Question 1

```
# import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
# import pandas as pd


def perceptron():
    w1 = np.array([[1, 0, 0], [1, 0, 1], [1, 1, 0]])
    w2 = np.array([[1, 1, 1]])
    data = np.concatenate([w1, -w2], axis = 0)
    print(data)
    w1 = np.array([[0, 0], [0, 1], [1, 0]])
    w2 = np.array([[1, 1]])
    learning_rate = 0
    k = 0
    Q = 0                           #criterian theta
    a = np.array([0, 0, 0])            #weights
    missclasified_vectors = True
    epoch = 1
    while((missclasified_vectors) or not(k == 0)):
        if(k == 0):
            print("\n\n\nEpoch : ", epoch)
            epoch = epoch + 1
            missclasified_vectors = False
        print("Data index : ", k)
        print("weights : ", a)
        print("Data point : ", np.array(data[k]))
        dot_product = np.dot(a, data[k])
        print("The dot product is : ", dot_product)
        print(dot_product)
        if(dot_product <= 0):
            f, ax = plt.subplots(figsize=(7, 7))
            c1, c2 = "#3366AA", "#AA3333"
            ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
            ax.legend()
            ax.scatter(*w2.T, c=c2, marker="D", label = "w2")
            ax.legend()
            a = a + data[k]
            x_vec = np.linspace(-0.5, 2, 5)
            y_vec = ((a[1] * x_vec) + a[0])/a[2]
            # print(y_vec)
            y_vec = -y_vec
            plt.plot(x_vec, y_vec, 'r--')

            missclasified_vectors = True
        k = (k + 1) % (len(data))

    print("The final weight vector is : ", a)
    x_vec = np.linspace(-0.5, 2, 5)
    y_vec = ((a[1] * x_vec) + a[0])/a[2]
```

```
        # print(y_vec)
        y_vec = -y_vec
        plt.plot(x_vec, y_vec, 'r--')
        plt.show()

    z = []
    data = []
    alpha = []

    def func(alpha):
        alpha = np.array(alpha)
        value = np.sum(alpha)
        for i in range(len(alpha)):
            for j in range(len(alpha)):
                if(i == j):
                    pass
                value -= 0.5 * (alpha[i] * alpha[j] * z[i] * z[j] * (data[i].T.dot(data[j])))
        return (-value)

    def constraint(alpha):
        global z
        sum = 0
        for alpha_i, z_i in zip(alpha, z):
            sum += alpha_i * z_i
        return(sum)

    def abs(num):
        if(num < 0):
            return(-num)
        else:
            return(num)

    def svm():
        global z
        global data
        global alpha
        z = np.array([1, 1, 1, -1])
        w1 = np.array([[0, 0], [1, 0], [0, 1]])
        w2 = np.array([[1, 1]])
        data = np.concatenate([w1, w2], axis = 0)
        print("data : ", data)
        max_solution = 0
        consrt = {'type' : 'eq', 'fun' : constraint}
        alpha = np.zeros(len(data))
        b = (0.0000000000, None)
        bound = [b, b, b, b]
        sol = minimize(func, alpha, method = 'SLSQP', constraints = consrt, bounds = bound)
        print(sol)
        alpha = sol.x

        print("lagrange multiplier are : ", alpha)

        a = np.zeros(len(data[0]))
        for i in range(len(data)):
            a += alpha[i]*z[i]*data[i]
```

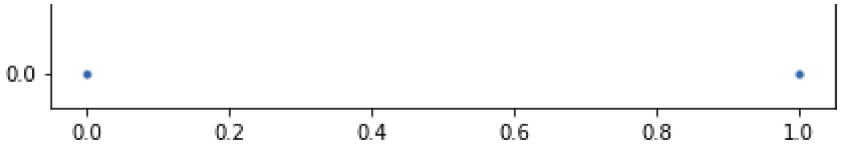
```
    bias = []                                           #finding the points that pass throu
    for i in range(len(alpha)):
        for j in range(len(alpha)):
            if(i == j):
                pass
            else:
                if(abs(0.5 * (alpha[i] * alpha[j] * z[i] * z[j] * (data[i].T.dot(data[j]))
                    bias.append(1/z[i] - a.T.dot(data[i]))

    bias = np.array(bias).mean()
    print("bias is ", bias)
    print("weight vector is : ", a)
    a = [bias, a[0] , a[1]]
    f, ax = plt.subplots(figsize=(7, 7))
    c1, c2 = "#3366AA", "#AA3333"

    x_vec = np.linspace(-0.5, 2, 5)
    y_vec = ((a[1] * x_vec) + a[0])/a[2]
    y_vec = -y_vec
    plt.plot(x_vec, y_vec, 'r--')
    y_vec = y_vec + (1 / a[2])
    plt.plot(x_vec, y_vec, 'g--')
    y_vec = y_vec - (2/ a[2])
    plt.plot(x_vec, y_vec, 'g--')

    ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
    ax.legend()
    ax.scatter(*w2.T, c=c2,s = 10, marker="D", label = "w2")
    ax.legend()

    plt.show()


if(__name__ == '__main__'):
    perceptron()
    svm()
```

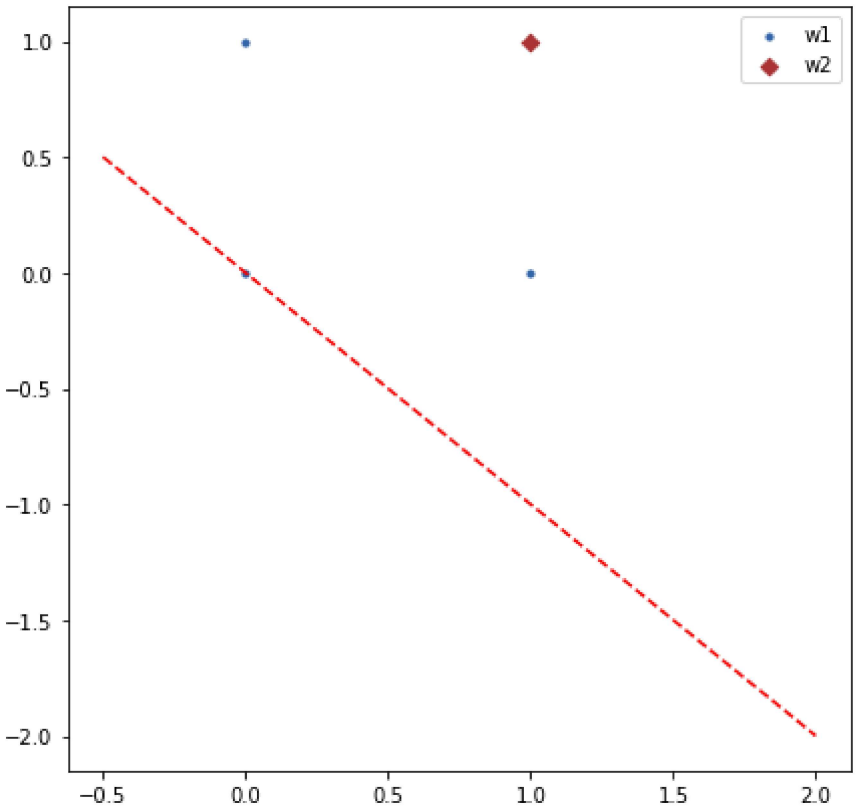
```
[[ 1  0  0]
 [ 1  0  1]
 [ 1  1  0]
 [-1 -1 -1]]



Epoch :  1
Data index :  0
weights :  [0 0 0]
Data point :  [1 0 0]
The dot product is :  0
0
Data index :  1
weights :  [1 0 0]
Data point :  [1 0 1]
The dot product is :  1
1
Data index :  2
weights :  [1 0 0]
Data point :  [1 1 0]
The dot product is :  1
1
Data index :  3
weights :  [1 0 0]
Data point :  [-1 -1 -1]
The dot product is :  -1
-1



Epoch :  2
Data index :  0
weights :  [ 0 -1 -1]
Data point :  [1 0 0]
The dot product is :  0
0
Data index :  1
weights :  [ 1 -1 -1]
Data point :  [1 0 1]
The dot product is :  0
0
Data index :  2
weights :  [ 2 -1  0]
Data point :  [1 1 0]
The dot product is :  1
1
Data index :  3
weights :  [ 2 -1  0]
Data point :  [-1 -1 -1]
The dot product is :  -1
-1



Epoch :  3
Data index :  0
weights :  [ 1 -2 -1]
Data point :  [1 0 0]
The dot product is :  1
1
```

