# NNFS LAB 2: Perceptron

## Perceptron Algorithm

Group 1:

Parth Deshpande 191060022

Ishan Deshpande 191060021

Dev Sharma 191060023

Kedar Daithankar 191060019

---

The **Perceptron algorithm** is a two-class (binary) classification machine learning algorithm. It is a type of neural network model, perhaps the simplest type of neural network model.

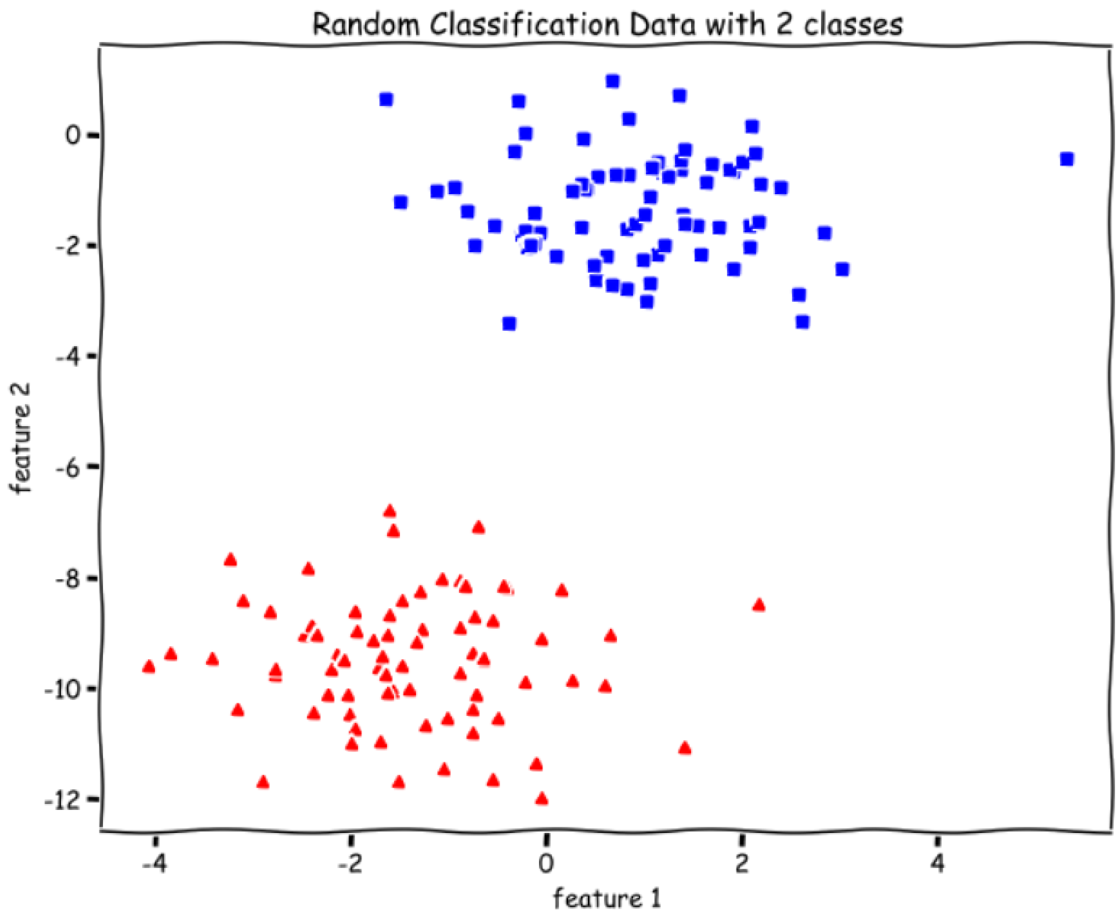
The perceptron is the building block of artificial neural networks, it is a simplified model of the biological neurons in our brain. A perceptron is the simplest neural network, one that is comprised of just one neuron. The perceptron algorithm was invented in 1958 by Frank Rosenblatt.

It consists of a single node or neuron that takes a row of data as input and predicts a class label. This is achieved by calculating the weighted sum of the inputs and a bias (set to 1). The weighted sum of the input of the model is called the activation.

- **Activation** = Weights * Inputs + Bias

If the activation is above 0.0, the model will output 1.0; otherwise, it will output 0.0.

- **Predict 1**: If Activation > 0.0
- **Predict 0**: If Activation <= 0.0



There are two classes, red and green, and we want to separate them by drawing a straight line between them. Or, more formally, we want to learn a set of parameters **theta** to find an optimal hyperplane (straight line for our data) that separates the two classes.

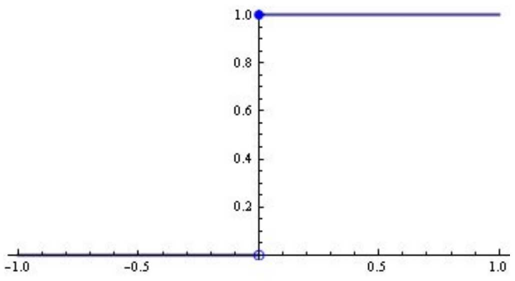
**theta is also known as weights vector.**

For the Perceptron algorithm, we apply a different function over theta.X , which is the Unit Step Function, which is defined as

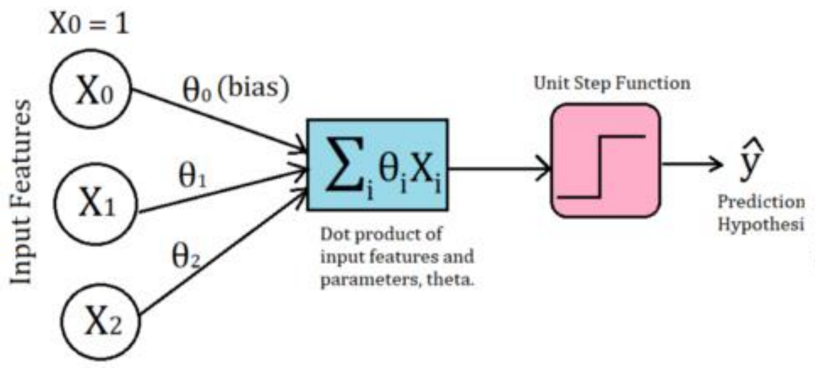
$$h_\theta(x) = g(\theta^T x)$$

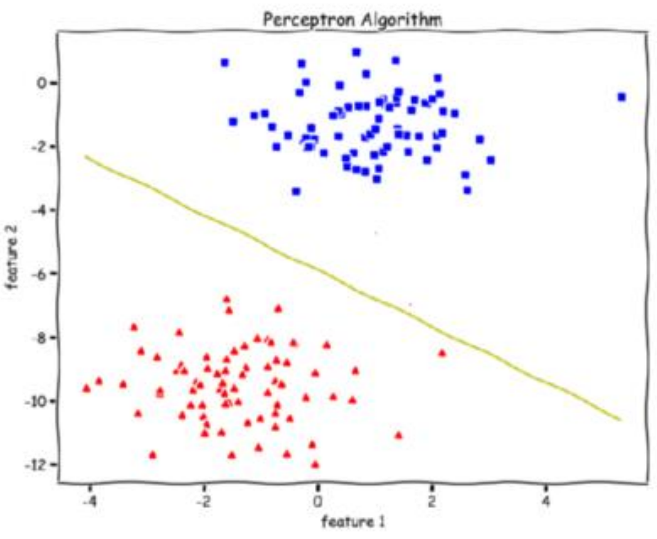
Where,

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



## Perceptron as a Neural Net:





We can visually understand the Perceptron by looking at the above image. For every training example, we first take the dot product of input features and parameters, theta. Then, we apply the Unit Step Function to make the prediction(y_hat).

And if the prediction is wrong or in other words the model has misclassified that example, we make the update for the parameters theta. We don’t update when the prediction is correct (or the same as the true/target value y).

## Perceptron Update Rule:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}.$$

**newθ = oldθ + learning_rate *(true_Y – predicted_Y)*training_x**

**θ is theta**

Implementation:

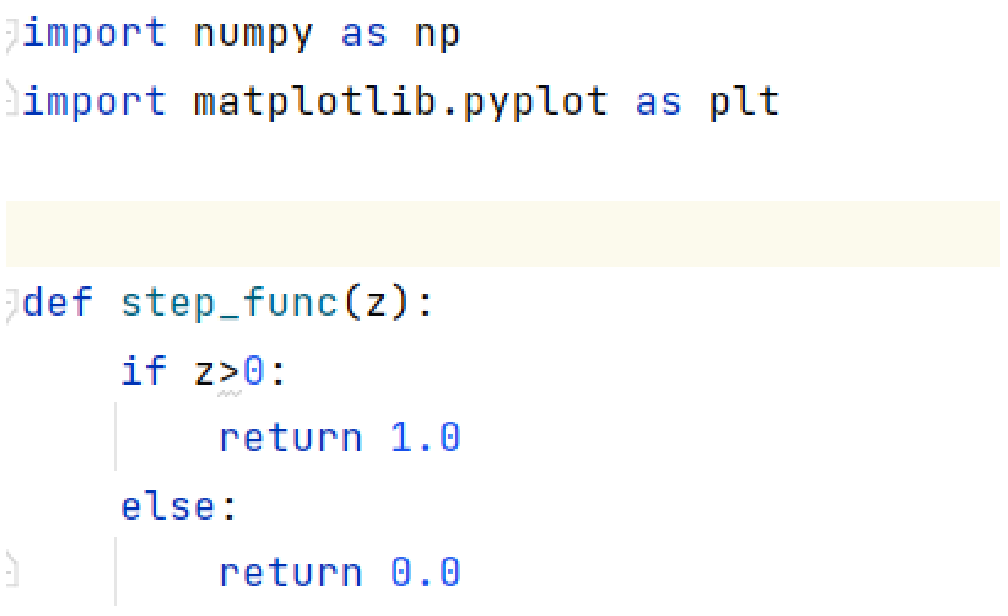
https://github.com/heisenberg-88/nnfs_lab2_Perceptron-Algorithm

We have used 4 types of datasets for testing this perceptron algorithm:

1) Linearly separable with less standard deviation.
2) Linearly separable with slightly more standard deviation.
3) Csv dataset we got from the internet.
4) Non-linearly separable dataset from csv.

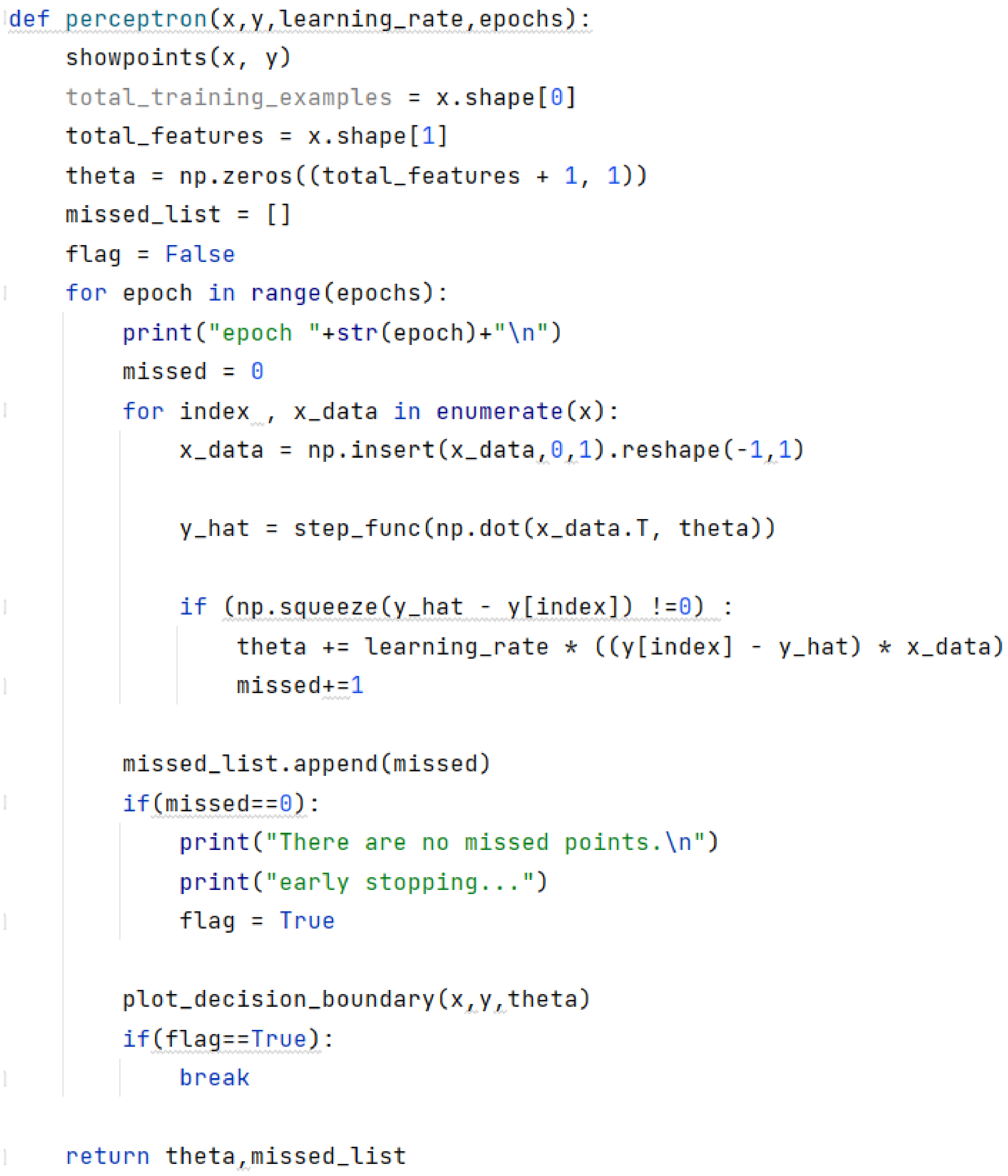
## Perceptron.py

Here we've implemented step function , main perceptron algorithm , plotting the decision boundary.



```python
import numpy as np
import matplotlib.pyplot as plt


def step_func(z):
    if z>0:
        return 1.0
    else:
        return 0.0
```

