Assignment 1

Introduction to Machine Learning

ENPM 808A

Name : Govind Ajith Kumar

UID : 116699488

Question_1

September 23, 2020

[1]:
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn
import sklearn.linear_model
```

[2]:
```
col_list = ["X1"]
# Load the data
sales_data = pd.read_csv("mlr05.csv", usecols=col_list)[:20]
print(sales_data)
```

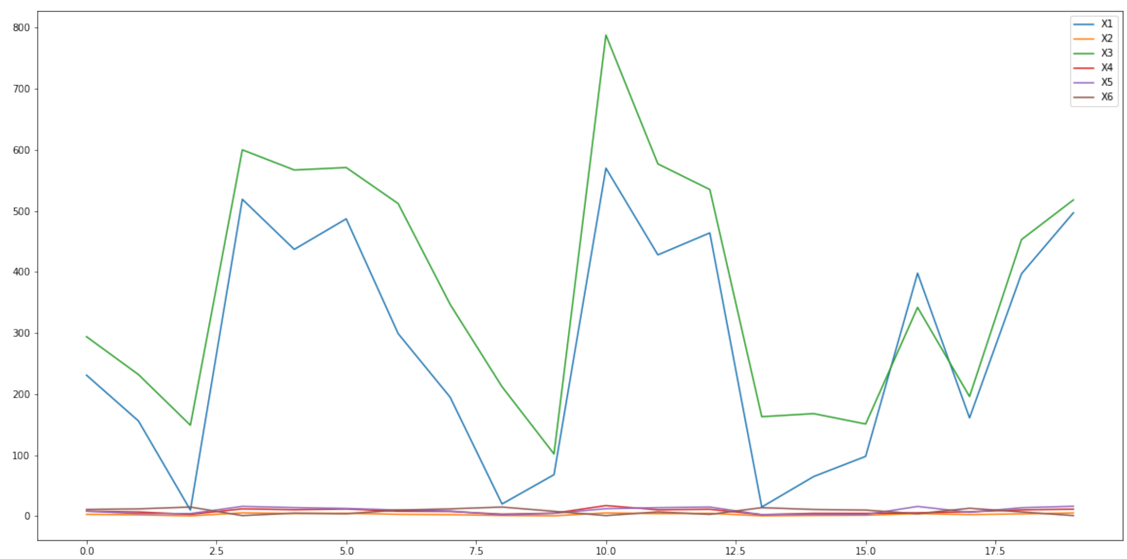
```
       X1
0   231.0
1   156.0
2    10.0
3   519.0
4   437.0
5   487.0
6   299.0
7   195.0
8    20.0
9    68.0
10  570.0
11  428.0
12  464.0
13   15.0
14   65.0
15   98.0
16  398.0
17  161.0
18  397.0
19  497.0
```

[3]:
```
predictor_col_list = ["X2","X3","X4","X5","X6"]
#Loading predictors
predictor = pd.read_csv("mlr05.csv", usecols=predictor_col_list)[:20]
print(predictor)
```

	X2	X3	X4	X5	X6
0	3.0	294	8.2	8.200000	11
1	2.2	232	6.9	4.100000	12
2	0.5	149	3.0	4.300000	15
3	5.5	600	12.0	16.100000	1
4	4.4	567	10.6	14.100000	5
5	4.8	571	11.8	12.700000	4
6	3.1	512	8.1	10.100000	10
7	2.5	347	7.7	8.400000	12
8	1.2	212	3.3	2.100000	15
9	0.6	102	4.9	4.700000	8
10	5.4	788	17.4	12.300000	1
11	4.2	577	10.5	14.000000	7
12	4.7	535	11.3	15.000000	3
13	0.6	163	2.5	2.500000	14
14	1.2	168	4.7	3.300000	11
15	1.6	151	4.6	2.700000	10
16	4.3	342	5.5	16.000000	4
17	2.6	196	7.2	6.300000	13
18	3.8	453	10.4	13.900000	7
19	5.3	518	11.5	16.299999	1

[4]:
```
# Prepare the data
ax = sales_data.plot(figsize=(20,10))
predictor.plot(ax=ax)
```

[4]: <matplotlib.axes._subplots.AxesSubplot at 0x1859df20f88>



[5]:
```
# Select a linear model
lin_reg_model = sklearn.linear_model.LinearRegression()
```

[6]:
```
# Train the model
lin_reg_model.fit(predictor,sales_data)
```

[6]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[7]:
```
print('Final predicted X1 ValueS: ')
for i in range(20,27):
    new_predictors = pd.read_csv("mlr05.csv", usecols=predictor_col_list)[i:i+1]
    #X2, X3, X4, X5, X6 values for predicting the new X1 values
    predictor_list = list(new_predictors.loc[i])
    #predicted X1 Values
    print(lin_reg_model.predict([predictor_list]))
```