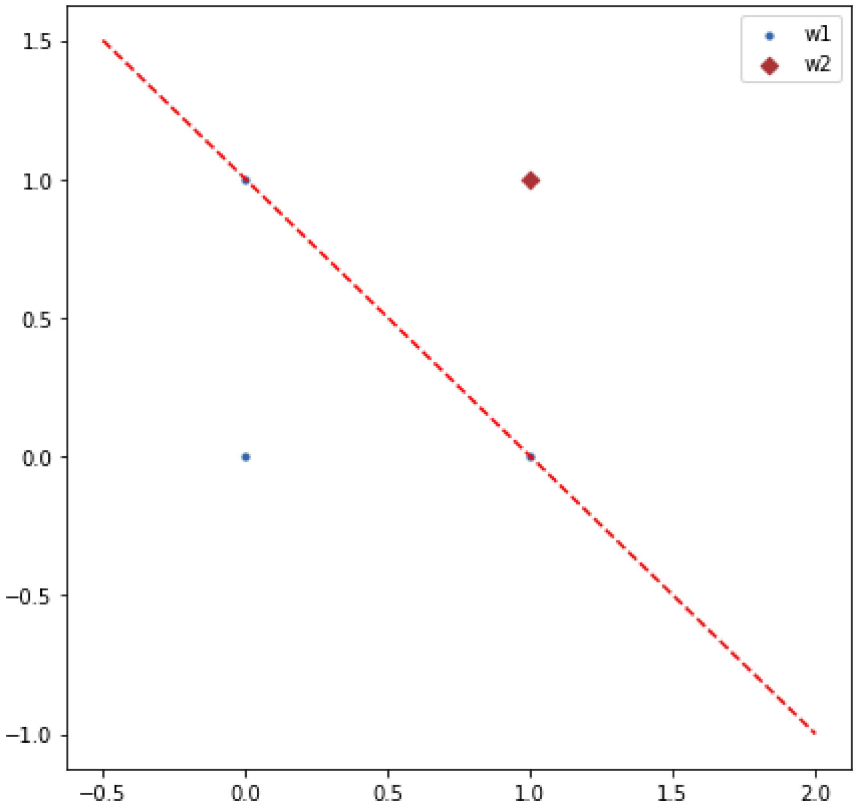
```
Data index :  1
weights :  [ 1 -2 -1]
Data point :  [1 0 1]
The dot product is :  0
0
Data index :  2
weights :  [ 2 -2  0]
Data point :  [1 1 0]
The dot product is :  0
0
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:40: RuntimeWarning: divi
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:40: RuntimeWarning: divi
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:40: RuntimeWarning: inva
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:40: RuntimeWarning: divi
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:40: RuntimeWarning: divi
Data index :  3
weights :  [ 3 -1  0]
Data point :  [-1 -1 -1]
The dot product is :  -2
-2



Epoch :  4
Data index :  0
weights :  [ 2 -2 -1]
Data point :  [1 0 0]
The dot product is :  2
2
Data index :  1
weights :  [ 2 -2 -1]
Data point :  [1 0 1]
The dot product is :  1
1
Data index :  2
weights :  [ 2 -2 -1]
Data point :  [1 1 0]
The dot product is :  0
0
Data index :  3
weights :  [ 3 -1 -1]
Data point :  [-1 -1 -1]
The dot product is :  -1
-1



Epoch :  5
Data index :  0
weights :  [ 2 -2 -2]
Data point :  [1 0 0]
The dot product is :  2
2
Data index :  1
weights :  [ 2 -2 -2]
Data point :  [1 0 1]
The dot product is :  0
0
Data index :  2
weights :  [ 3 -2 -1]
Data point :  [1 1 0]
The dot product is :  1
```

```
The dot product is :  1
1
Data index :  3
weights :  [ 3 -2 -1]
Data point :  [-1 -1 -1]
The dot product is :  0
0



Epoch :  6
Data index :  0
weights :  [ 2 -3 -2]
Data point :  [1 0 0]
The dot product is :  2
2
Data index :  1
weights :  [ 2 -3 -2]
Data point :  [1 0 1]
The dot product is :  0
0
Data index :  2
weights :  [ 3 -3 -1]
Data point :  [1 1 0]
The dot product is :  0
0
Data index :  3
weights :  [ 4 -2 -1]
Data point :  [-1 -1 -1]
The dot product is :  -1
-1



Epoch :  7
Data index :  0
weights :  [ 3 -3 -2]
Data point :  [1 0 0]
The dot product is :  3
3
Data index :  1
weights :  [ 3 -3 -2]
Data point :  [1 0 1]
The dot product is :  1
1
Data index :  2
weights :  [ 3 -3 -2]
Data point :  [1 1 0]
The dot product is :  0
0
Data index :  3
weights :  [ 4 -2 -2]
Data point :  [-1 -1 -1]
The dot product is :  0
0



Epoch :  8
Data index :  0
weights :  [ 3 -3 -3]
Data point :  [1 0 0]
```

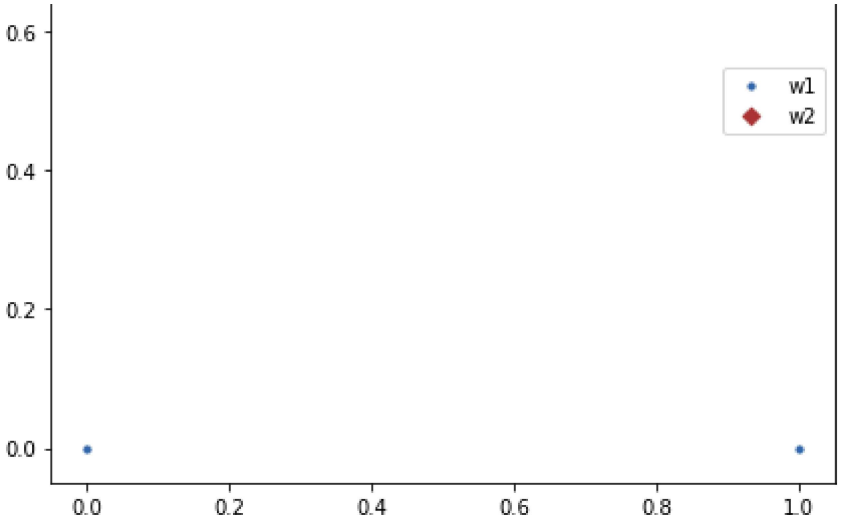
```
The dot product is :  3
3
Data index :  1
weights :  [ 3 -3 -3]
Data point :  [1 0 1]
The dot product is :  0
0
Data index :  2
weights :  [ 4 -3 -2]
Data point :  [1 1 0]
The dot product is :  1
1
Data index :  3
weights :  [ 4 -3 -2]
Data point :  [-1 -1 -1]
The dot product is :  1
1



Epoch :  9
Data index :  0
weights :  [ 4 -3 -2]
Data point :  [1 0 0]
The dot product is :  4
4
Data index :  1
weights :  [ 4 -3 -2]
Data point :  [1 0 1]
The dot product is :  2
2
Data index :  2
weights :  [ 4 -3 -2]
Data point :  [1 1 0]
The dot product is :  1
1
Data index :  3
weights :  [ 4 -3 -2]
Data point :  [-1 -1 -1]
The dot product is :  1
1
The final weight vector is :  [ 4 -3 -2]
```