step function returns 1.0 when input is greater than 0 and returns 0.0 otherwise.



```python
def perceptron(x,y,learning_rate,epochs):
    showpoints(x, y)
    total_training_examples = x.shape[0]
    total_features = x.shape[1]
    theta = np.zeros((total_features + 1, 1))
    missed_list = []
    flag = False
    for epoch in range(epochs):
        print("epoch "+str(epoch)+"\n")
        missed = 0
        for index , x_data in enumerate(x):
            x_data = np.insert(x_data,0,1).reshape(-1,1)

            y_hat = step_func(np.dot(x_data.T, theta))

            if (np.squeeze(y_hat - y[index]) !=0) :
                theta += learning_rate * ((y[index] - y_hat) * x_data)
                missed+=1

        missed_list.append(missed)
        if(missed==0):
            print("There are no missed points.\n")
            print("early stopping...")
            flag = True

        plot_decision_boundary(x,y,theta)
        if(flag==True):
            break

    return theta,missed_list
```

As the shape of input X is (number of samples **x** number of features)

So , **total_training_examples** and **total_features** stores this.

**missed_list[]** keeps track of how many points are misclassified.

After this,

Perceptron weight updation runs for all samples and for all epochs until it's early stopped.

In the **x_data** extra '1' is inserted. Because, in rosanblatt's perceptron Bias is included in input itself.

```
x_data = np.insert(x_data,0,1).reshape(-1,1)
```

Now,
**y_hat** is calculated which is the predicted value using step function.
And then if difference between predicted and true is not 0 then, the updating formula is used.
At the same time, we've also missed a point in the separation line so we add to the **missed** variable.

```
y_hat = step_func(np.dot(x_data.T, theta))

if (np.squeeze(y_hat - y[index]) !=0) :
    theta += learning_rate * ((y[index] - y_hat) * 
x_data)
    missed+=1
```

***EARLY STOPPING*:**

We've also added early stopping feature in this. So that whenever there are no missed points, that means the decision boundary is perfect and there's no need to iterate for further epochs.

And after 0 missed points the algorithm stops ad shows separation line.

```
if(missed==0):
    print("There are no missed points.\n")
    print("early stopping...")
    flag = True


...

if(flag==True):
    break
```

here it checks if the missed are 0 then, it turns the **flag** to **TRUE**.

The epochs for-loop checks the flag variable every time.

So, if the flag is TRUE. Then the for-loop **breaks**.

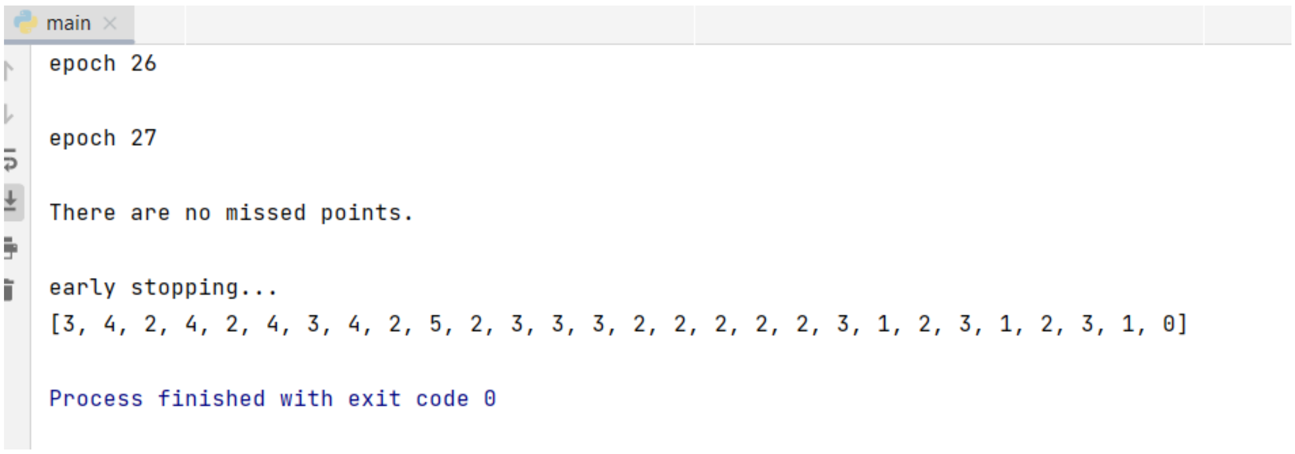
This is how the algorithm stops in 20 epochs and sometimes 5 epochs according to the variance of the input data.

---

perceptron function returns theta and missed_list

missed_list can be used to check how many epochs are used for finding the correct decision boundary.

Also, it can be used to find number of missed points every time.



```
epoch 26

epoch 27

There are no missed points.

early stopping...
[3, 4, 2, 4, 2, 4, 3, 4, 2, 5, 2, 3, 3, 3, 2, 2, 2, 2, 2, 3, 1, 2, 3, 1, 2, 3, 1, 0]

Process finished with exit code 0
```

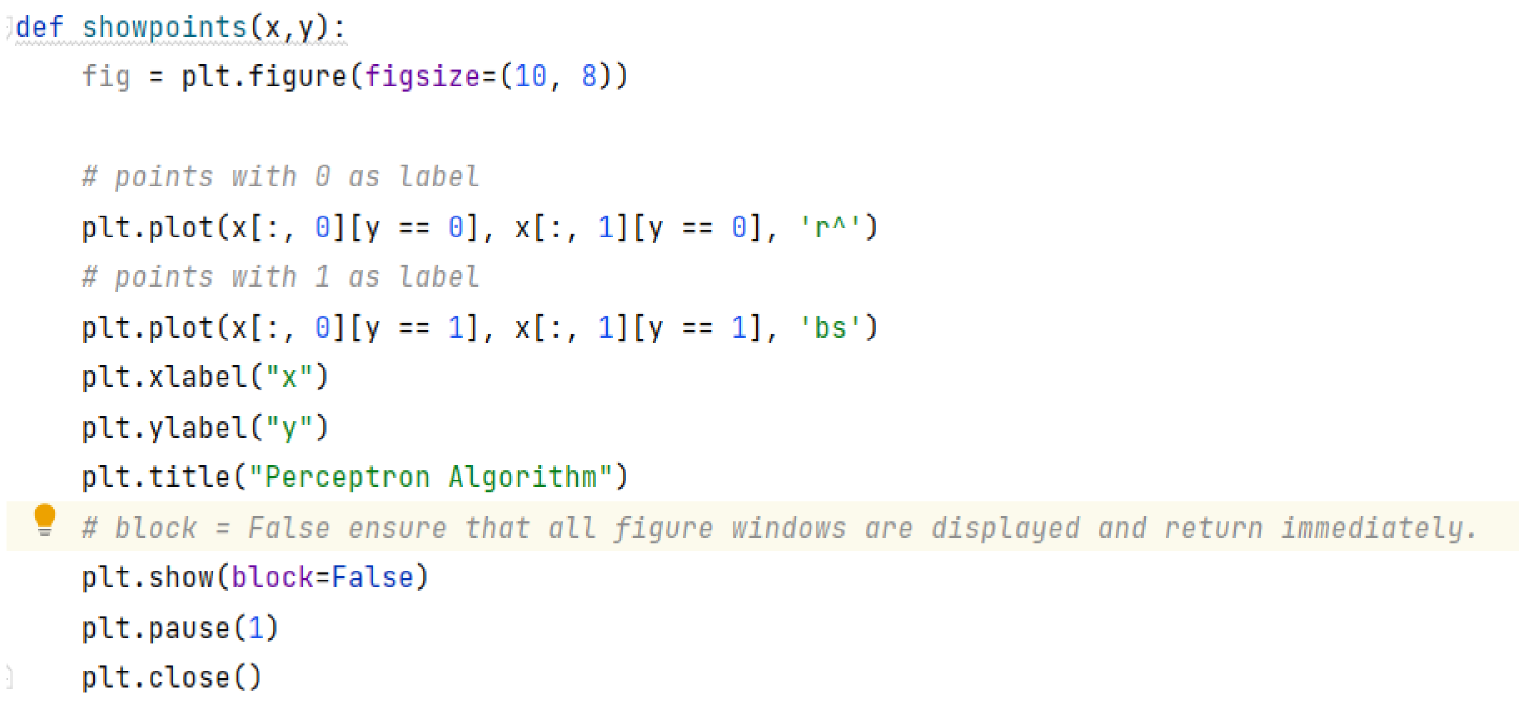
**0** in the end of missed_list shows no more missed points were left so, early stopped!

We know that the model makes a prediction of —

y=1 when y_hat ≥ 0

y=0 when y_hat < 0

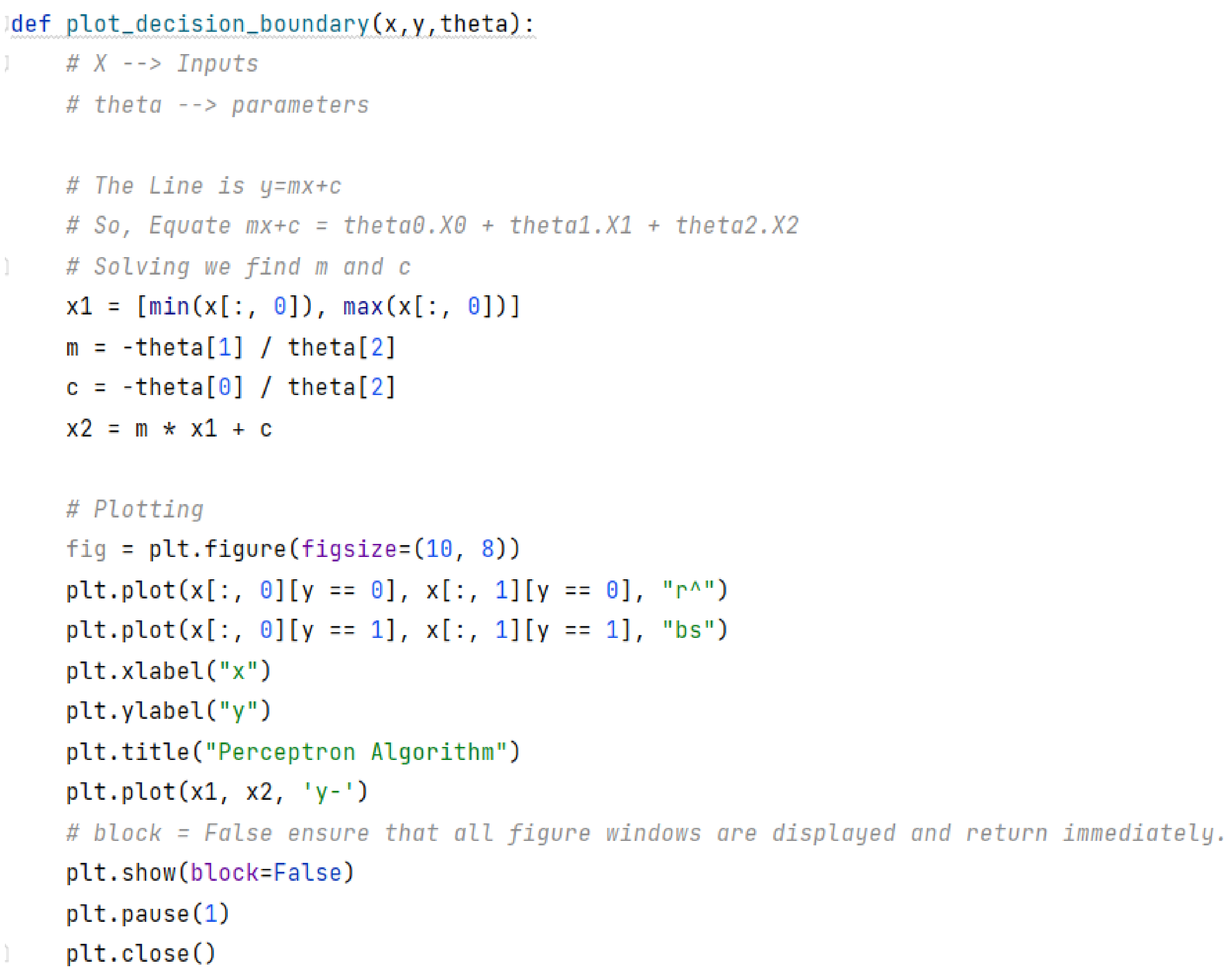
So, theta.X = 0 is going to be our Decision boundary.



```python
def showpoints(x,y):
    fig = plt.figure(figsize=(10, 8))

    # points with 0 as label
    plt.plot(x[:, 0][y == 0], x[:, 1][y == 0], 'r^')
    # points with 1 as label
    plt.plot(x[:, 0][y == 1], x[:, 1][y == 1], 'bs')
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Perceptron Algorithm")
    # block = False ensure that all figure windows are displayed and return immediately.
    plt.show(block=False)
    plt.pause(1)
    plt.close()
```

showpoints() is called in the starting of perceptron function. This is used for showing the input data points. Different colour and shape for two different classes.