```
Final predicted X1 ValueS:
[[554.4944811]]
[[71.75780418]]
[[34.23834288]]
[[351.67227227]]
[[342.77345791]]
[[524.83570362]]
[[548.58784667]]
```

[ ]:
```

```

[ ]:
```

```

Q.2

$h(x) = \text{sign}(w^T x)$

$w = [w_0, w_1, w_2]^T$

$x = [1, x_1, x_2]^T$

(a) If $h(x) = +1$

$\Rightarrow \text{sign}(w^T x) = +1$

$\Rightarrow \quad w^T x > 0$

But we know that

$W = [W_0, W_1, W_2]^T$ and $X = [X_0, X_1, X_2]^T$

$\Rightarrow W^T X = W_0 + W_1 X_1 + W_2 X_2 > 0$

Converting into the format of a line equation :

$W_2 X_2 = -W_1 X_1 - W_0$

$\Rightarrow \quad X_2 = \left(-\dfrac{W_1}{W_2}\right) X_1 - \dfrac{W_0}{W_2}$

$= mx + c,$

which is the form of a line,

where: $m = -\frac{W_1}{W_2}$ and

$$c = -\frac{W_0}{W_2}$$

(b) To draw lines for $W = [1, 2, 3]^T$

$\Rightarrow W_0 = 1$; $W_1 = 2$; $W_2 = 3$

$$\Rightarrow m = \left(-\frac{W_1}{W_2}\right) = \left(-\frac{2}{3}\right)$$

$$c = \left(-\frac{W_0}{W_2}\right) = -\frac{1}{3}$$

Hence the line would be

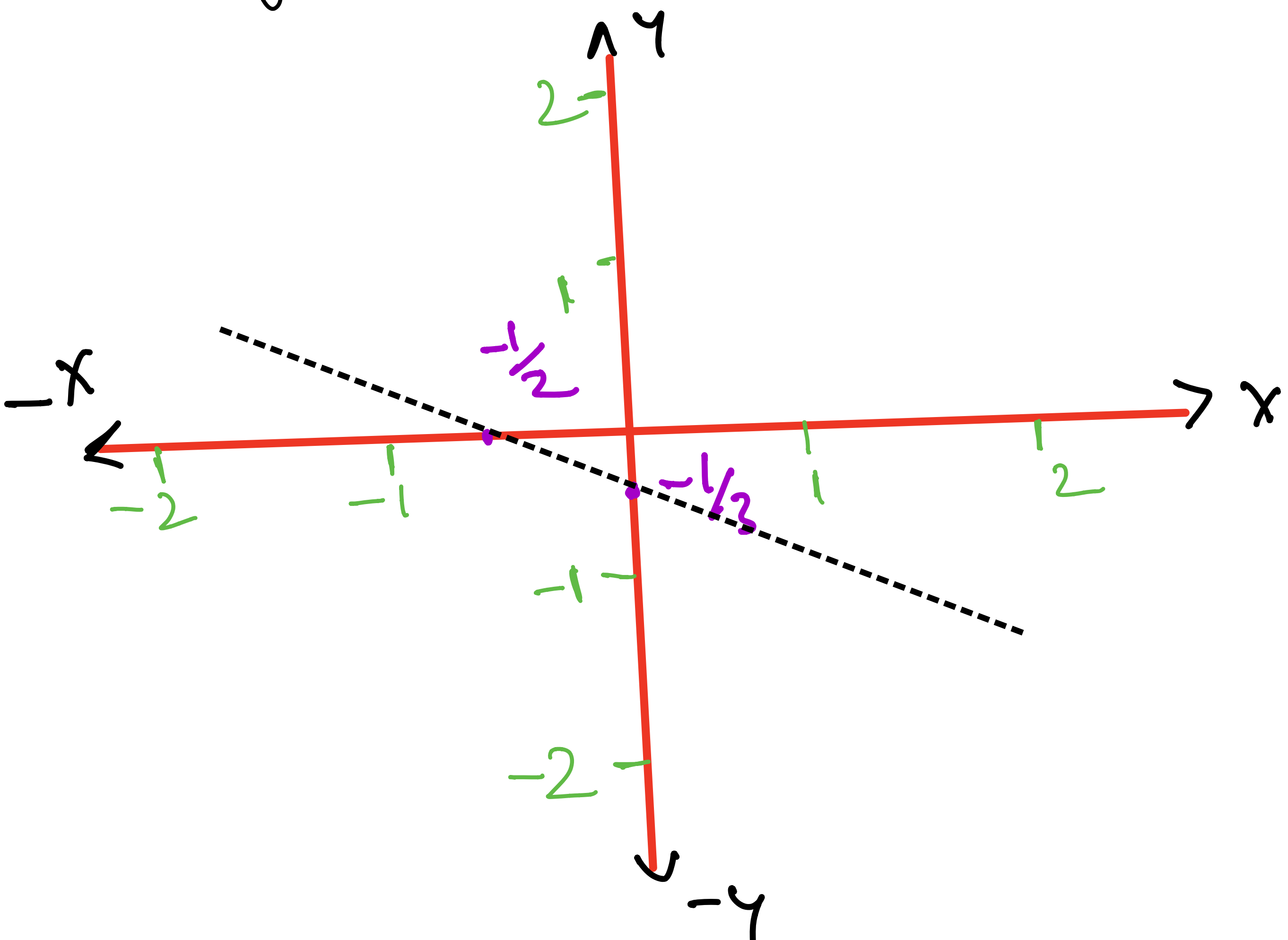
$$y = -\frac{2}{3}x - \frac{1}{3}$$

for $y = 0 \Rightarrow \frac{-2}{3}x = \frac{1}{3}$

$$\Rightarrow x\text{-intercept} = \frac{-1}{2}$$

for $x = 0$,

$y\text{-intercept} = -\frac{1}{3}$



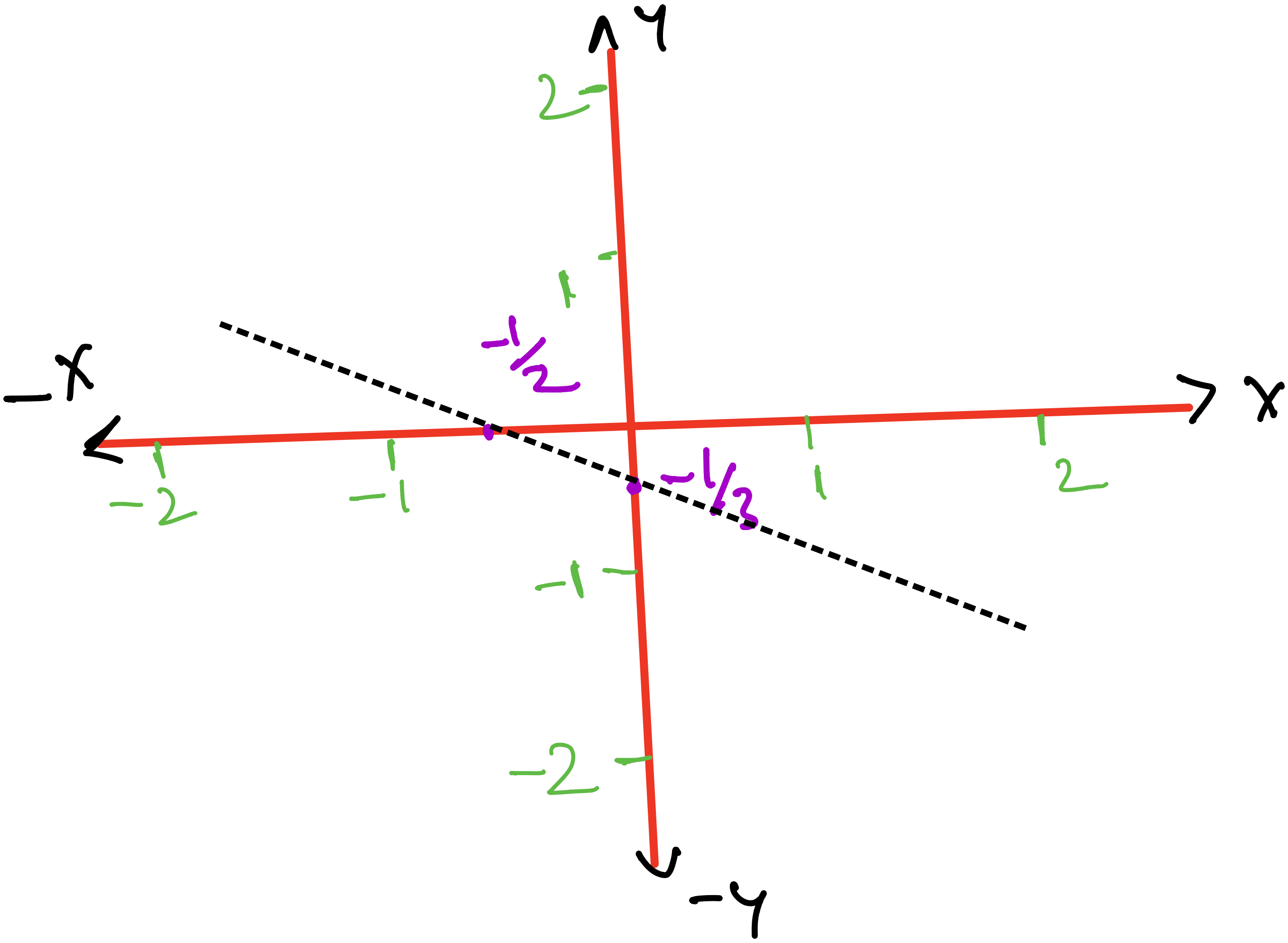
for $w = -(1, 2, 3)^T$

$\Rightarrow w_0 = -1, \quad w_1 = -2, \quad w_2 = -3$

$$m = -\left[\frac{w_1}{w_2}\right] = -\frac{(-2)}{-3} = -\frac{2}{3}$$

$$c = -\left[\frac{w_0}{w_2}\right] = -\left(\frac{-1}{-3}\right) = -\frac{1}{3}$$