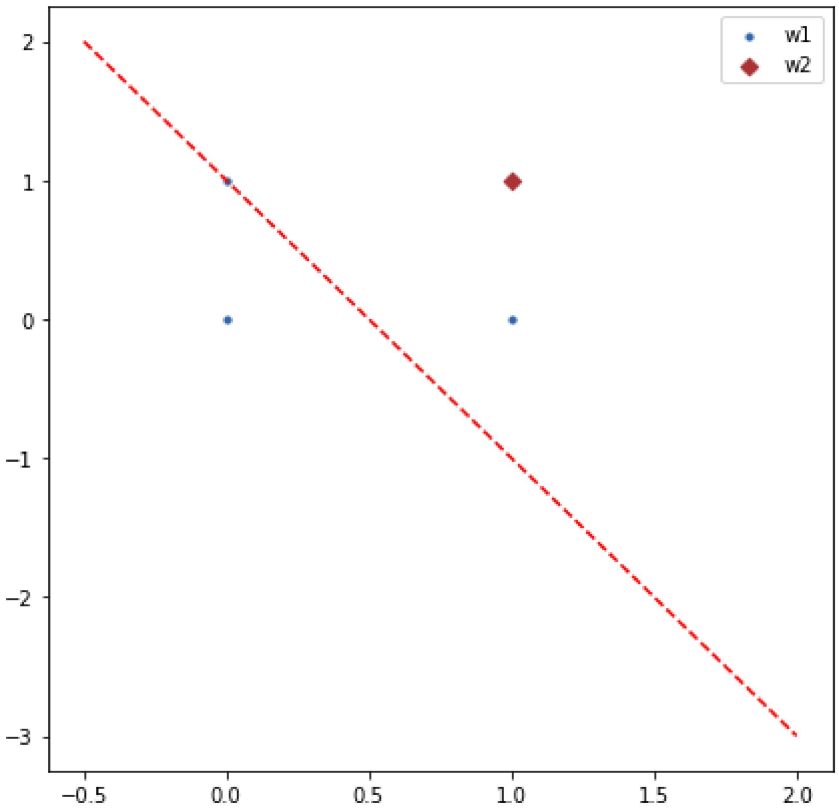

0.0
0.0
0.2
0.4
0.6
0.8
1.0

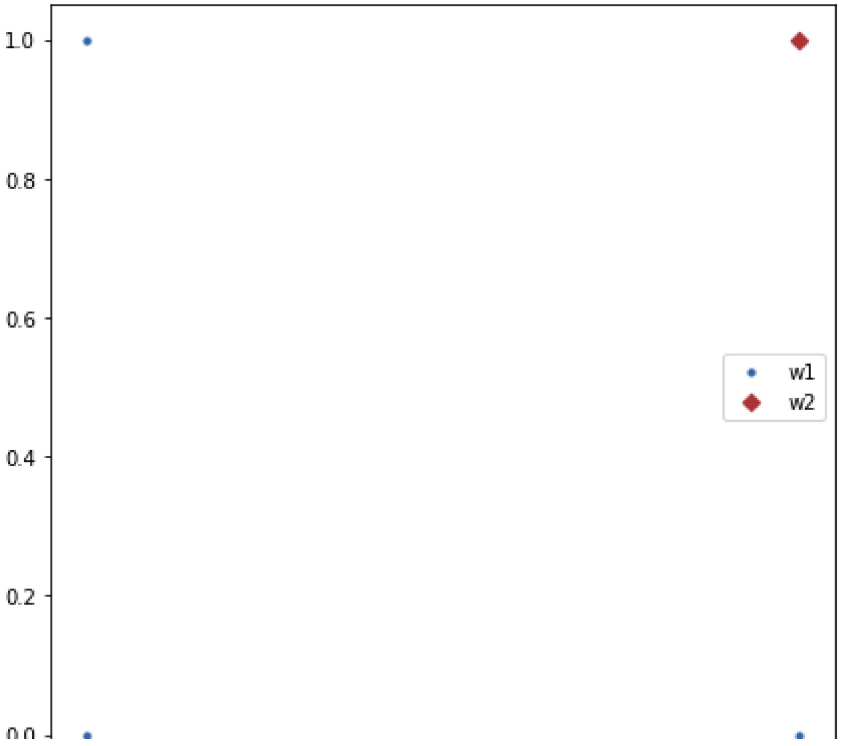

w1
w2
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−0.5
0.0
0.5
1.0
1.5
2.0

