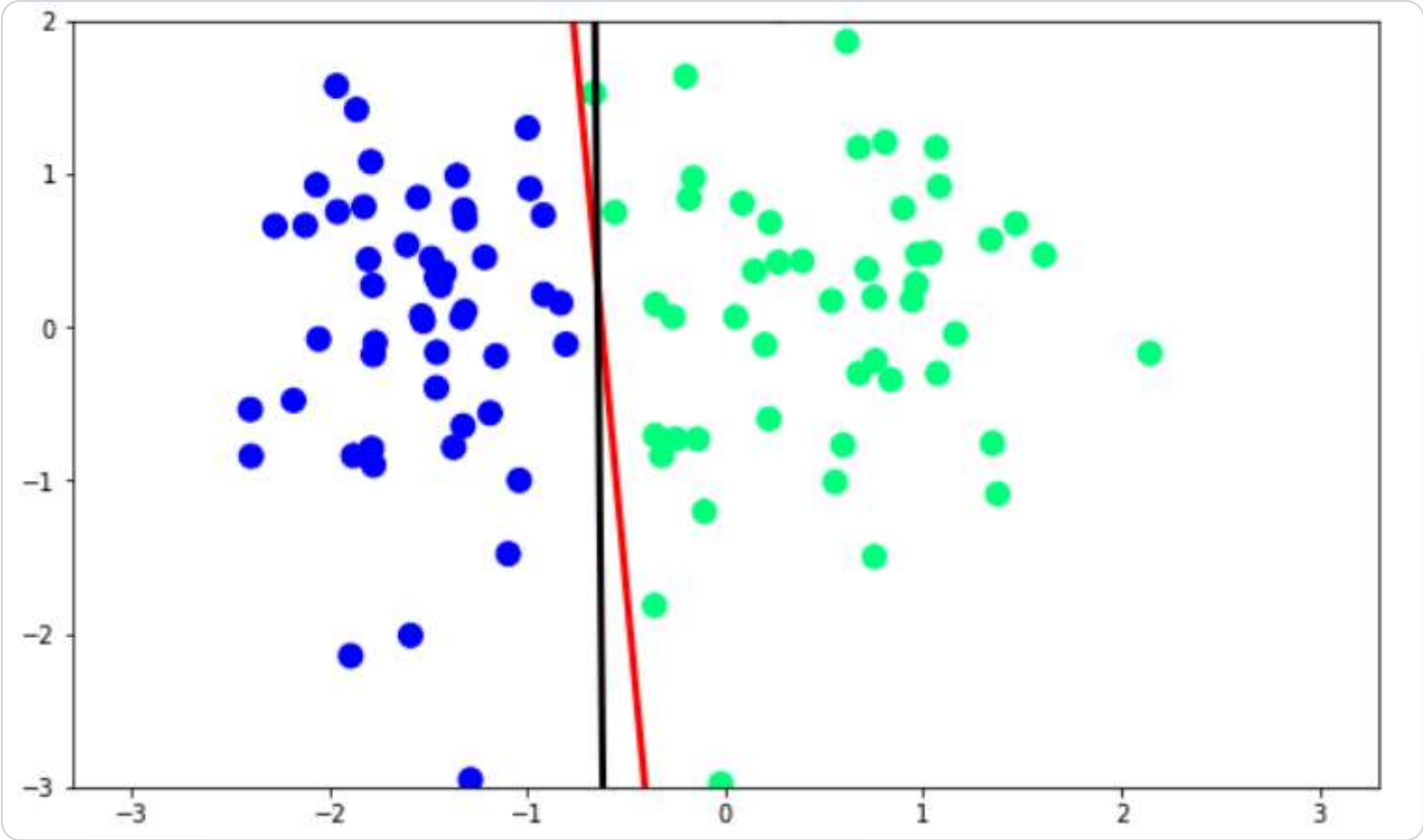


The reason for such a difference in the line is because we only considered the misclassified points and updated the equation. We didn't take into account the correctly classified points when writing the algorithm. If you remember, we mentioned in previous article that if the point is correctly classified then do nothing, otherwise update the equation.

Solution

Since, we know that the misclassified points pull the line towards them, we can make the correctly classified points to do opposite of it.

Therefore, a possible solution will be to make a correctly classified point push the line farther from it. Now, we must decide the magnitude of this pull and push.

Let's say, we have a point that is far from the line. This point will pull the line with a higher magnitude than the point which is near to the line. Thus, this is the case for misclassified point.

When we consider correctly classified point, we must do opposite of it. That is, the point which is near to the line will push it with higher magnitude than the point which is far from the line.

Modifying the equation

The equation that was the brain of the previous algorithm is given below:

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

In order to consider the correctly classified point, we must make sure that difference between the actual value and predicted value never becomes 0. Because, when it becomes 0 it keeps the weight same and doesn't update the old equation.

If you remember, our W is actually defined as follows,

$$W_0 + W_1X_1 + W_2X_2 = 0$$

According to previous algorithm, we just substituted the value of X1 and X2, multiply with weights and then add them to get a value. If this value is greater than 0, we predict the result as 1, however if value is less than equal to 0 then result is 0. That is, we are using step function in this case.

However, since we do not want 0 as result ever because it will mean that our update equation will not change if the point is correctly classified.

Hence, to solve this, we use sigmoid function instead of step function.

Sigmoid Function

The sigmoid function is such a function that it squeezes any value between the range of 0 to 1. It means if sigmoid function is applied to any value ranging from negative infinity to positive infinity, then we will get the answer in range 0 to 1.