After plt.show() , there's plt.pause(1) which pauses the plot for 1 sec and closes so that next graph after each epoch can be shown.



```python
def plot_decision_boundary(x,y,theta):
    # X --> Inputs
    # theta --> parameters

    # The Line is y=mx+c
    # So, Equate mx+c = theta0.X0 + theta1.X1 + theta2.X2
    # Solving we find m and c
    x1 = [min(x[:, 0]), max(x[:, 0])]
    m = -theta[1] / theta[2]
    c = -theta[0] / theta[2]
    x2 = m * x1 + c

    # Plotting
    fig = plt.figure(figsize=(10, 8))
    plt.plot(x[:, 0][y == 0], x[:, 1][y == 0], "r^")
    plt.plot(x[:, 0][y == 1], x[:, 1][y == 1], "bs")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Perceptron Algorithm")
    plt.plot(x1, x2, 'y-')
    # block = False ensure that all figure windows are displayed and return immediately.
    plt.show(block=False)
    plt.pause(1)
    plt.close()
```

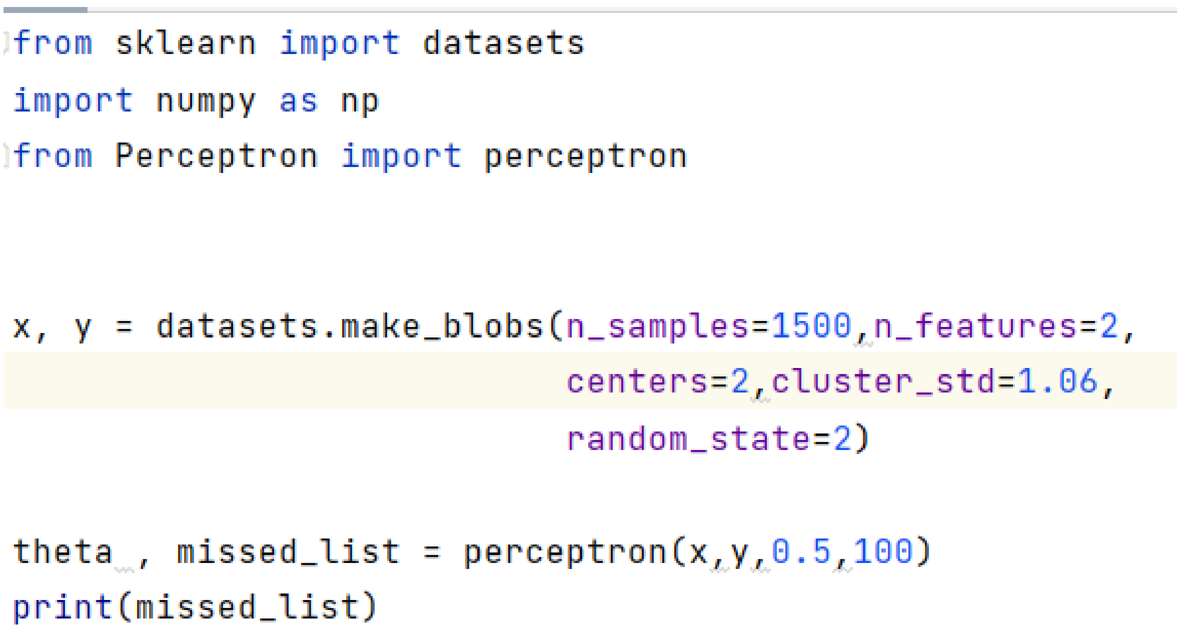
This plots the decision boundary and this function is called every time we update the theta in every epoch.

After plt.show() , there's  plt.pause(1)  which pauses the plot for 1 sec and closes so that next graph after each epoch can be shown.

**main.py**

**As mention in the start we have used 4 types of input data.**

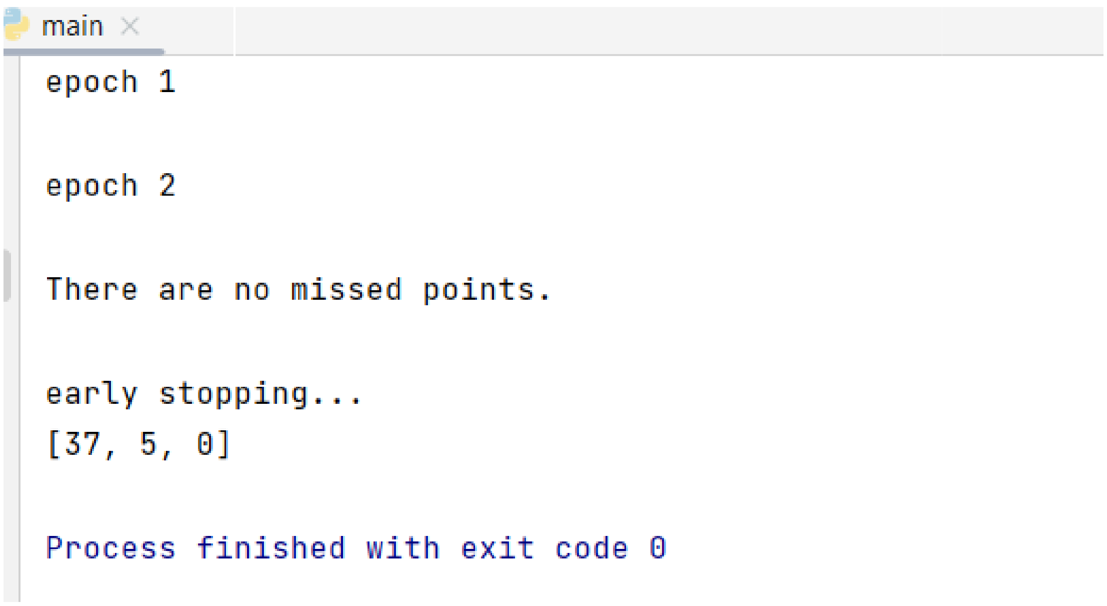
1) Linearly separable with less standard deviation.



```python
from sklearn import datasets
import numpy as np
from Perceptron import perceptron


x, y = datasets.make_blobs(n_samples=1500,n_features=2,
                           centers=2,cluster_std=1.06,
                           random_state=2)

theta , missed_list = perceptron(x,y,0.5,100)
print(missed_list)
```

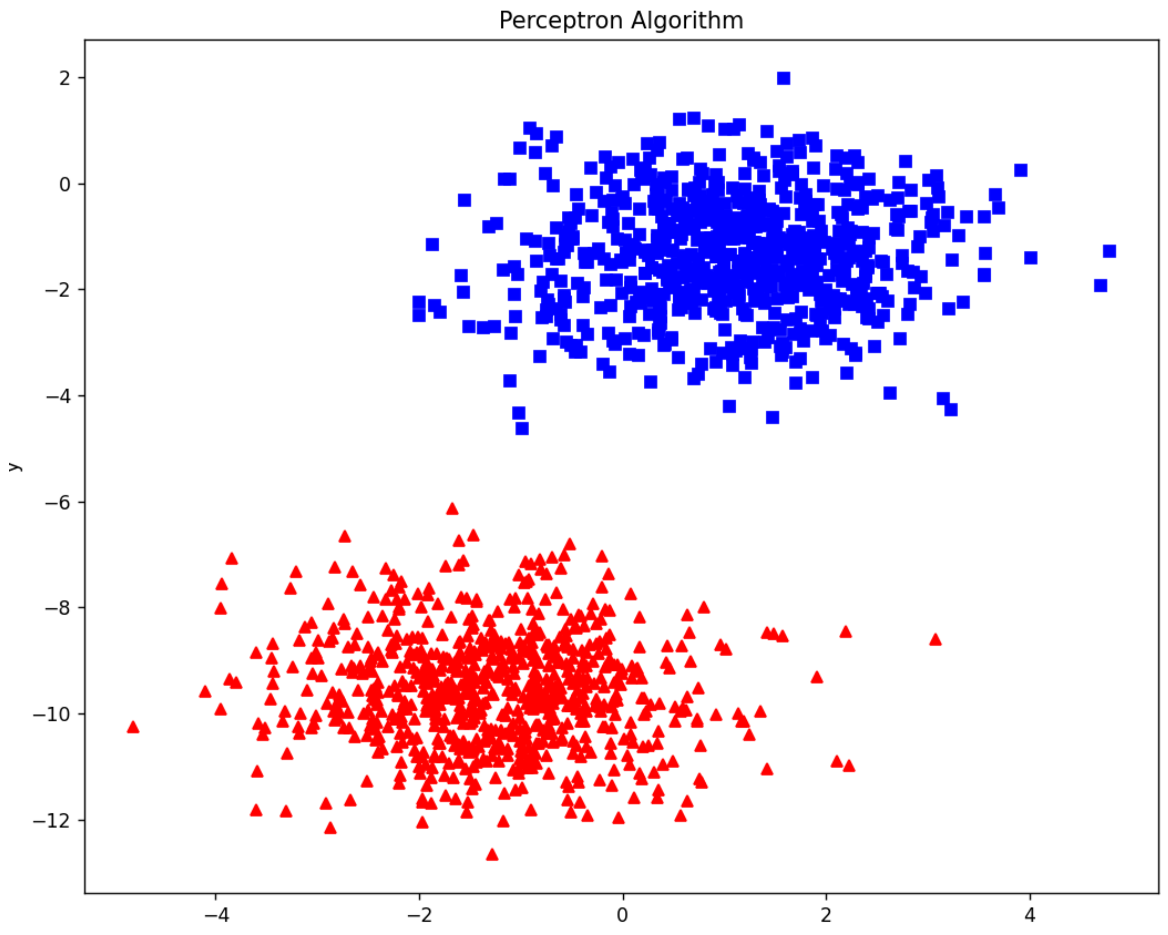
First, we have imported the perceptron function from perceptron.py created before.
And for the input data we have used sklearn.datasets.
In the arguments, we have passed **standard deviation** as **1.06**

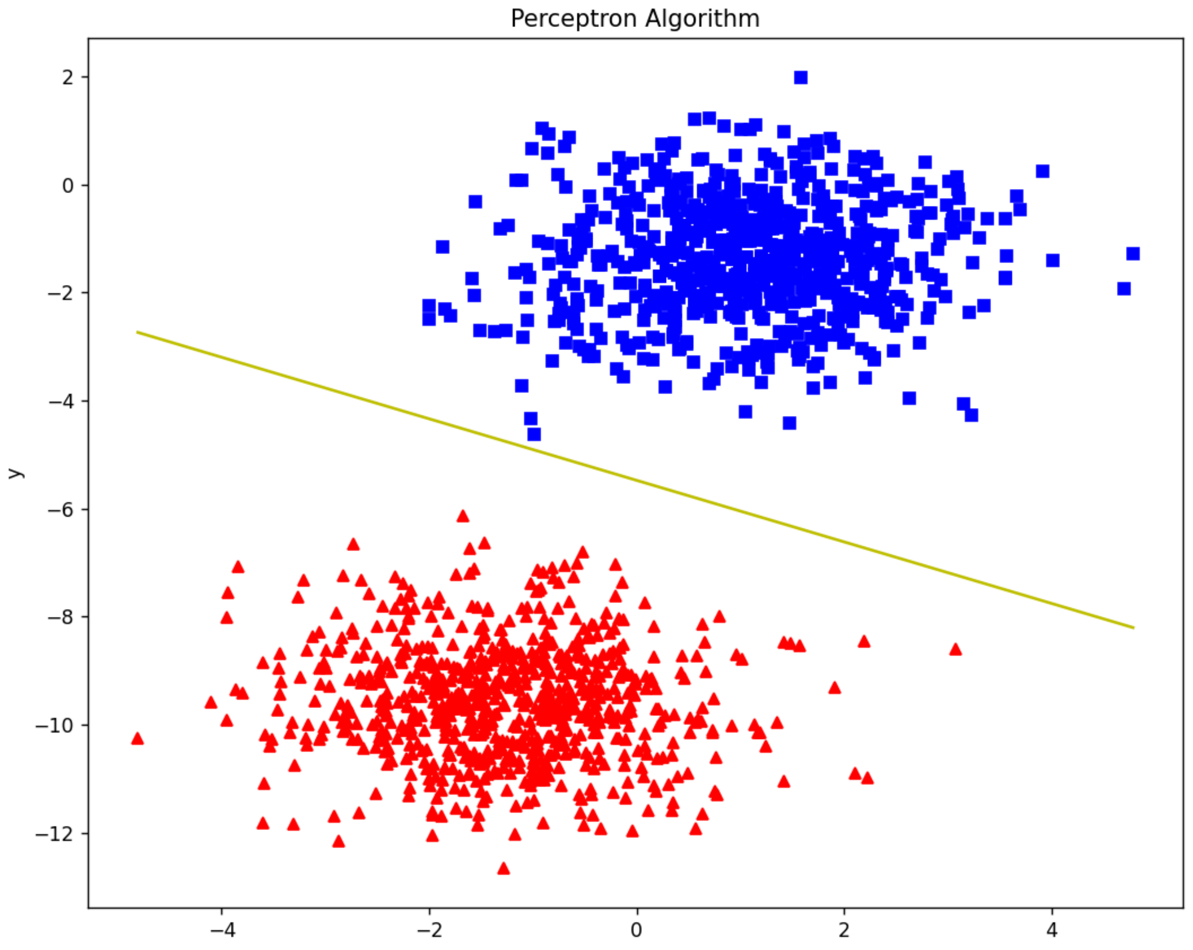
Learning rate is 0.5 and number of epochs are 100