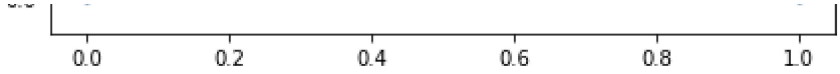

w1
w2
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

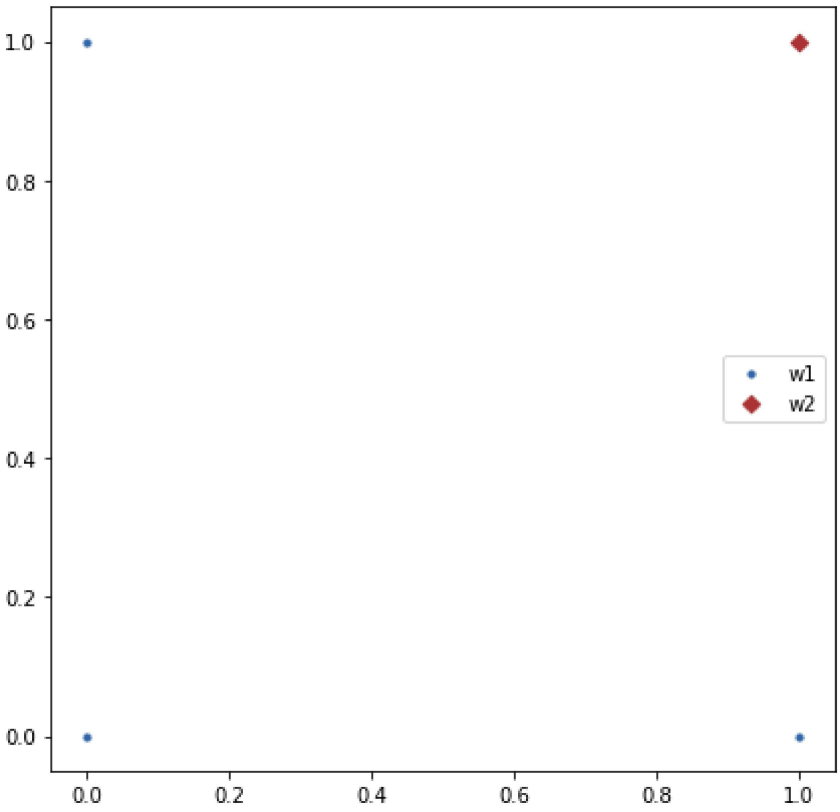

1.0
0.8

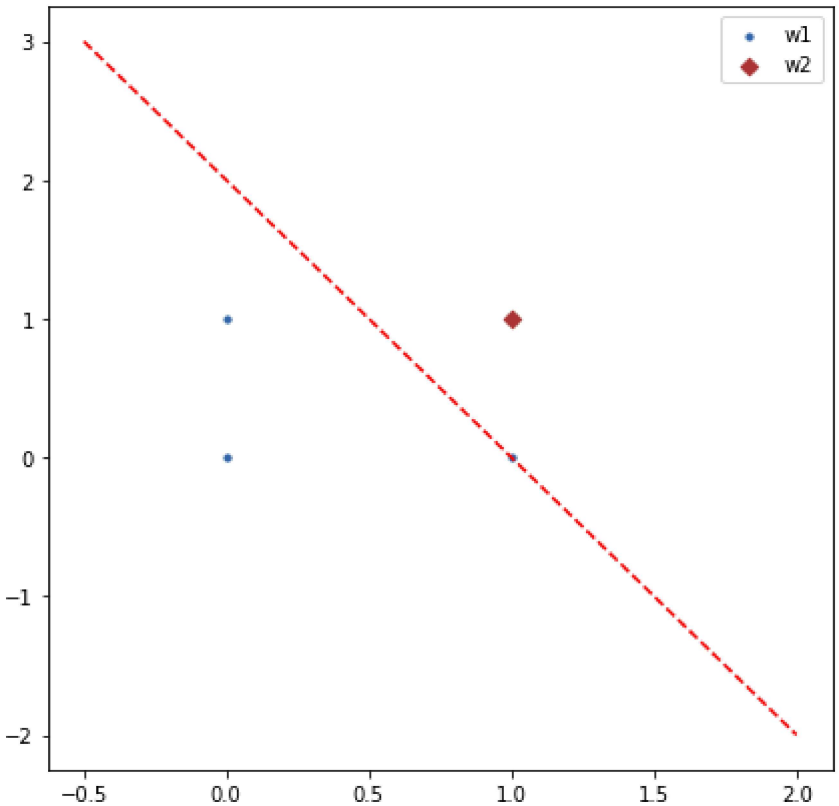

w1
w2
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

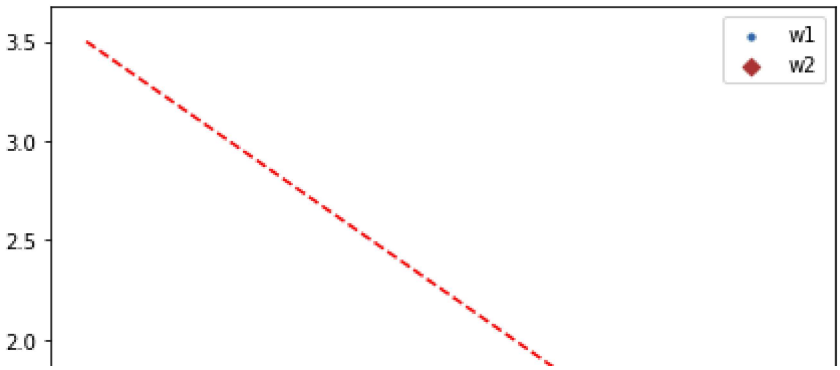

w1
w2
2
1
0
−1
−2
−3
−0.5
0.0
0.5
1.0
1.5
2.0

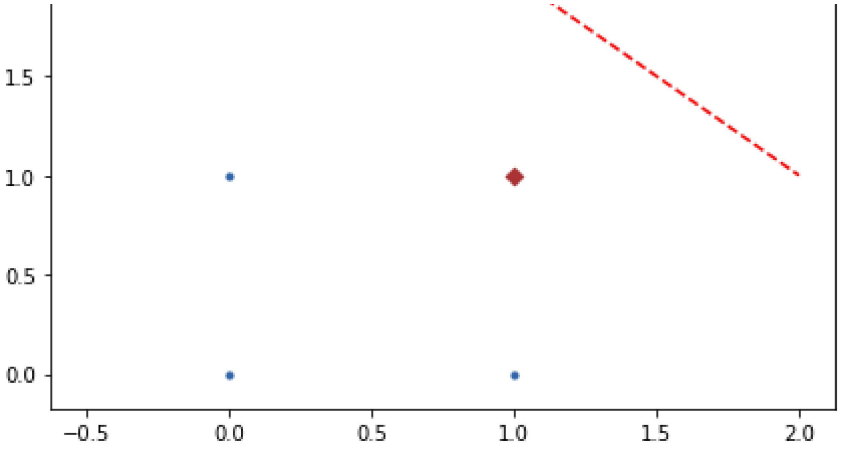

w1
w2
1.0
0.8
0.6
0.4
0.2
0.0

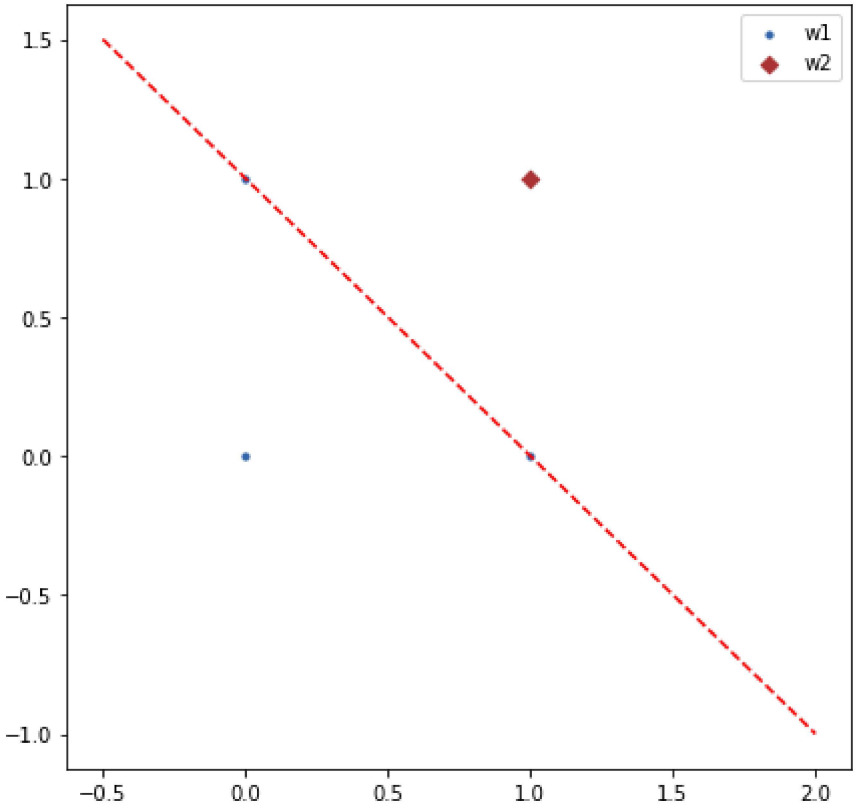

0.0
0.2
0.4
0.6
0.8
1.0

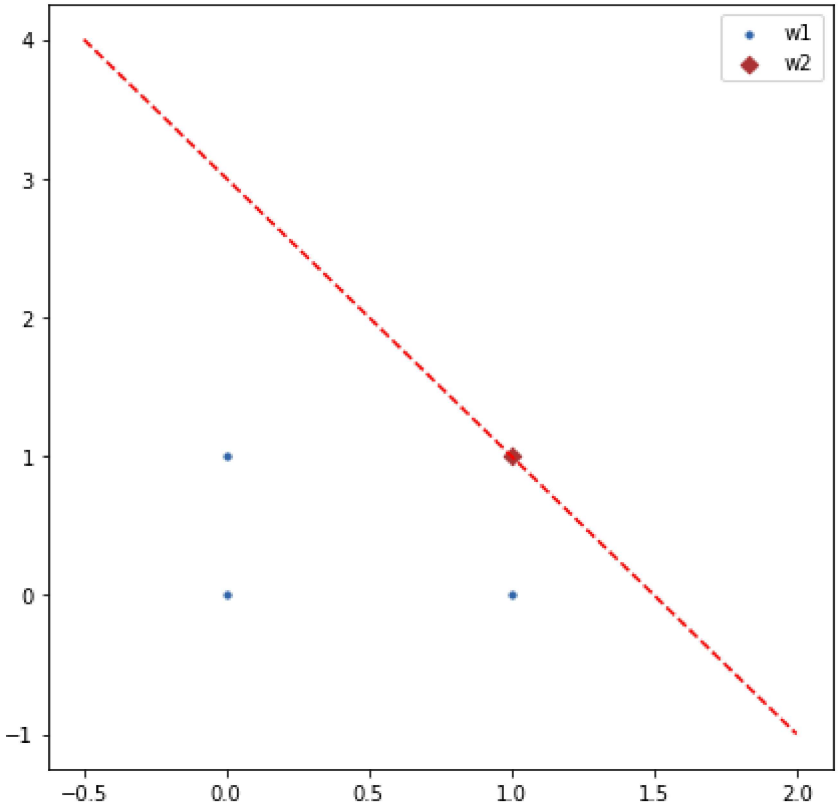

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