The sigmoid function is given by,

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Now, consider the following cases:

- If point lies on line

$$\sum_{i=1}^{2} W_i X_i = 0$$

and thus, sigmoid of 0 is 0.5

$$\begin{aligned} \sigma(0) &= \frac{1}{1 + e^{0}} \\ &= \frac{1}{2} \\ &= 0.5 \end{aligned}$$

- If point is on positive side of line

$$\sum_{i=1}^{2} W_i X_i > 0$$

and thus, sigmoid of this value is greater than 0.5

$$\sigma > 0.5$$

- If point is on negative side of line

$$\sum_{i=1}^{2} W_i X_i < 0$$

and thus, sigmoid of this value is less than 0.5

$$\sigma < 0.5$$

The value of the sigmod function determines the probability of the class being positive or true. Or if we consider example given in previous article, it is probability that loan is approved.

Impact of Sigmoid Function

So based on our discussion till now, we have,

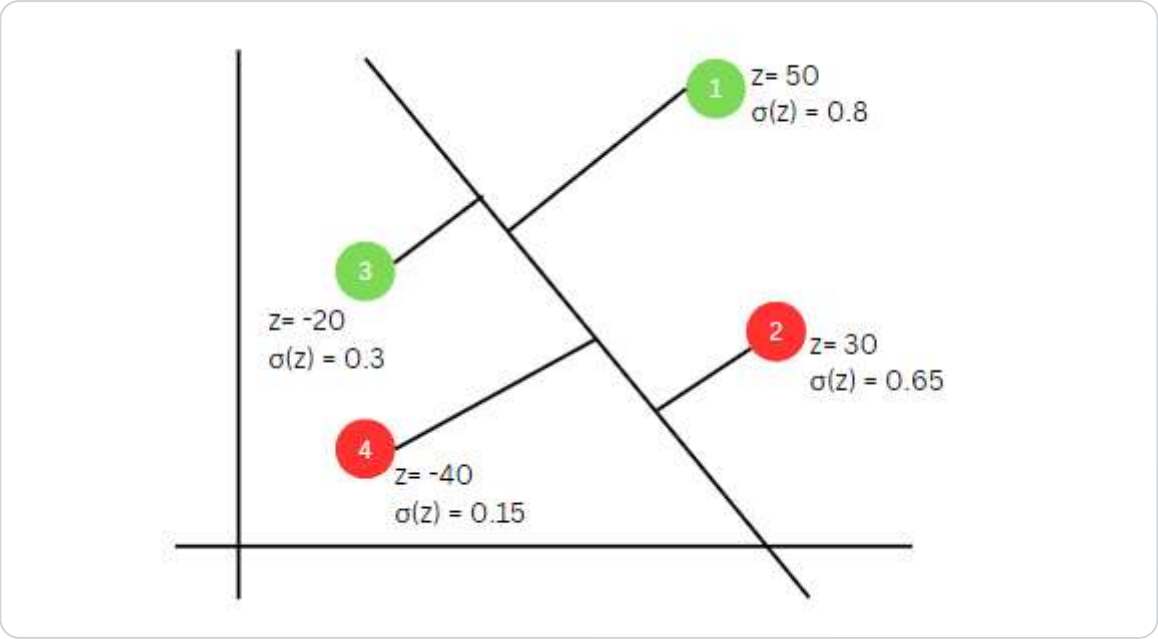
$$W_n = W_o + n(y_i - \hat{y}_i)X_i$$

and

$$\hat{y}_i = \sigma(z)$$

where,

$$z = \sum W_i X_i$$



When we consider the above figure we see that points 3 and 2 are misclassified. We calculate z for each and then apply the sigmoid function on z. Thus, we get

Data Point	**Actual value (y_i)**	**Predicted(hat{y_i})**	**Actual - Predicted(y_i - hat{y_i})**
1	1	0.8	0.2
2	0	0.65	-0.65
3	1	0.3	0.7
4	0	0.15	-0.15

As you can see, data point 1 has a positive value of difference between actual and predicted value. When we were using step function, we used to get zero for correctly classified points and hence no change in line. But since, we got 0.2 value, the line will move away from(push) the point.

$$W_n = W_o + n * 0.2 * X_i$$

Data point 2 is misclassified, thus it will pull the line towards itself with greater magnitude, it is given by,

$$W_n = W_o - n * 0.65 * X_i$$

Similarly, misclassified point 3, will pull the line downwards towards itself by,

$$W_n = W_o + n * 0.7 * X_i$$

And you guessed it right, point 4 will push the line as it is correctly classified.

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 = sigmoid(np.dot(X[j],weights))
        weights = weights + lr*(y[j]-y_hat)*X[j]

    return weights[0],weights[1:]
def sigmoid(z):
    return 1/(1 + np.exp(-z))

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

```
m = -(coef_[0]/coef_[1])
b = -(intercept_/coef_[1])


x_input = np.linspace(-3,3,100)
y_input = m*x_input + b


plt.figure(figsize=(10,6))
plt.plot(x_input,y_input,color='red',linewidth=3)
plt.scatter(X[:,0],X[:,1],c=y,cmap='winter',s=100)
```


More from this blog

Logistic Regression / Perceptron Trick

We have studied different types of linear regression algorithms, now we will understand logistic regression. It is one of the most important Machine Learning algorithms especially if you are going to study Dee...

Jun 9, 2024 6 min read 666