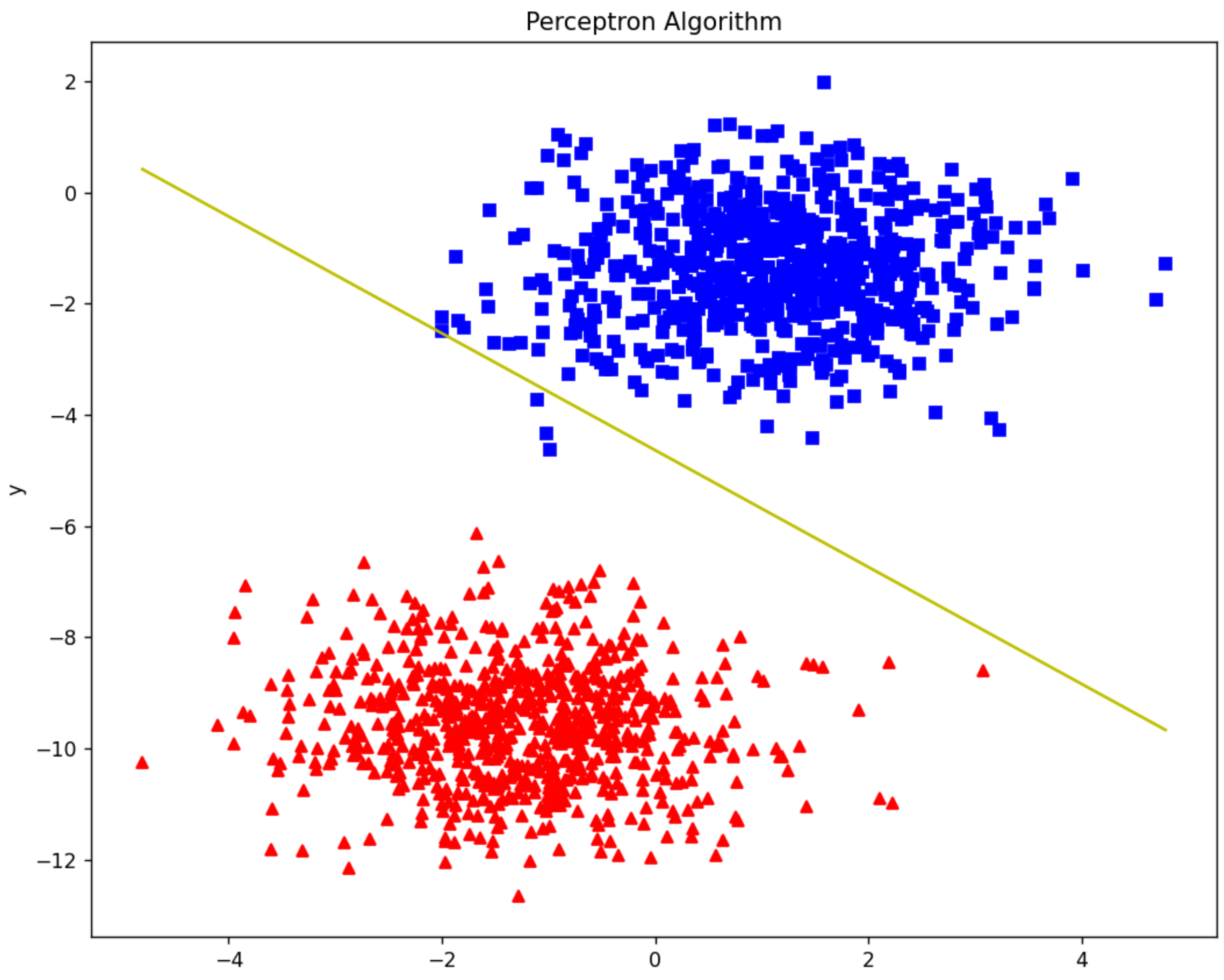


main ×

```
epoch 1

epoch 2

There are no missed points.

early stopping...
[37, 5, 0]

Process finished with exit code 0
```


Perceptron Algorithm
y
2
0
−2
−4
−6
−8
−10
−12
−4
−2
0
2
4

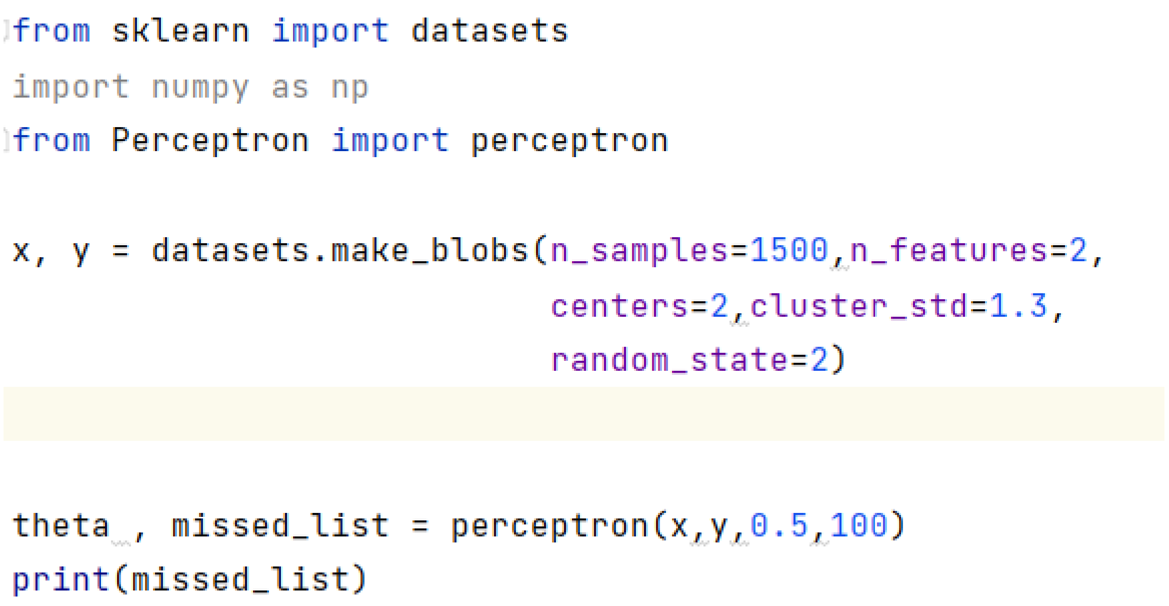

Perceptron Algorithm
y
2
0
−2
−4
−6
−8
−10
−12
−4
−2
0
2
4



This is the perfect decision boundary for the training input data with standard deviation = 1.06.

Epochs required after early stopping = 2

2) Linearly separable with slightly more standard deviation.

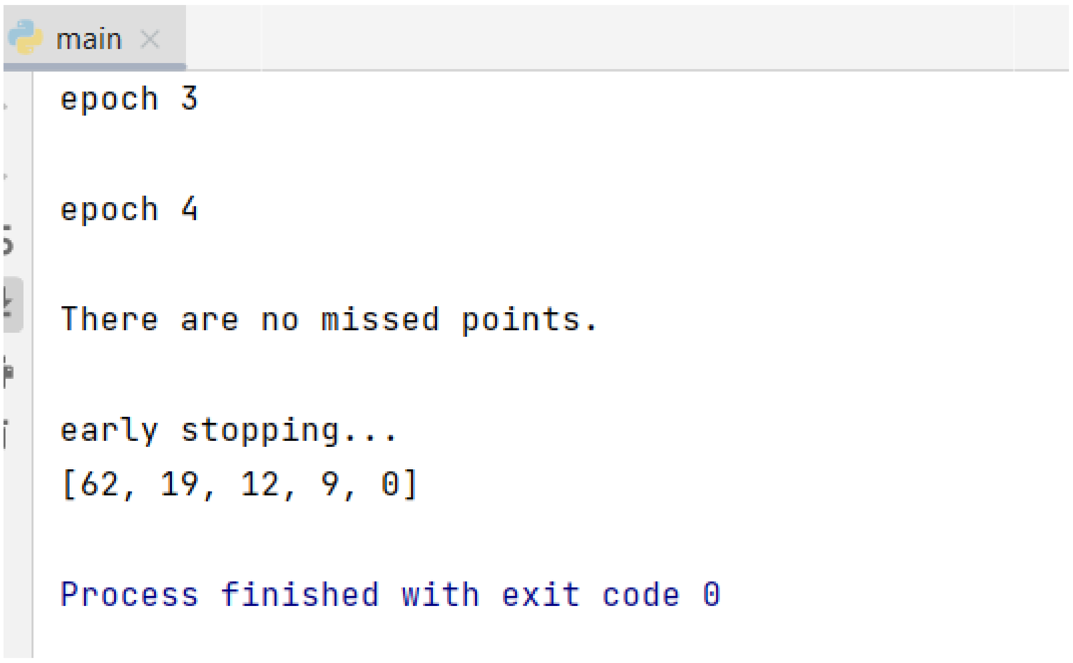


```
from sklearn import datasets
import numpy as np
from Perceptron import perceptron

x, y = datasets.make_blobs(n_samples=1500,n_features=2,
                           centers=2,cluster_std=1.3,
                           random_state=2)


theta , missed_list = perceptron(x,y,0.5,100)
print(missed_list)
```

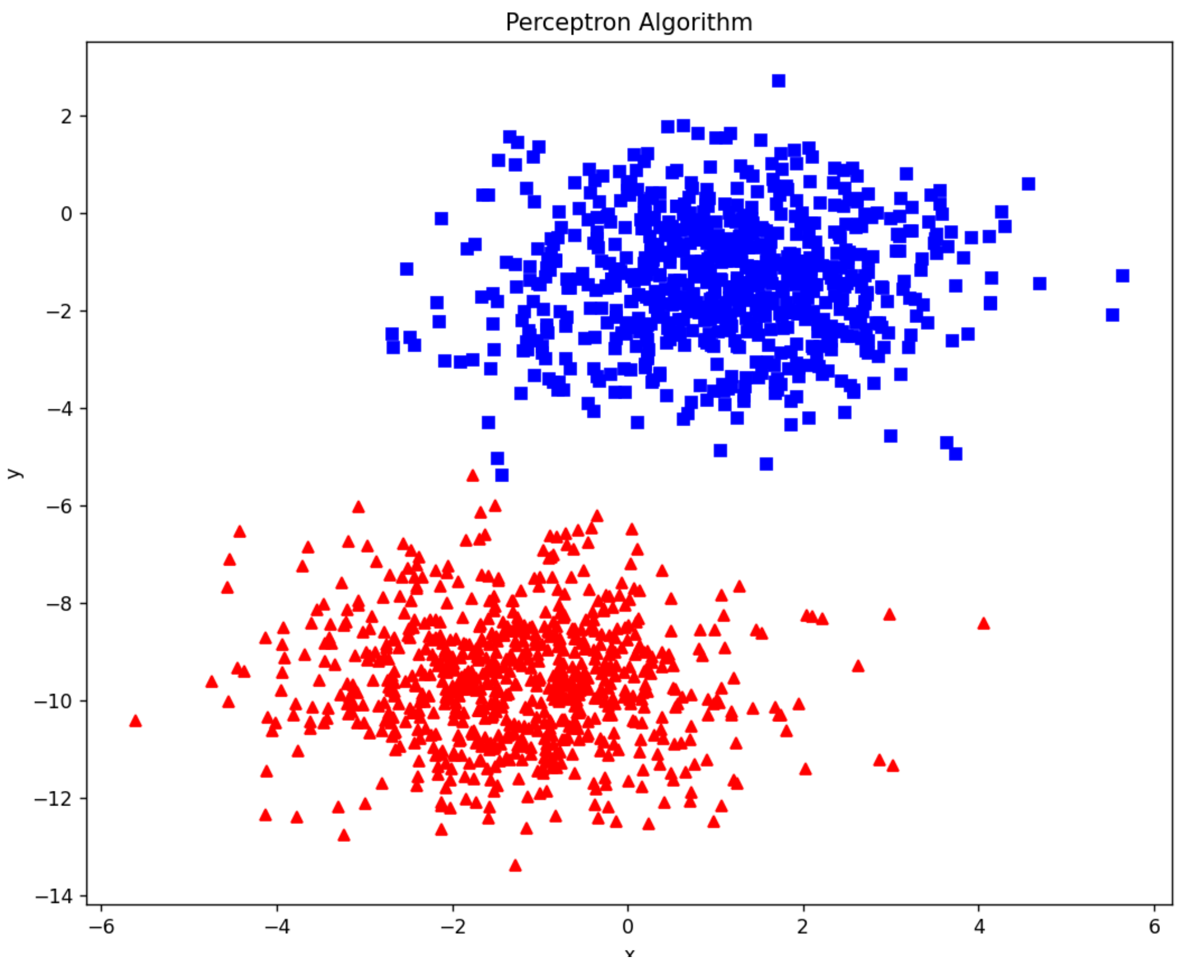
This is same as the previous dataset with some changes.