Hence, the graph remains the same for this as well



Q.3 Linearly Separable dataset:

$(x_1, y_1), (x_2, y_2) \ldots - - - - - (x_n, y_n)$

$B = \min\{\|w\| : \forall i \in [m], y_i(w^T x_i) \geq 1\}$

$R = \max_i \|x_i\|$

<u>To prove:</u> Perceptron algorithm stops after at most $(RB)^2$ iterations

For the perceptron algorithm to get exhausted it must have solved all the cases. This means that, if we have 'T' cases we should prove:

$$T \leq (RB)^2$$

$$\Rightarrow \frac{\sqrt{T}}{RB} \leq 1$$

We know that $w^1 = 0$, and let $w^* = \arg\min\{\|w\| : \langle w_i, x_i \rangle y_i \geq 1\}$

$$\langle w^*, w^{t+1} \rangle = \langle w^*, w^{(t)} + y_i x_i \rangle$$

$$\Rightarrow \langle w^*, w^{t+1} \rangle - \langle w^*, w^t \rangle$$

$$= \langle w^*, w^{(t)} + y_i x_i \rangle - \langle w^*, w^t \rangle$$

$$= \langle w^*, y_i x_i \rangle$$

which is ≥ 1

Hence as we stated earlier, if we have 'T' cases and go through T iterations,

$$\langle w^*, w^{(T+1)} \rangle \geq T$$

To find out the maximum value

$$\|w^{(t+1)}\|^2 = \|w^{(t)} + y_i x_i\|^2$$

$$= \|w^{(t)}\|^2 + \|y_i x_i\|^2 + 2 y_i \langle w^{(t)}, x_i \rangle$$

$$\leq \|w^{(t)}\|^2 + \|y_i x_i\|^2$$

But $R = \max_i \|x_i\|$

$$\therefore \|w^{(t+1)}\|^2 \leq \|w^{(t)}\|^2 + R^2$$

Now, after T iterations:

$$\|w^{(T+1)}\|^2 \leq TR^2 \quad (\because w^{(1)} = 0)$$

But $\langle w^*, w^{(T+1)} \rangle \geq T$

and now; $\|w^{T+1}\| \leq \sqrt{T}R$