w1
w2

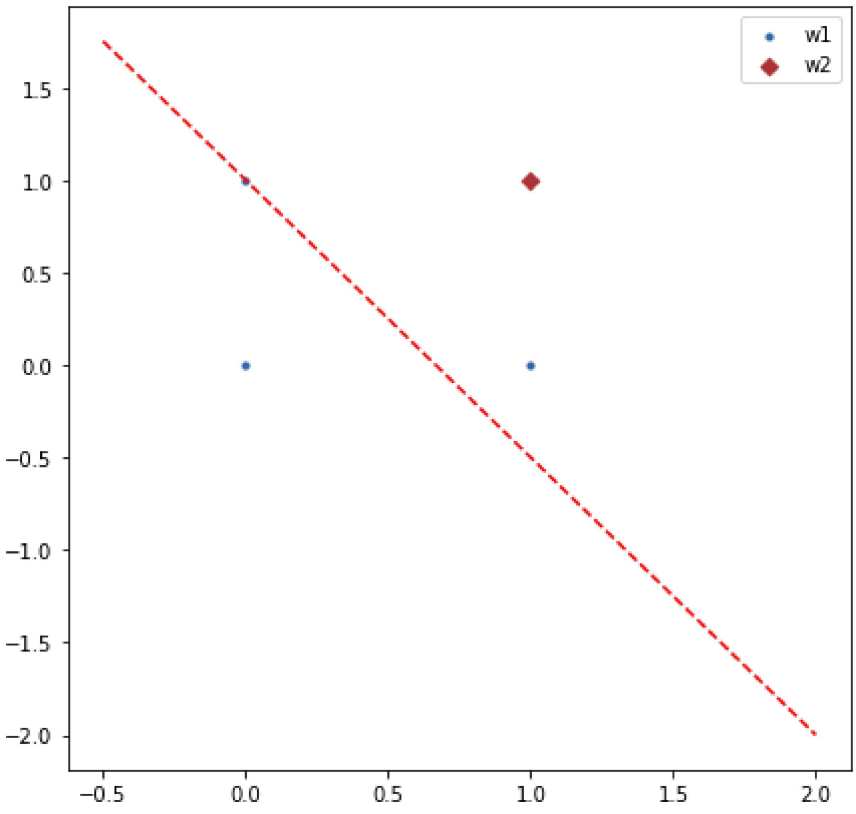

3
2
1
0
−1
−2
−0.5
0.0
0.5
1.0
1.5
2.0
w1
w2

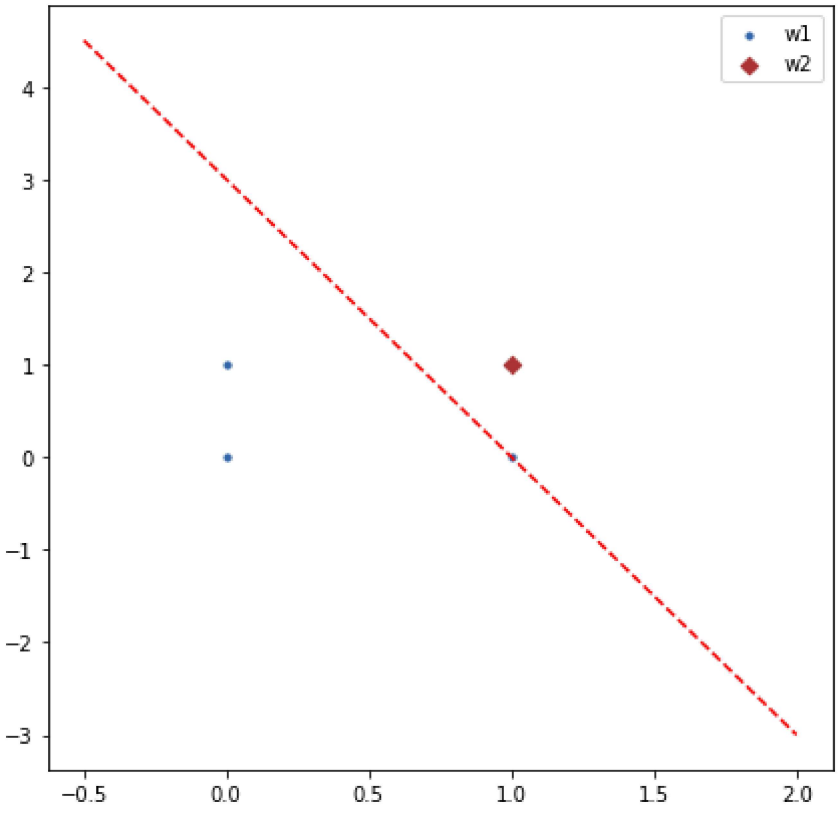

3.5
3.0
2.5
2.0
w1
w2

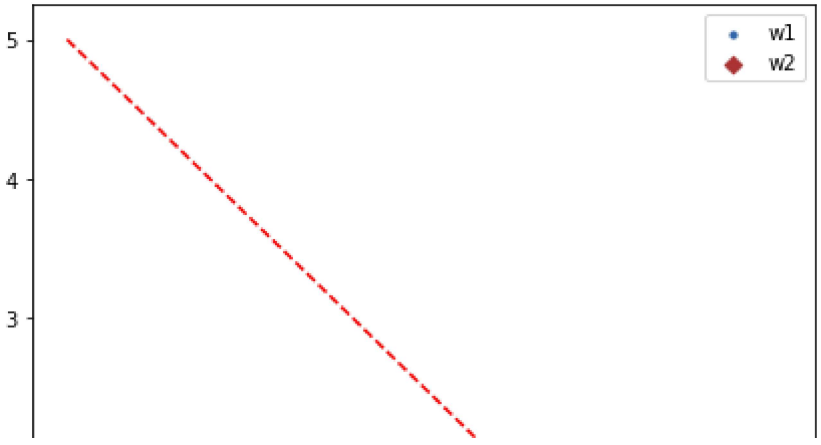

1.5
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0

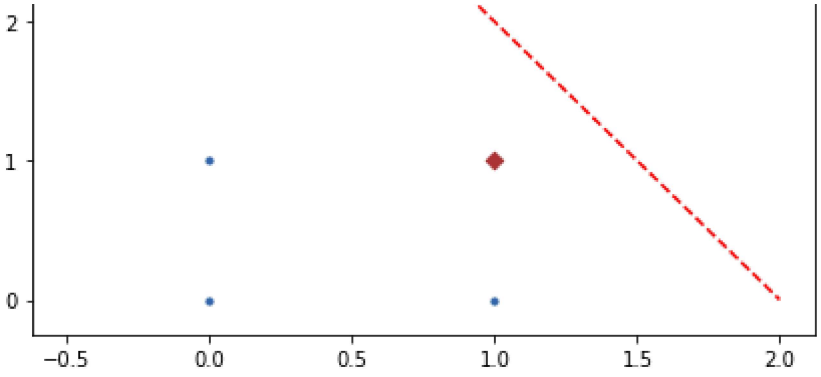

w1
w2
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

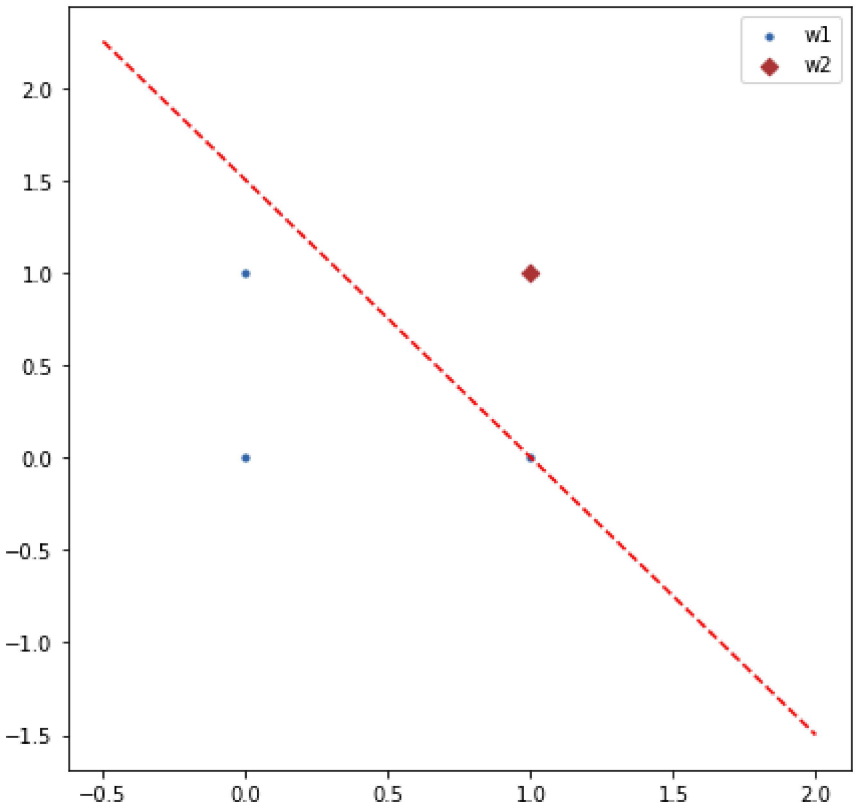

w1
w2
4
3
2
1
0
−1
−0.5
0.0
0.5
1.0
1.5
2.0

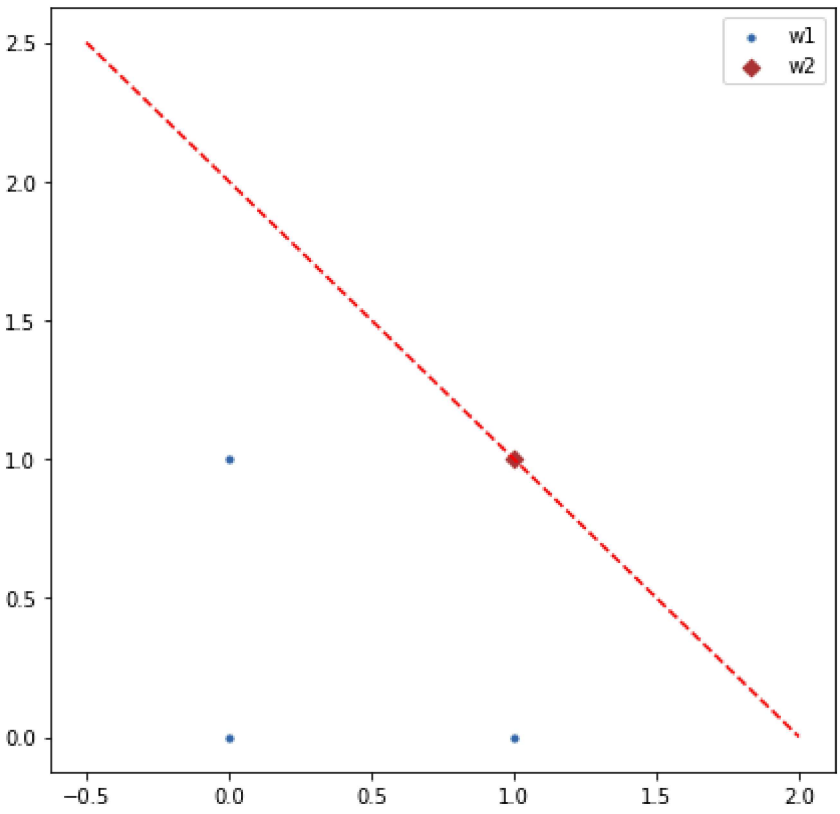

w1
w2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−0.5
0.0
0.5
1.0
1.5
2.0