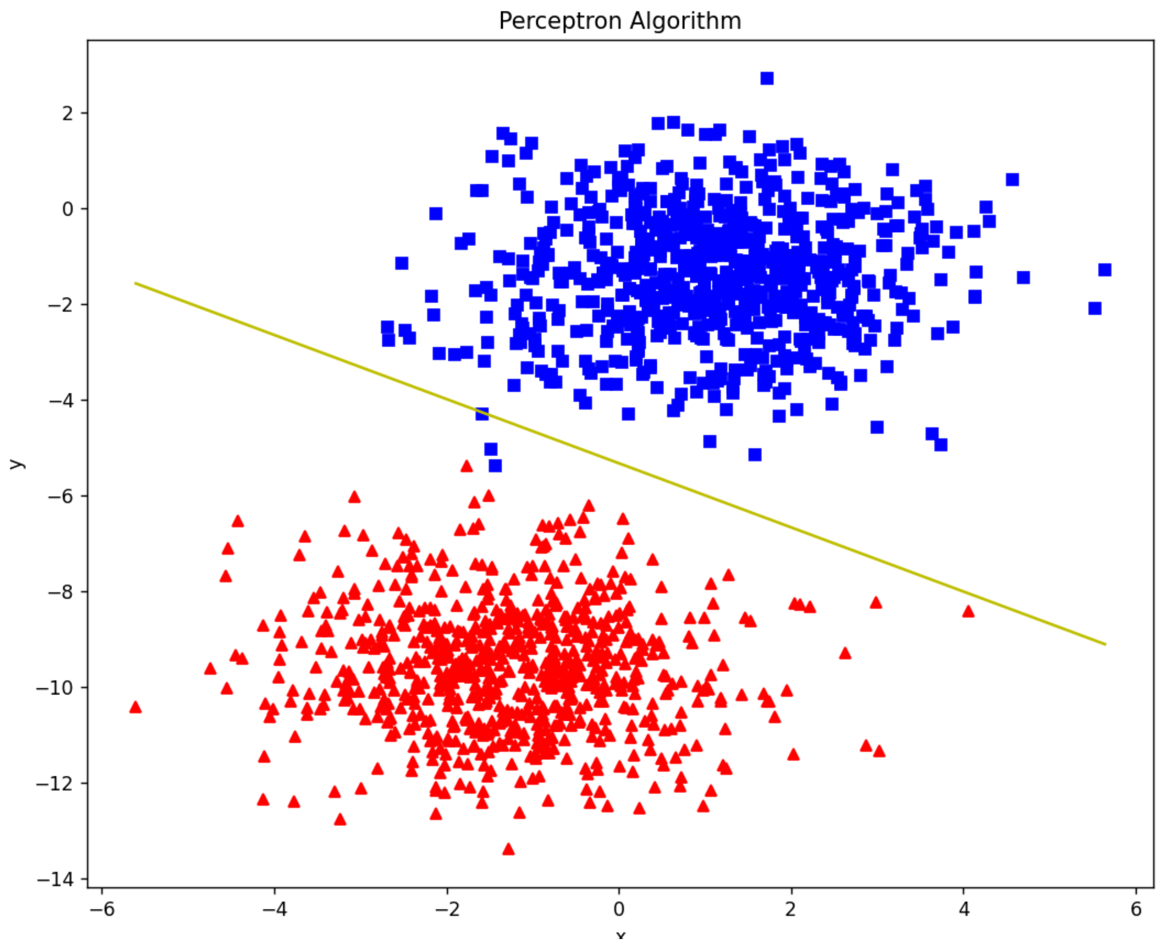
**standard deviation** as **1.3 (more than previous data)**

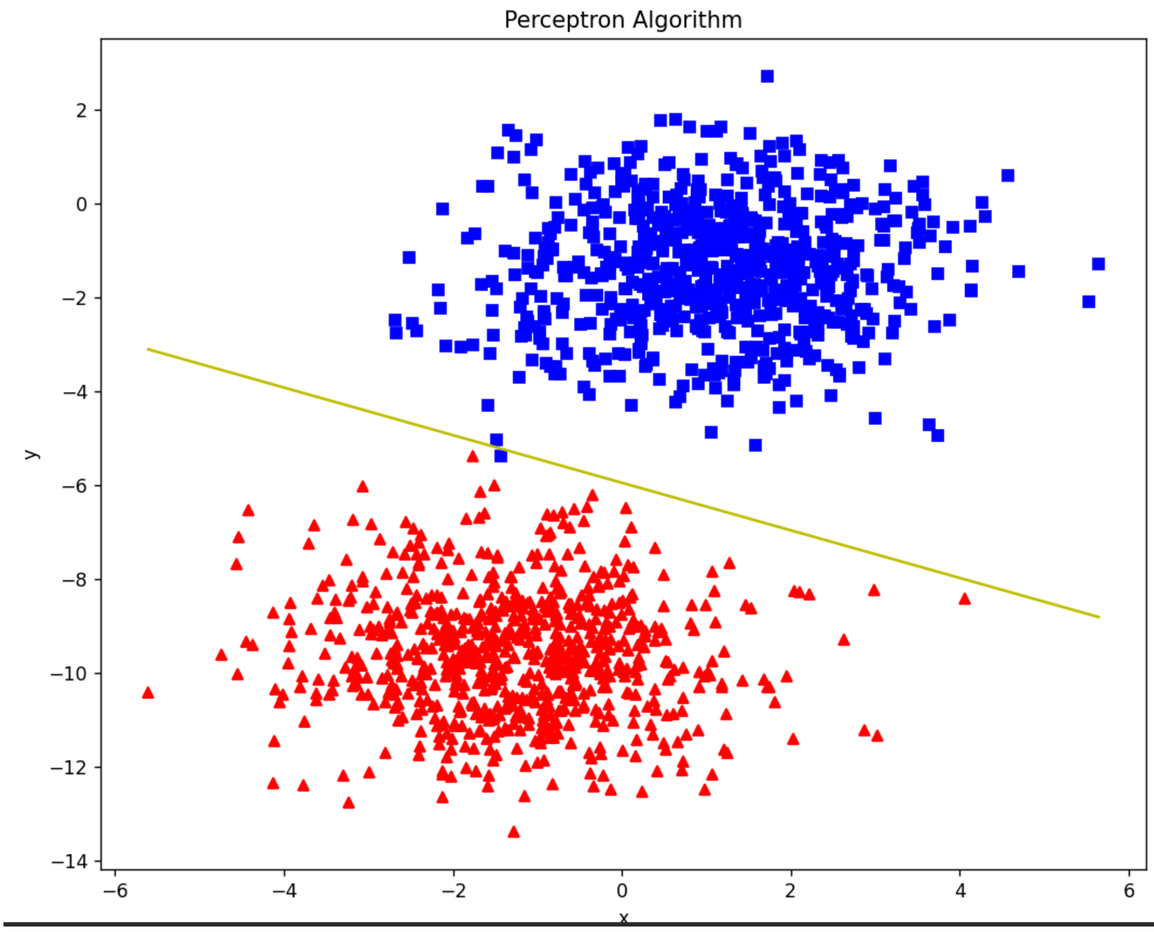
Learning rate is 0.5 and number of epochs are 100