Combining these 2, we have:

$$\frac{\langle w^*, w^{(T+1)} \rangle}{\|w^*\| \|w^{(T+1)}\|} \geq \frac{T}{B\sqrt{T}R}$$

$$\Rightarrow \sqrt{T} \leq RB$$

$$\Rightarrow \boxed{T \leq (RB)^2}$$

Hence proved.

Question_4_Complete

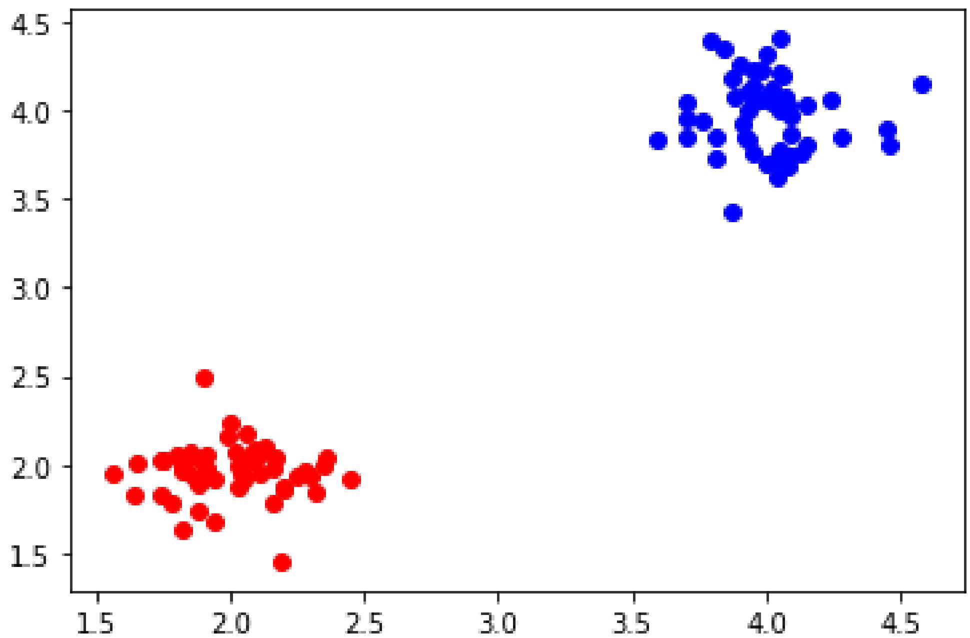
September 28, 2020

[1]:
```python
#importing necessary libraries
import numpy as np
from matplotlib import pyplot as plt
import random
random.seed(101)
```

[2]:
```python
# user chooses the length of the data
length_dataset = 100
length_test_dataset = 10000
```

[3]:
```python
#creating the linearly separable dataset
x1 = np.random.normal(4, 0.2, (int(length_dataset/2),2))
x2 = np.random.normal(2, 0.2, (int(length_dataset/2),2))
all_input = np.concatenate((x1, x2)) #creating a combined dataset of
```

[4]:
```python
#Visualizing the linearly separable dataset
plt.scatter(x1[:,0], x1[:,1], color='blue')
plt.scatter(x2[:,0], x2[:,1], color='red')
#separating the blue and the red points and categorizing the same
d1 = -1 * (np.ones(int(length_dataset/2)))
d2 = np.ones(int(length_dataset/2))
all_combined_targets = np.concatenate((d1,d2))
plt.show()
```



[5]:
```
def Y_predict(x_vector,w):
    x_new = [1]
    for i in x_vector:
        x_new.append(i)
    x_new = np.array((x_new))
    res = (np.dot(x_new,w))
    if res > 0:
        Y = 1
        return Y
    elif res < 0:
        Y = -1
        return Y
    elif res ==0:
        Y = 0
        return Y