w1
w2
4
3
2
1
0
−1
−2
−3
−0.5
0.0
0.5
1.0
1.5
2.0

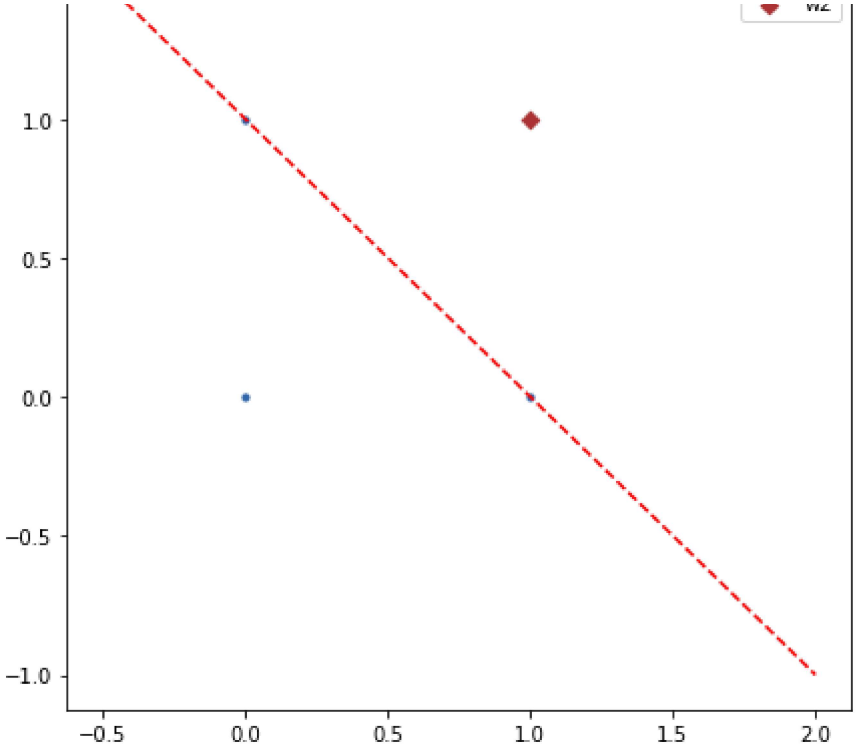

w1
w2
5
4
3

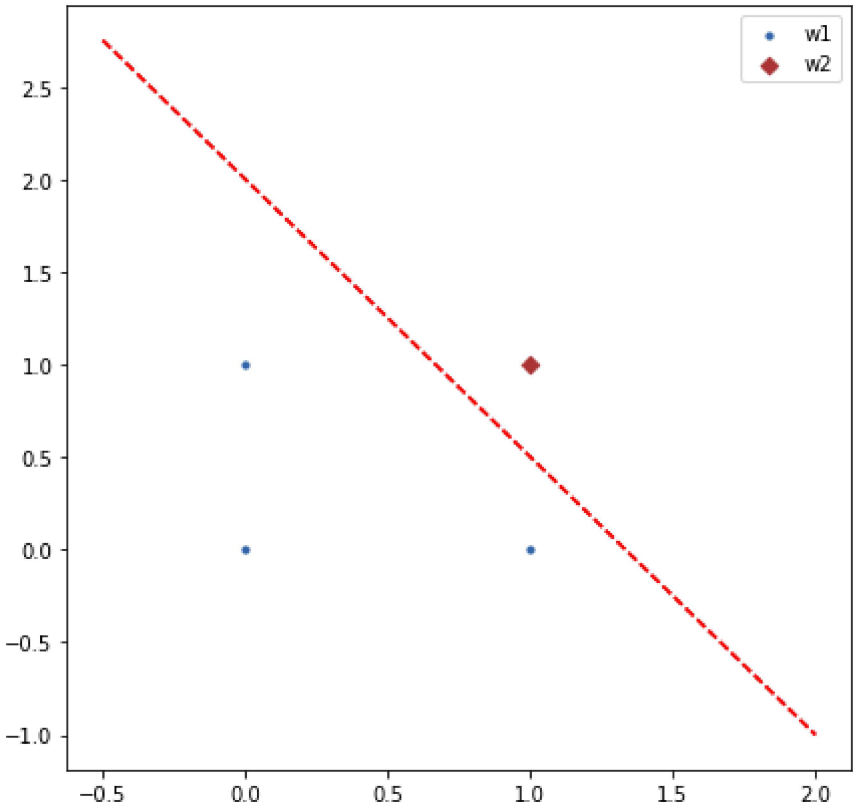

2
1
0
−0.5
0.0
0.5
1.0
1.5
2.0


w1
w2
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−0.5
0.0
0.5
1.0
1.5
2.0


w1
w2
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0

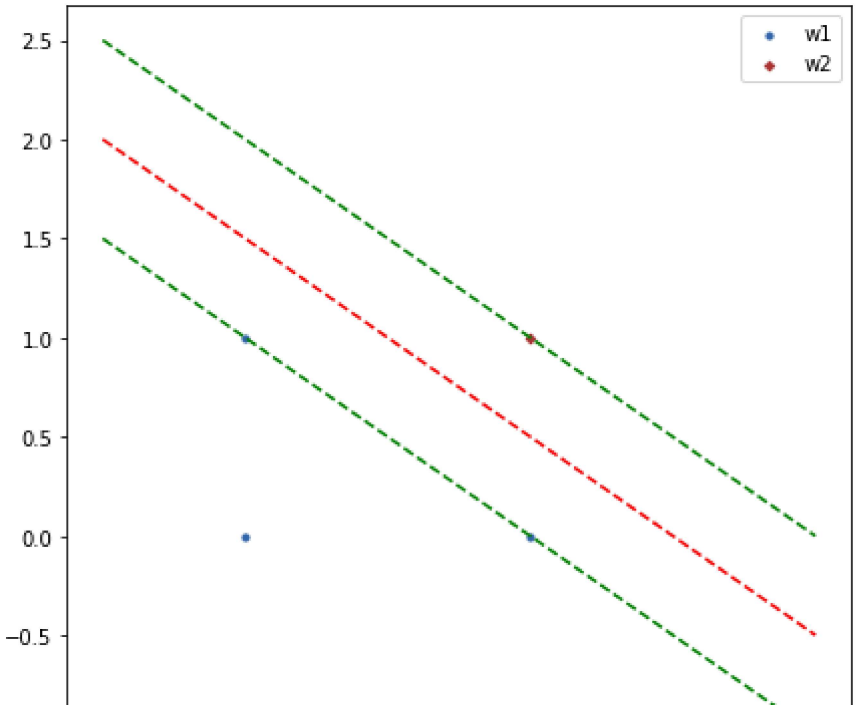

w1
1.5






```
data :  [[0 0]
 [1 0]
 [0 1]
 [1 1]]
     fun: -3.999999999999984
     jac: array([-1.        , -2.99999976, -2.99999976,  2.99999988])
 message: 'Optimization terminated successfully.'
    nfev: 30
     nit: 5
    njev: 5
  status: 0
 success: True
       x: array([0.        , 1.99999986, 1.99999986, 3.99999972])
lagrange multiplier are :  [0.         1.99999986 1.99999986 3.99999972]
bias is  2.9999997882406007
```

```
weight vector is :  [-1.99999986 -1.99999986]
```



Question 2

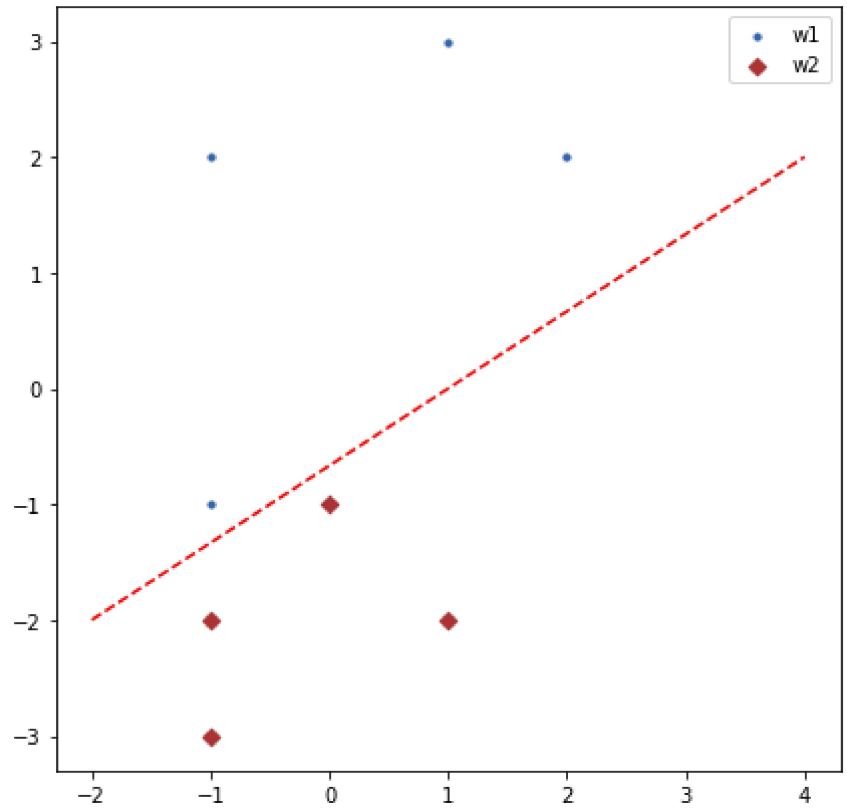
```
# import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# import pandas as pd
```

```
def perceptron():
    w1 = np.array([[1, 2, 2], [1, -1, 2], [1, 1, 3], [1, -1, -1]])
    w2 = np.array([[1, -1, -3], [1, 0, -1], [1, 1, -2], [1, -1, -2]])
    data = np.concatenate([w1, -w2], axis = 0)
    w1 = np.array([[2, 2], [-1, 2], [1, 3], [-1, -1]])
    w2 = np.array([[-1, -3], [0, -1], [1, -2], [-1, -2]])
    total_epoch = [0, 0]
    for learning_rate in [0.01, 0.5]:
        k = 0
        Q = 0                           #criterian theta
        a = np.array([0, 0, 0])            #weights
        missclasified_vectors = True
        epoch = 1
        while((missclasified_vectors) or not(k == 0)):
            if(k == 0):
                epoch = epoch + 1
                missclasified_vectors = False
            dot_product = np.dot(a, data[k])
            if(dot_product <= 0):
                a = a + learning_rate * data[k]
                missclasified_vectors = True
```