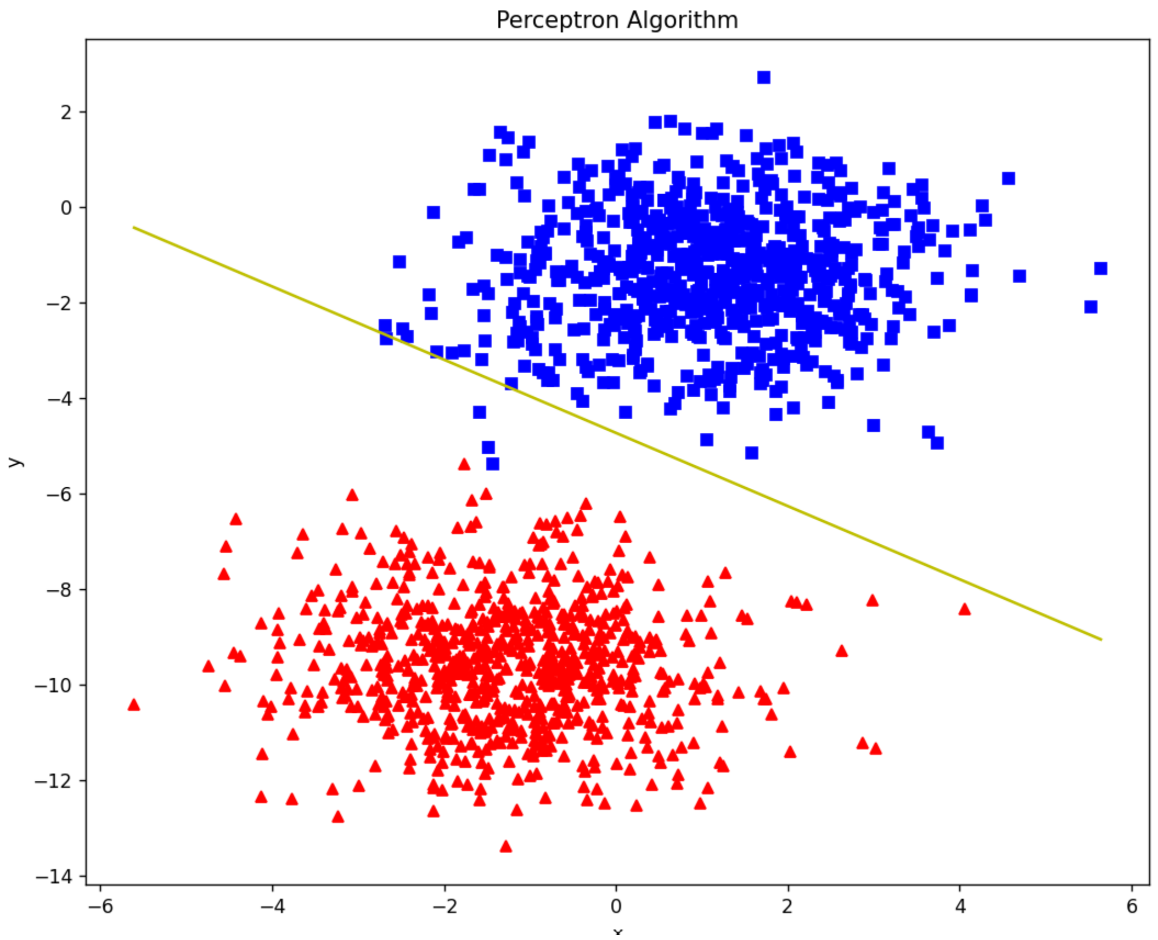


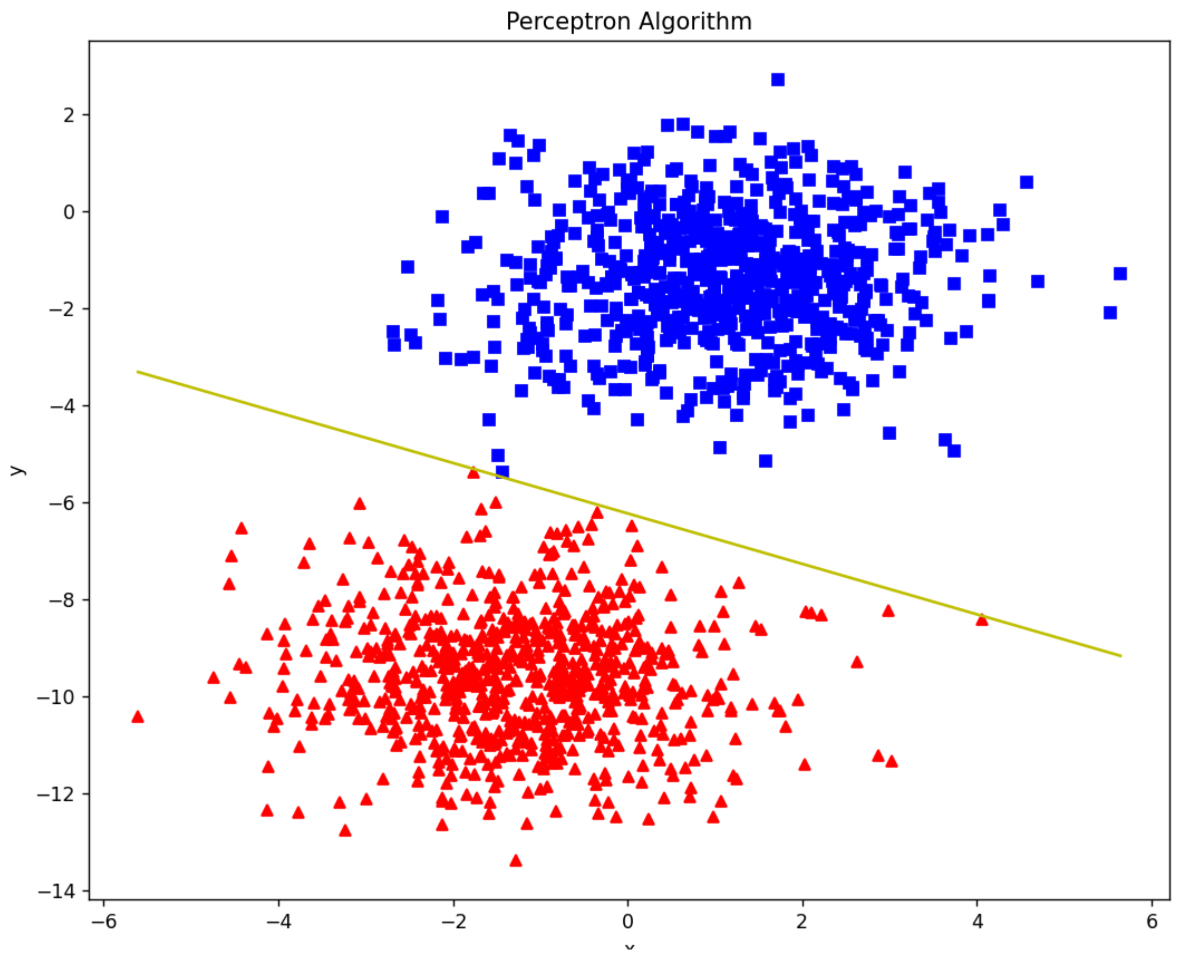
```
epoch 3

epoch 4

There are no missed points.

early stopping...
[62, 19, 12, 9, 0]

Process finished with exit code 0
```


Perceptron Algorithm
y
x
2
0
−2
−4
−6
−8
−10
−12
−14
−6
−4
−2
0
2
4
6


Perceptron Algorithm
y
x
2
0
−2
−4
−6
−8
−10
−12
−14
−6
−4
−2
0
2
4
6


Perceptron Algorithm
y
x
2
0
−2
−4
−6
−8
−10
−12
−14
−6
−4
−2
0
2
4
6


Perceptron Algorithm
y
x
2
0
−2
−4
−6
−8
−10
−12
−14
−6
−4
−2
0
2
4
6



This is the perfect decision boundary for the training input data with standard deviation = 1.3.

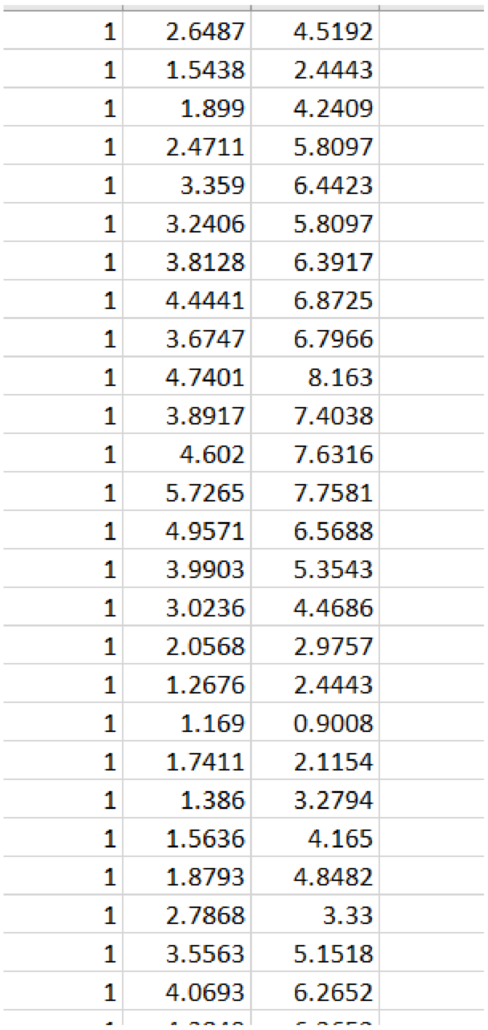
Epochs required after early stopping = 4

More epochs than previous example are required due to more standard deviation.

A standard deviation (or σ) is a measure of how dispersed the data is in relation to the mean.

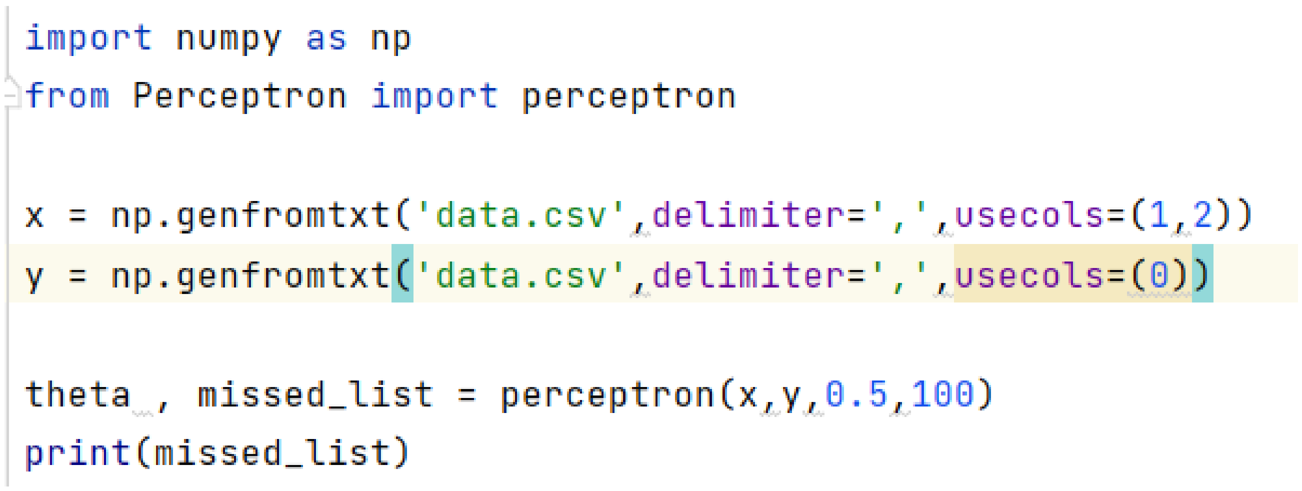
3) Csv dataset we got from the internet.

Dataset: https://github.com/cuekoo/Binary-classification-dataset



| | | |
|---|---|---|
| 1 | 2.6487 | 4.5192 |
| 1 | 1.5438 | 2.4443 |
| 1 | 1.899 | 4.2409 |
| 1 | 2.4711 | 5.8097 |
| 1 | 3.359 | 6.4423 |
| 1 | 3.2406 | 5.8097 |
| 1 | 3.8128 | 6.3917 |
| 1 | 4.4441 | 6.8725 |
| 1 | 3.6747 | 6.7966 |
| 1 | 4.7401 | 8.163 |
| 1 | 3.8917 | 7.4038 |
| 1 | 4.602 | 7.6316 |
| 1 | 5.7265 | 7.7581 |
| 1 | 4.9571 | 6.5688 |
| 1 | 3.9903 | 5.3543 |
| 1 | 3.0236 | 4.4686 |
| 1 | 2.0568 | 2.9757 |
| 1 | 1.2676 | 2.4443 |
| 1 | 1.169 | 0.9008 |
| 1 | 1.7411 | 2.1154 |
| 1 | 1.386 | 3.2794 |
| 1 | 1.5636 | 4.165 |
| 1 | 1.8793 | 4.8482 |
| 1 | 2.7868 | 3.33 |
| 1 | 3.5563 | 5.1518 |
| 1 | 4.0693 | 6.2652 |
| [illegible] | [illegible] | [illegible] |

Column 1: labels (1 or 0)
Column 2 & 3 are features.

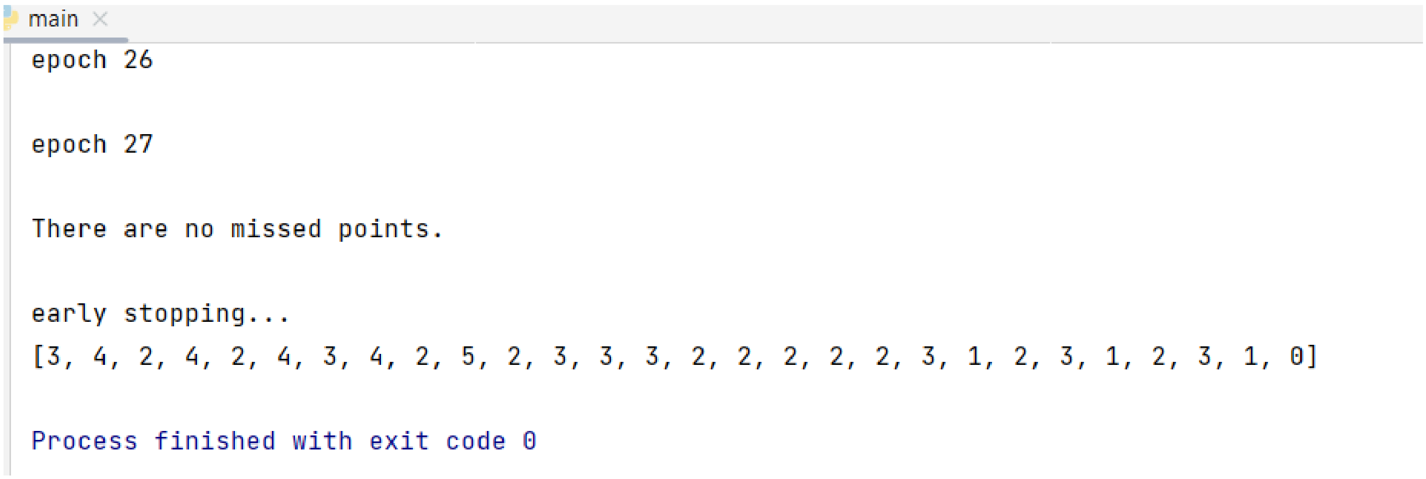


```
import numpy as np
from Perceptron import perceptron

x = np.genfromtxt('data.csv',delimiter=',',usecols=(1,2))
y = np.genfromtxt('data.csv',delimiter=',',usecols=(0))

theta , missed_list = perceptron(x,y,0.5,100)
print(missed_list)
```

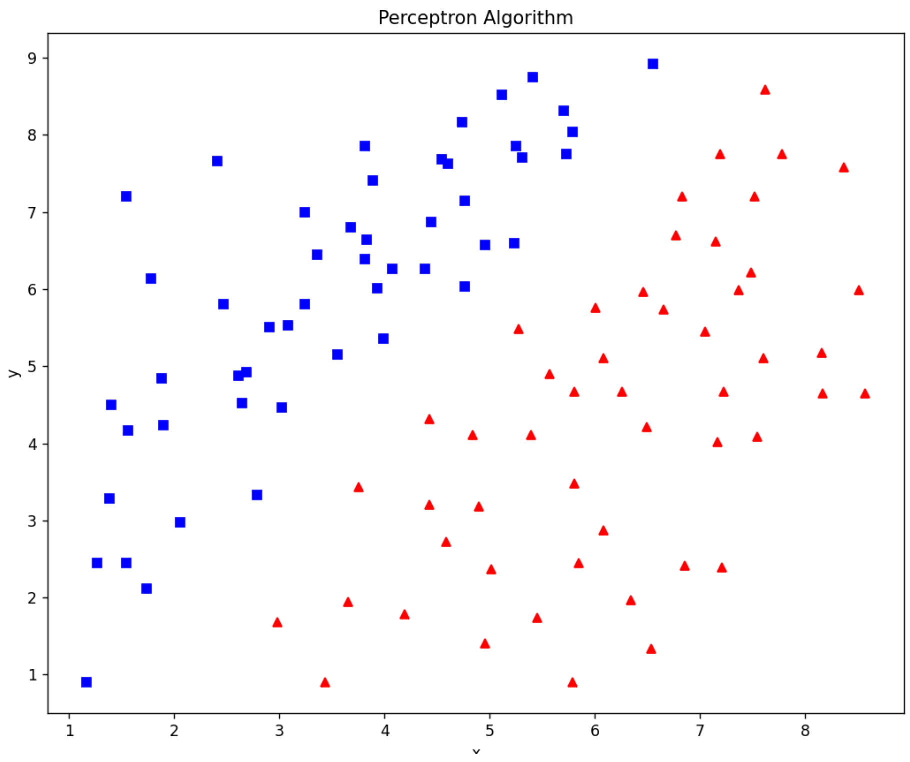
To convert this csv data into usable numpy array, we’ve used the below numpy function to separate the features and labels from the csv file.