def train(X,iterations,eta):
    global count
    global w
    global all_combined_targets
    for y_idx in range (len(X)):
        ran_num = random.randint(0,len(X)-1)
        x_train = X[ran_num]
        y_t = Y_predict(x_train,w)
        misrepresented_list = []
```

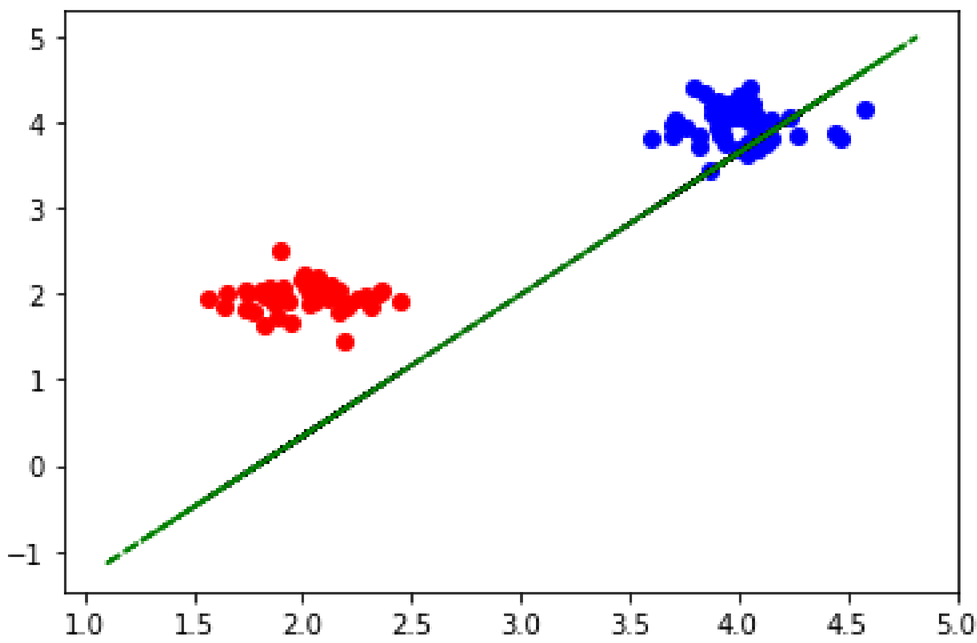
```
        for i,j in enumerate(all_combined_targets):
            if j!=y_t:
                misrepresented_list.append(i)
        if len(misrepresented_list)==0:
            print('Full accuracy achieved')
            break
        random_selection = random.randint(0,len(misrepresented_list)-1)
        random_index = misrepresented_list[random_selection]
        x_selected = X[random_index]
        y_selected = all_combined_targets[random_index]
#         print(x_selected,y_selected)
        x_with1 = [1]
        for i in x_selected:
            x_with1.append(i)
        x_with1 = np.array((x_with1))
#         print('old w - > ',w)
        s_t = np.matmul(w,x_with1)
#         print('x_with1',x_with1)
#         print('s_t',s_t)
#         print('y_selected',y_selected)
#         print('y_selected*s_t',y_selected*s_t)
        if (y_selected*s_t)<=1:
            w = w+(eta*(y_selected-s_t)*x_with1)
#             print('w - > ' , w)
#             print(' ')
#             print(' ')
#             print(' ')
        if (count==iterations):
            print('maximum iterations reached in the training block')
            break
            count+=1
```

0.1 eta = 100

[6]:
```
#initializing all parameters
count = 0
w0 = random.randint(1,4)
w1 = random.randint(1,4)
w2 = random.randint(1,4)
w = np.array((w0,w1,w2))
weight= 0
iterations = 20
eta = 0.001
#calling the function
train(all_input,iterations,eta)
```

[7]:
```
#Visualizing the linearly separable dataset
plt.scatter(x1[:,0], x1[:,1], color='blue')
plt.scatter(x2[:,0], x2[:,1], color='red')
m = -(w[1]/w[2])
c = -(w[0]/w[2])
plt.plot(all_input, m*all_input + c,'k--')
#test plotting
#creating the linearly separable dataset
xtest = np.random.normal(4, 0.2, (int(length_test_dataset/2),2))
x2test = np.random.normal(2, 0.2, (int(length_test_dataset/2),2))
all_input_test = np.concatenate((xtest, x2test)) #creating a combined dataset of
plt.plot(all_input_test, m*all_input_test + c,'g--')
print('w:',w)
```