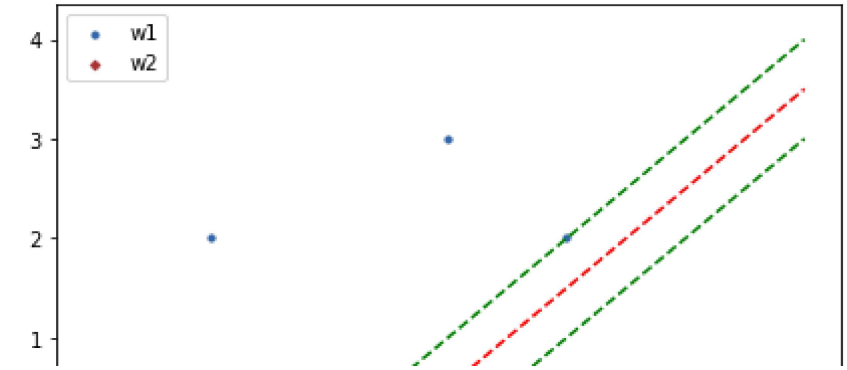
```
            k = (k + 1) % (len(data))
            if(learning_rate == 0.01):
                total_epoch[1] = epoch
            else:
                total_epoch[0] = epoch
    print("For learning rate = 0.01, epoch/ total number of iterations are : {}\nFor learr
    print("The final weight vector is : ", a)
    f, ax = plt.subplots(figsize=(7, 7))
    c1, c2 = "#3366AA", "#AA3333"
    ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
    ax.legend()
    ax.scatter(*w2.T, c=c2, marker="D", label = "w2")
    ax.legend()
    x_vec = np.linspace(-2, 4, 5)
    y_vec = ((a[1] * x_vec) + a[0])/a[2]
    y_vec = -y_vec
    plt.plot(x_vec, y_vec, 'r--')
    plt.show()

z = []
data = []
alpha = []

def func(alpha):
    alpha = np.array(alpha)
    value = np.sum(alpha)
    for i in range(len(alpha)):
        for j in range(len(alpha)):
            if(i == j):
                pass
            value -= 0.5 * (alpha[i] * alpha[j] * z[i] * z[j] * (data[i].T.dot(data[j])))
    return (-value)

def constraint(alpha):
    global z
    sum = 0
    for alpha_i, z_i in zip(alpha, z):
        sum += alpha_i * z_i
    return(sum)

def abs(num):
    if(num < 0):
        return(-num)
    else:
        return(num)

def svm():
    global z
    global data
    global alpha
    z = np.array([1, 1, 1, 1, -1, -1, -1, -1])
    w1 = np.array([[2, 2], [-1, 2], [1, 3], [-1, -1]])
    w2 = np.array([[-1, -3], [0, -1], [1, -2], [-1, -2]])
    data = np.concatenate([w1, w2], axis = 0)
    print("data : ", data)
```

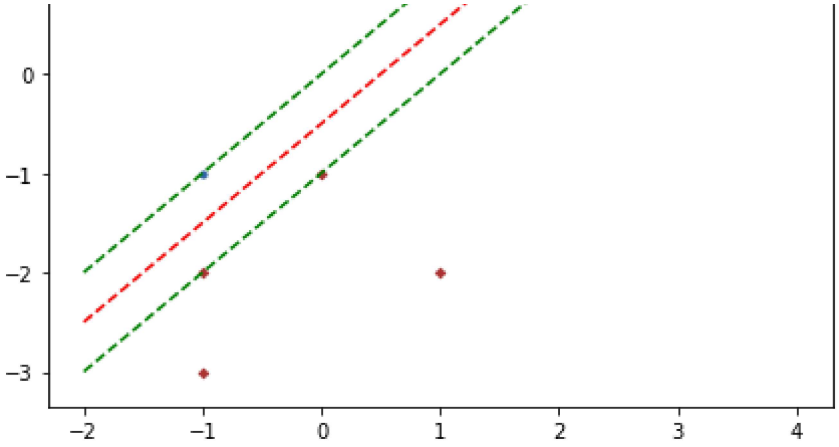
```
    max_solution = 0
    consrt = {'type' : 'eq', 'fun' : constraint}
    alpha = np.zeros(len(data))
    b = (0.0000000000, None)
    bound = [b, b, b, b, b, b, b, b]
    sol = minimize(func, alpha, method = 'SLSQP', constraints = consrt, bounds = bound)
    print(sol)
    alpha = sol.x

    print("lagrange multiplier are : ", alpha)

    a = np.zeros(len(data[0]))
    for i in range(len(data)):
        a += alpha[i]*z[i]*data[i]

    bias = []                                          #finding the points that pass throu
    for i in range(len(alpha)):
        for j in range(len(alpha)):
            if(i == j):
                pass
            else:
                if(abs(0.5 * (alpha[i] * alpha[j] * z[i] * z[j] * (data[i].T.dot(data[j]))
                    bias.append(1/z[i] - a.T.dot(data[i]))

    bias = np.array(bias).mean()
    print("bias is ", bias)
    print("weight vector is : ", a)
    a = [bias, a[0] , a[1]]
    f, ax = plt.subplots(figsize=(7, 7))
    c1, c2 = "#3366AA", "#AA3333"

    x_vec = np.linspace(-2, 4, 5)
    y_vec = ((a[1] * x_vec) + a[0])/a[2]
    y_vec = -y_vec
    plt.plot(x_vec, y_vec, 'r--')
    y_vec = y_vec + (1 / a[2])
    plt.plot(x_vec, y_vec, 'g--')
    y_vec = y_vec - (2/ a[2])
    plt.plot(x_vec, y_vec, 'g--')

    ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
    ax.legend()
    ax.scatter(*w2.T, c=c2,s = 10, marker="D", label = "w2")
    ax.legend()

    plt.show()


if(__name__ == '__main__'):
    perceptron()
    svm()
```

```
For learning rate = 0.01, epoch/ total number of iterations are : 7
For learning rate = 0.5, epoch/ total number of iterations are : 7
The final weight vector is :  [ 1.  -1.   1.5]
```




```
data :  [[ 2  2]
 [-1  2]
 [ 1  3]
 [-1 -1]
 [-1 -3]
 [ 0 -1]
 [ 1 -2]
 [-1 -2]]
     fun: -3.999999991725792
     jac: array([-0.9998796 ,  5.00028753,  3.0002923 , -1.00006056,  3.0002923 ,
        1.00011575,  5.00028741,  1.00017607])
 message: 'Optimization terminated successfully.'
    nfev: 72
     nit: 7
    njev: 7
  status: 0
 success: True
       x: array([2.24984385e-01, 0.00000000e+00, 0.00000000e+00, 3.77518726e+00,
       1.39428399e-16, 2.67500883e+00, 0.00000000e+00, 1.32516281e+00])
lagrange multiplier are :  [2.24984385e-01 0.00000000e+00 0.00000000e+00 3.77518726e+
 1.39428399e-16 2.67500883e+00 0.00000000e+00 1.32516281e+00]
bias is  1.0000579839668868
weight vector is :  [-2.00005568  2.00011597]
```





Question 3

```
import cv2
import numpy as np
from skimage import io
import matplotlib.pyplot as plt
from scipy.optimize import minimize
```