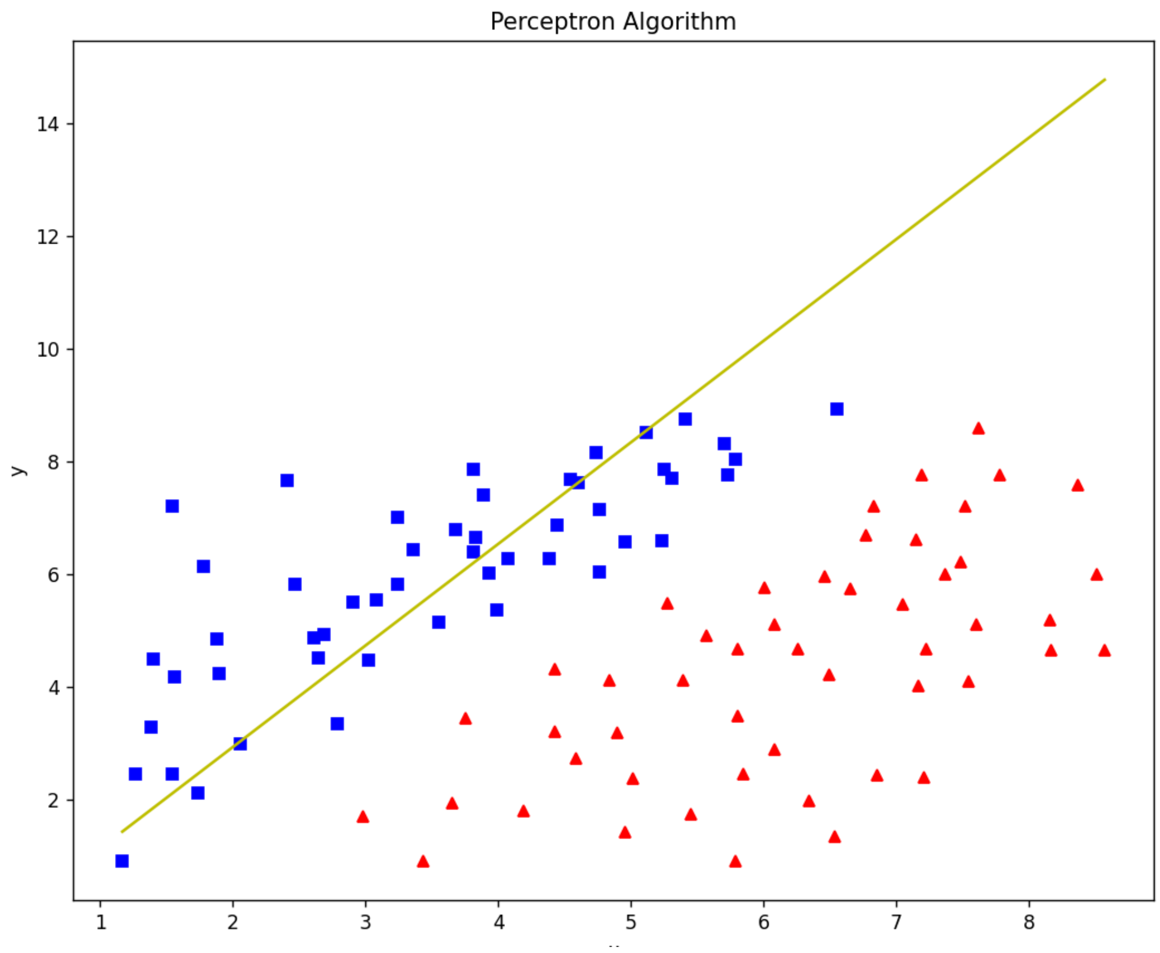
```
x = np.genfromtxt('data.csv',delimiter=',',usecols=(1,2))
y = np.genfromtxt('data.csv',delimiter=',',usecols=(0))
```

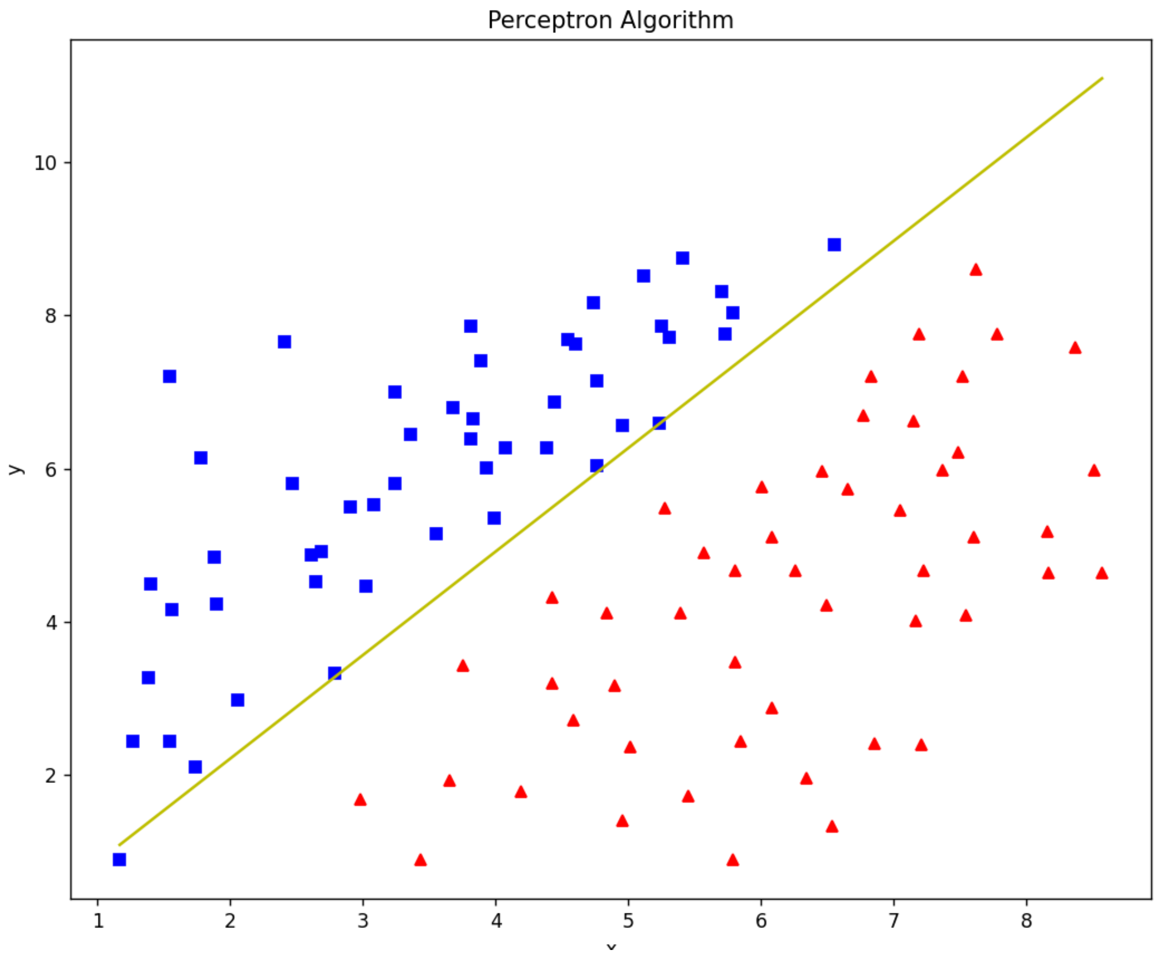
then it is used as an input to perceptron algorithm.

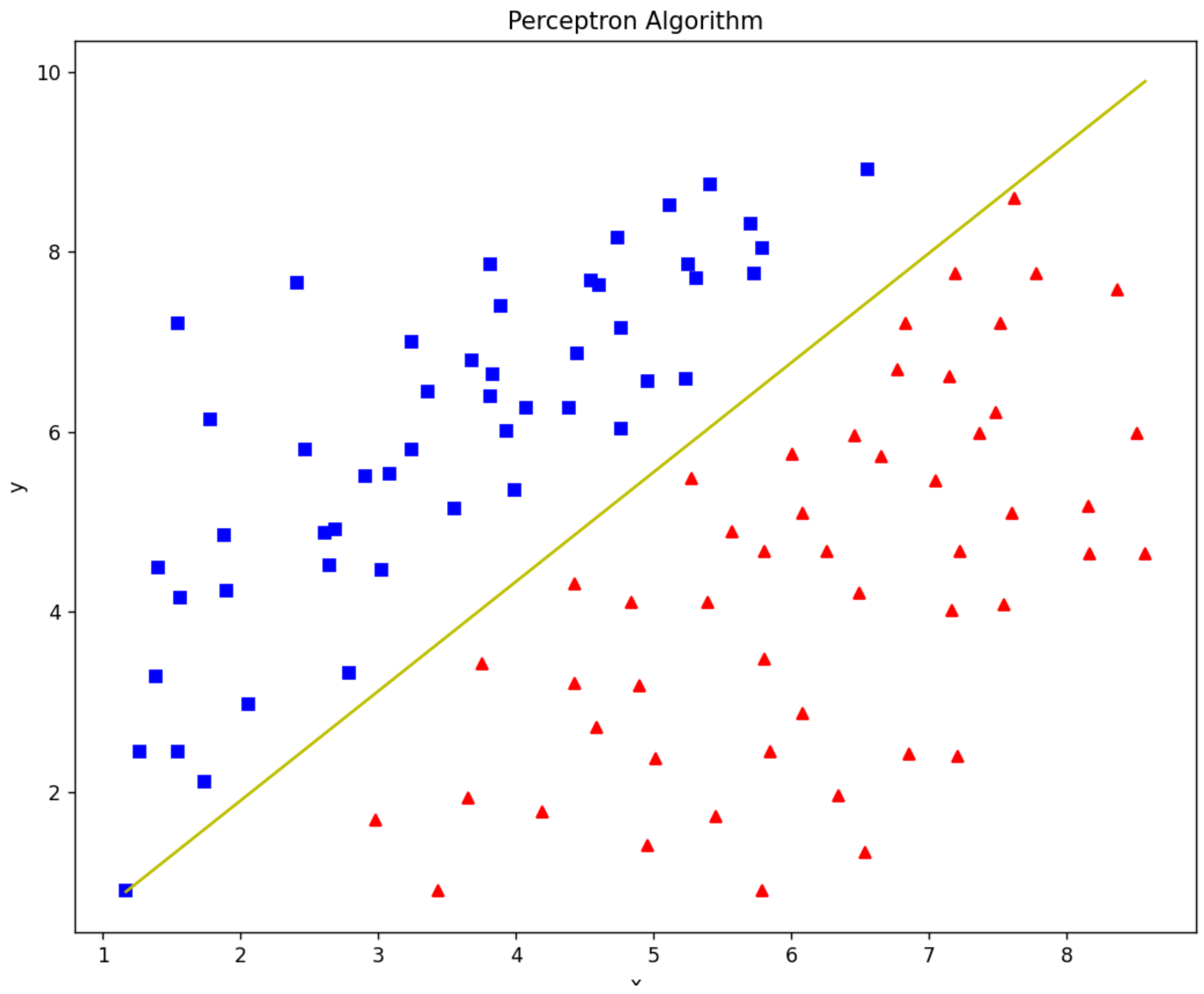


```
epoch 26

epoch 27

There are no missed points.

early stopping...
[3, 4, 2, 4, 2, 4, 3, 4, 2, 5, 2, 3, 3, 3, 2, 2, 2, 2, 2, 3, 1, 2, 3, 1, 2, 3, 1, 0]

Process finished with exit code 0
```