```
w: [ 1.08709874 -0.60569865  0.36709108]
```



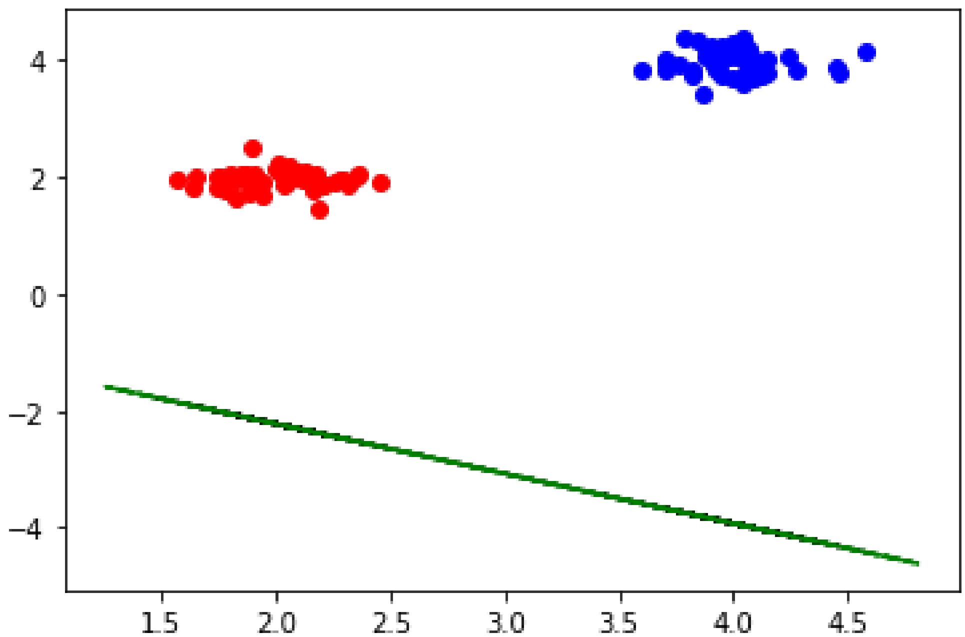
0.2 eta = 1

[8]:
```
#initializing all parameters
count = 0
w0 = random.randint(1,4)
w1 = random.randint(1,4)
w2 = random.randint(1,4)
w = np.array((w0,w1,w2))
weight= 0
iterations = 20
```