Perceptron Algorithm
y
14
12
10
8
6
4
2
1
2
3
4
5
6
7
8

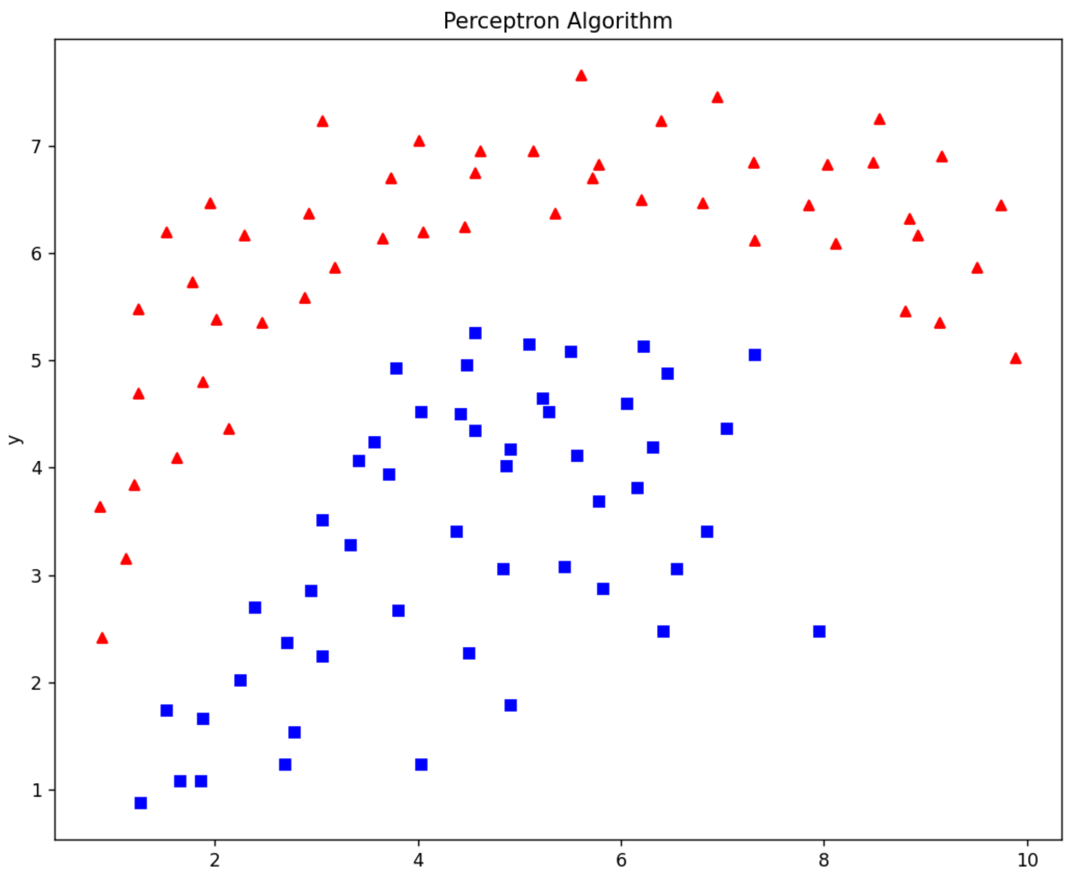

Perceptron Algorithm
y
10
8
6
4
2
1
2
3
4
5
6
7
8

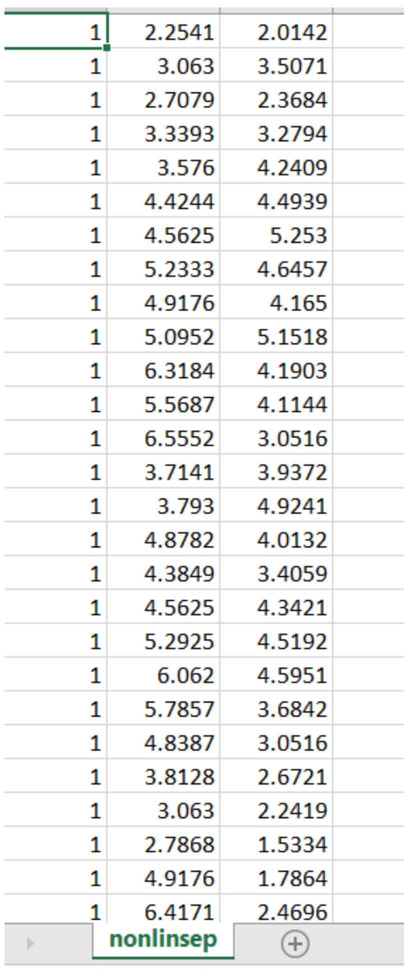


Only the decision boundaries with significant changes are shown from all 27 epochs.

Number of epochs: 27

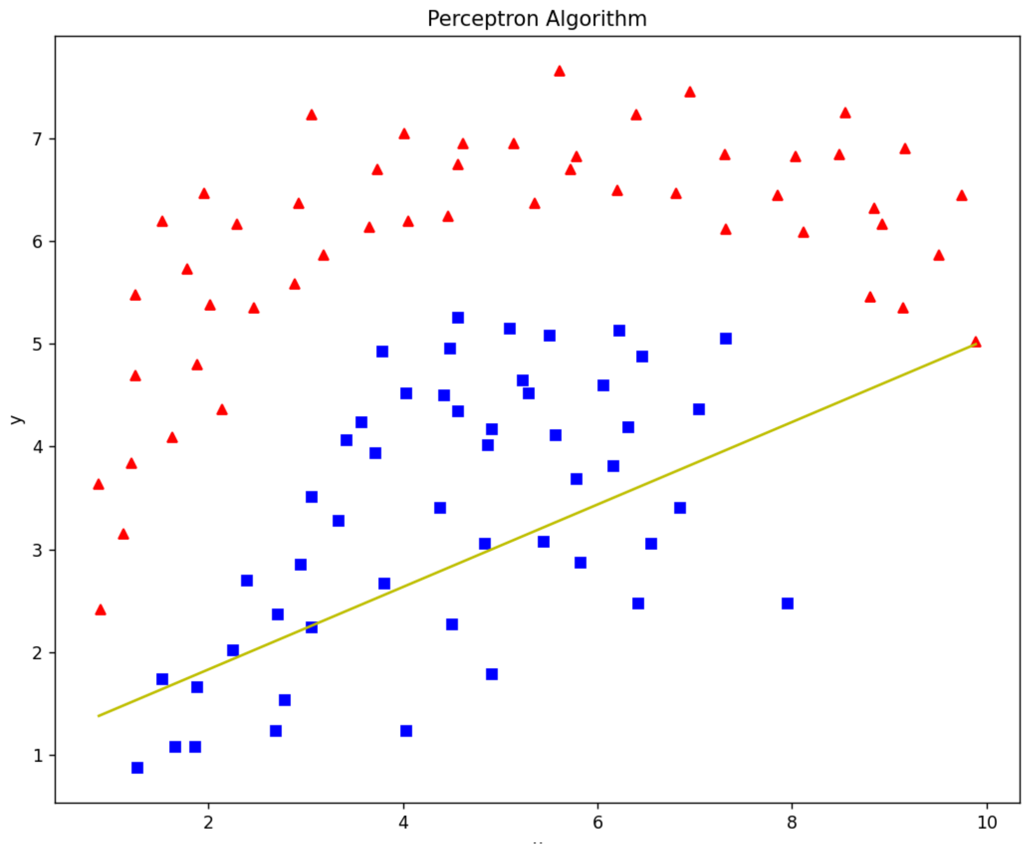
1) Non-linearly separable csv dataset.

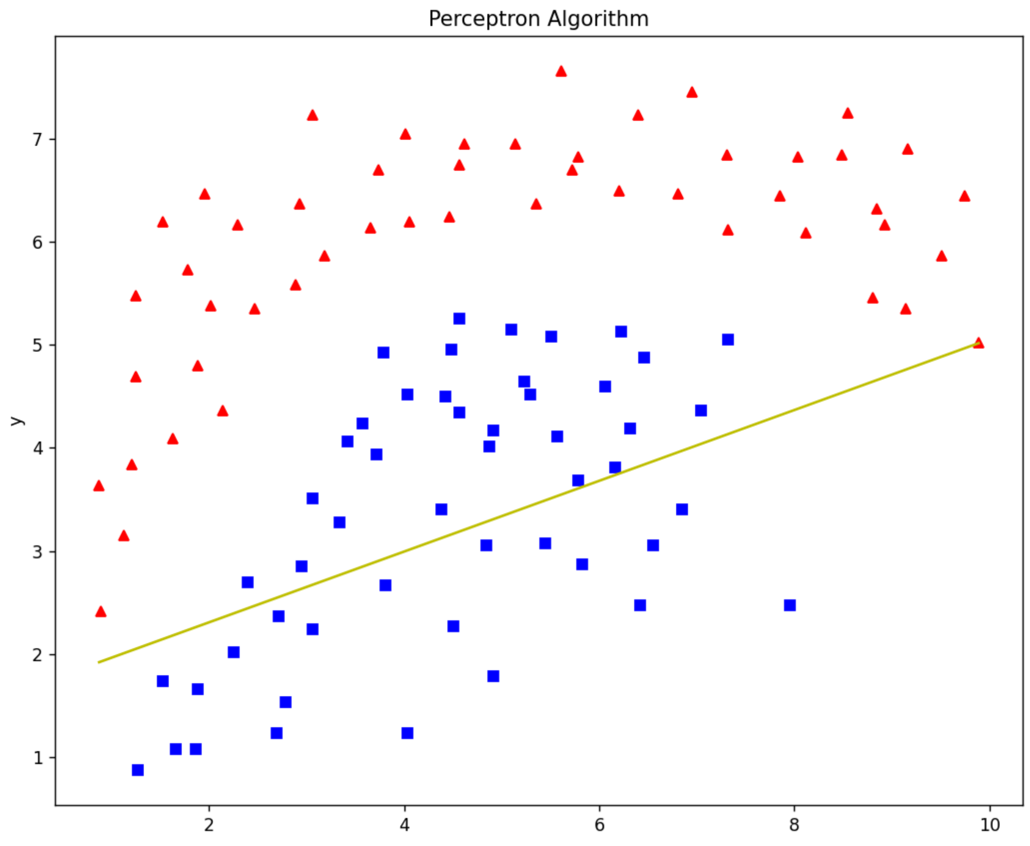




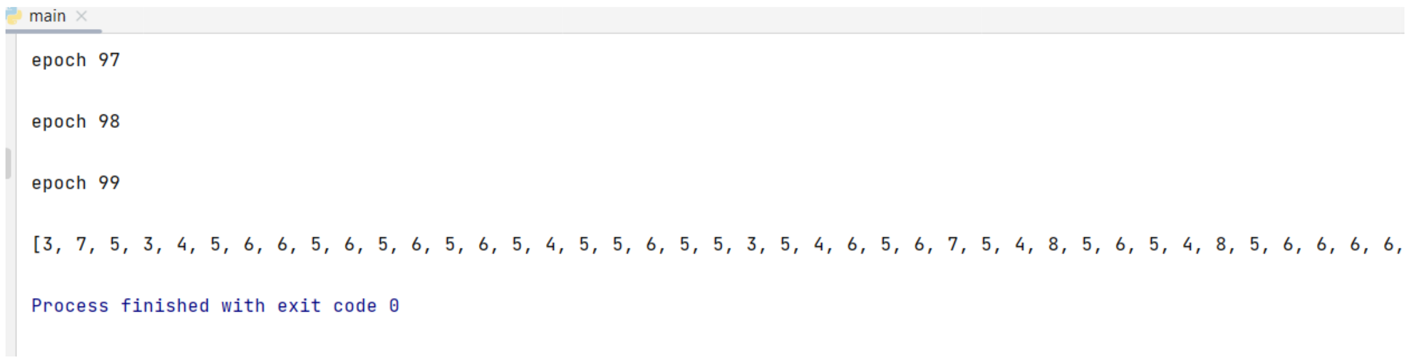
| | | |
|---|---|---|
| 1 | 2.2541 | 2.0142 |
| 1 | 3.063 | 3.5071 |
| 1 | 2.7079 | 2.3684 |
| 1 | 3.3393 | 3.2794 |
| 1 | 3.576 | 4.2409 |
| 1 | 4.4244 | 4.4939 |
| 1 | 4.5625 | 5.253 |
| 1 | 5.2333 | 4.6457 |
| 1 | 4.9176 | 4.165 |
| 1 | 5.0952 | 5.1518 |
| 1 | 6.3184 | 4.1903 |
| 1 | 5.5687 | 4.1144 |
| 1 | 6.5552 | 3.0516 |
| 1 | 3.7141 | 3.9372 |
| 1 | 3.793 | 4.9241 |
| 1 | 4.8782 | 4.0132 |
| 1 | 4.3849 | 3.4059 |
| 1 | 4.5625 | 4.3421 |
| 1 | 5.2925 | 4.5192 |
| 1 | 6.062 | 4.5951 |
| 1 | 5.7857 | 3.6842 |
| 1 | 4.8387 | 3.0516 |
| 1 | 3.8128 | 2.6721 |
| 1 | 3.063 | 2.2419 |
| 1 | 2.7868 | 1.5334 |
| 1 | 4.9176 | 1.7864 |
| 1 | 6.4171 | 2.4696 |

Perceptron algorithm can't perform nonlinear classification or implement arbitrary nonlinear functions.
It can only classify linearly separable datasets.





In this way the model will continue to move that decision boundary but will never find a perfect boundary to classify.



```
epoch 97

epoch 98

epoch 99

[3, 7, 5, 3, 4, 5, 6, 6, 5, 6, 5, 6, 5, 6, 5, 4, 5, 5, 6, 5, 5, 3, 5, 4, 6, 5, 6, 7, 5, 4, 8, 5, 6, 5, 4, 8, 5, 6, 6, 6, 6,

Process finished with exit code 0
```

All 100 epochs are finished but, perfect decision boundary is not found.

## Limitations of Perceptron Algorithm:

- It is only a linear classifier, can never separate data that are not linearly separable.
- The algorithm is used only for Binary Classification problems.