```
eta = 1
#calling the function
train(all_input,iterations,eta)
```

[9]:
```
#Visualizing the linearly separable dataset
plt.scatter(x1[:,0], x1[:,1], color='blue')
plt.scatter(x2[:,0], x2[:,1], color='red')
m = -(w[1]/w[2])
c = -(w[0]/w[2])
plt.plot(all_input, m*all_input + c,'k--')
#test plotting
#creating the linearly separable dataset
xtest = np.random.normal(4, 0.2, (int(length_test_dataset/2),2))
x2test = np.random.normal(2, 0.2, (int(length_test_dataset/2),2))
all_input_test = np.concatenate((xtest, x2test)) #creating a combined dataset of
plt.plot(all_input_test, m*all_input_test + c,'g--')
print('w:',w)
```

```
w: [3.92614577e+118 6.48064357e+118 7.60059729e+118]
```



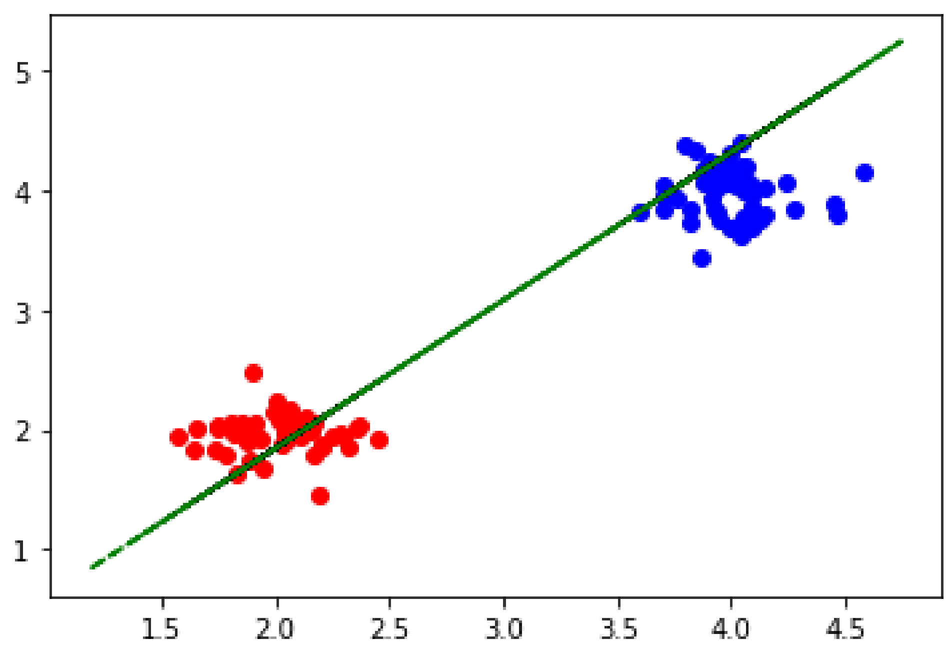
0.3 eta = 0.01

[10]:
```
#initializing all parameters
count = 0
w0 = random.randint(1,4)
w1 = random.randint(1,4)
```

```
w2 = random.randint(1,4)
w = np.array((w0,w1,w2))
weight= 0
iterations = 20
eta = 0.01
#calling the function
train(all_input,iterations,eta)
```

[11]:
```
#Visualizing the linearly separable dataset
plt.scatter(x1[:,0], x1[:,1], color='blue')
plt.scatter(x2[:,0], x2[:,1], color='red')
m = -(w[1]/w[2])
c = -(w[0]/w[2])
plt.plot(all_input, m*all_input + c,'k--')
#test plotting
#creating the linearly separable dataset
xtest = np.random.normal(4, 0.2, (int(length_test_dataset/2),2))
x2test = np.random.normal(2, 0.2, (int(length_test_dataset/2),2))
all_input_test = np.concatenate((xtest, x2test)) #creating a combined dataset of
plt.plot(all_input_test, m*all_input_test + c,'g--')
print('w:',w)
```

```
w: [ 0.51616892 -1.01058716  0.81523726]
```

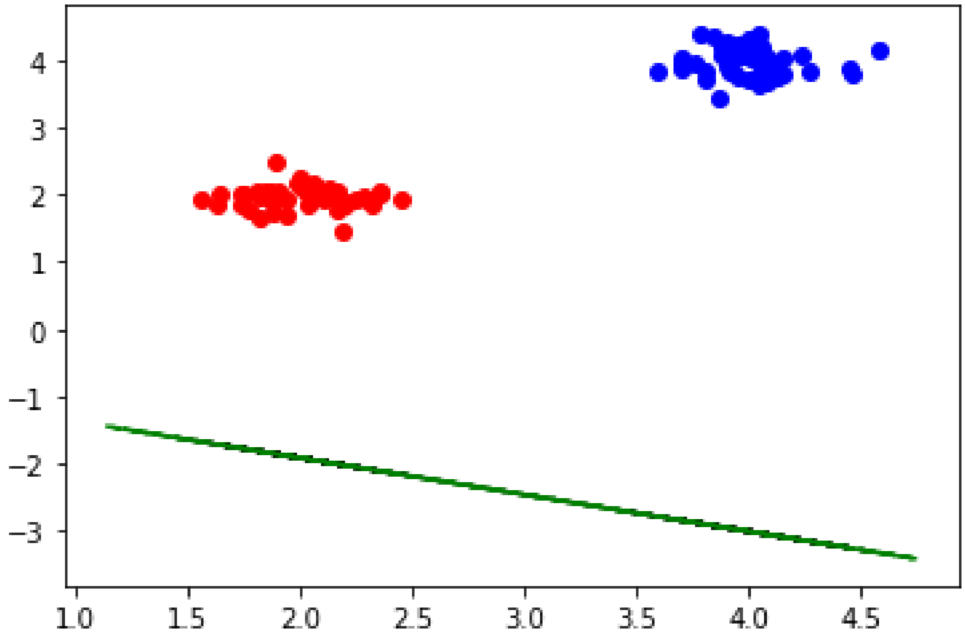


0.4 eta = 0.0001

[12]:
```
#initializing all parameters
count = 0
w0 = random.randint(1,4)
w1 = random.randint(1,4)
w2 = random.randint(1,4)
w = np.array((w0,w1,w2))
weight= 0
iterations = 20
eta = 0.0001
#calling the function
train(all_input,iterations,eta)
```

[13]:
```
#Visualizing the linearly separable dataset
plt.scatter(x1[:,0], x1[:,1], color='blue')
plt.scatter(x2[:,0], x2[:,1], color='red')
m = -(w[1]/w[2])
c = -(w[0]/w[2])
plt.plot(all_input, m*all_input + c,'k--')
#test plotting
#creating the linearly separable dataset
xtest = np.random.normal(4, 0.2, (int(length_test_dataset/2),2))
x2test = np.random.normal(2, 0.2, (int(length_test_dataset/2),2))
all_input_test = np.concatenate((xtest, x2test)) #creating a combined dataset of
plt.plot(all_input_test, m*all_input_test + c,'g--')
print('w:',w)
```

```
w: [1.80332769 1.20856561 2.21148356]
```



1 Comparison of Results

[15]:
```
# We can see that in case a) the results are not too bad but the line cuts
# through the linearly separable points
```

[16]:
```
# In case b) The line is very off
```

[17]:
```
# In case c) It almost verifies the linear separability
```

[18]:
```
# In case d) The line goes off the path again
```

[19]:
```
# Conclusion the eta value that resulted in the minimu classification error␣
 ↪wass when eta = 0.01
```

[ ]:
```
```