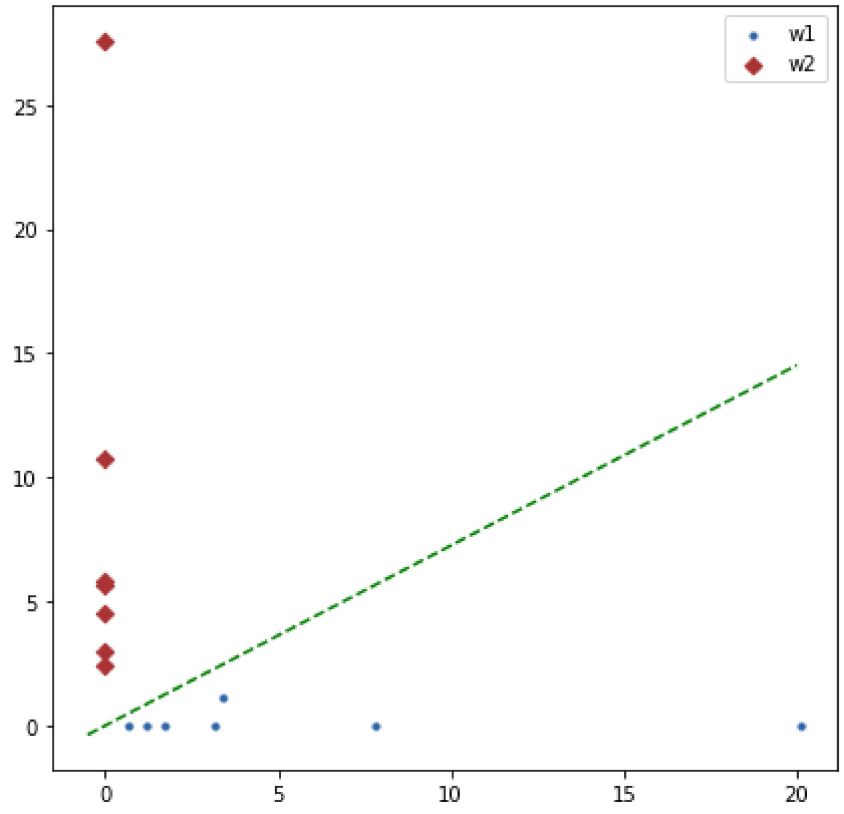
```
w1 = []
w2 = []

def perceptron():
    global w1
    global w2
    data = []
    for i in range(1, 15):
        img = cv2.imread('poly{}.png'.format(i))
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY)
        img[thresh == 255] = 0                                                    #converting bl
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
        myimg = cv2.erode(img, kernel, iterations = 1)
        avg_color_per_row = np.average(myimg, axis=0)
        avg_color = np.average(avg_color_per_row, axis=0)
        if(i < 8):
            w1.append([avg_color[1], avg_color[2]])
            data.append([1, avg_color[1], avg_color[2]])
        else:
            w2.append([avg_color[1], avg_color[2]])
            data.append([-1, -avg_color[1], -avg_color[2]])
        # print(avg_color)
    print("The features chosen are avg red and avg green pixel value over the image.")
    w1 = np.array(w1)
    w2 = np.array(w2)
    data = np.array(data)
    print("w1 is : \n", w1)
    print("w2 is : \n", w2)
    learning_rate = 0.01
    k = 0
    Q = 0                              #criterian theta
    a = np.array([0, 0, 0])            #weights
```

```
        a = np.array([0, 0, 0])             #weights
        missclasified_vectors = True
        epoch = 1
        while(((missclasified_vectors) or (not(k == 0))) and (epoch < 10)):
            if(k == 0):
                epoch = epoch + 1
                missclasified_vectors = False
            dot_product = np.dot(a, data[k])
            if(dot_product <= 0):
                a = a + learning_rate * data[k]
                missclasified_vectors = True
            k = (k + 1) % (len(data))
        print("weights are : ", a)
        f, ax = plt.subplots(figsize=(7, 7))
        c1, c2, c3 = "#3366AA", "#AA3333", 'm'
        ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
        ax.legend()
        ax.scatter(*w2.T, c=c2, marker="D", label = "w2")
        ax.legend()

        x_vec = np.linspace(-0.5, 20, 5)
        y_vec = ((a[1] * x_vec) + a[0])/a[2]
        y_vec = -y_vec
        plt.plot(x_vec, y_vec, 'g--')
        plt.show()

    z = []
    data = []
    alpha = []

    def func(alpha):
        alpha = np.array(alpha)
        value = np.sum(alpha)
        for i in range(len(alpha)):
            for j in range(len(alpha)):
                if(i == j):
                    pass
                value -= 0.5 * (alpha[i] * alpha[j] * z[i] * z[j] * (data[i].T.dot(data[j])))
        return (-value)

    def constraint(alpha):
        global z
        sum = 0
        for alpha_i, z_i in zip(alpha, z):
            sum += alpha_i * z_i
        return(sum)

    def abs(num):
        if(num < 0):
            return(-num)
        else:
            return(num)

    def svm():
        global z
        global data
```

```
    global data
    global alpha
    global w1
    global w2
    z = np.array([1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1])
    data = np.concatenate([w1, w2], axis = 0)
    print("data : ", data)
    max_solution = 0
    consrt = {'type' : 'eq', 'fun' : constraint}
    alpha = np.zeros(len(data))
    b = (0.0000000000, None)
    bound = [b, b, b, b, b, b, b, b, b, b, b, b, b, b]
    sol = minimize(func, alpha, method = 'SLSQP', constraints = consrt, bounds = bound)
    print(sol)
    alpha = sol.x

    print("lagrange multiplier are : ", alpha)

    a = np.zeros(len(data[0]))
    for i in range(len(data)):
        a += alpha[i]*z[i]*data[i]

    bias = []                                          #finding the points that pass throu
    for i in range(len(alpha)):
        for j in range(len(alpha)):
            if(i == j):
                pass
            else:
                if(abs(0.5 * (alpha[i] * alpha[j] * z[i] * z[j] * (data[i].T.dot(data[j]))
                    bias.append(1/z[i] - a.T.dot(data[i]))

    bias = np.array(bias).mean()
    print("bias is ", bias)
    print("weight vector is : ", a)
    a = [bias, a[0] , a[1]]
    f, ax = plt.subplots(figsize=(7, 7))
    c1, c2 = "#3366AA", "#AA3333"

    x_vec = np.linspace(-0.5, 20, 5)
    y_vec = ((a[1] * x_vec) + a[0])/a[2]
    y_vec = -y_vec
    plt.plot(x_vec, y_vec, 'r--')
    y_vec = y_vec + (1 / a[2])
    plt.plot(x_vec, y_vec, 'g--')
    y_vec = y_vec - (2/ a[2])
    plt.plot(x_vec, y_vec, 'g--')

    ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
    ax.legend()
    ax.scatter(*w2.T, c=c2,s = 10, marker="D", label = "w2")
    ax.legend()

    plt.show()


if( name   == " main "):
```

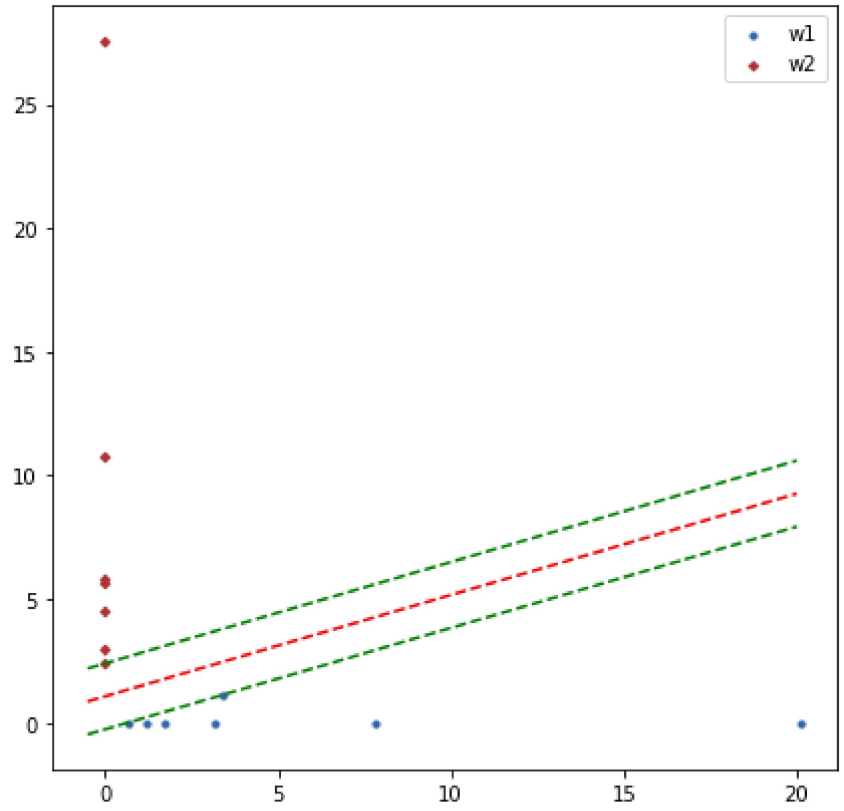
```
if(__name__ == '__main__'):
    perceptron()
    svm()
```

```
The features chosen are avg red and avg green pixel value over the image.
w1 is :
 [[ 7.80612245  0.        ]
 [20.10159196  0.        ]
 [ 1.70093202  0.        ]
 [ 0.65652412  0.        ]
 [ 1.20765129  0.        ]
 [ 3.17801339  0.        ]
 [ 3.40327803  1.12626467]]
w2 is :
 [[ 0.         10.75609756]
 [ 0.          5.85021521]
 [ 0.          5.64009662]
 [ 0.          2.965026  ]
 [ 0.         27.59203958]
 [ 0.          2.39705882]
 [ 0.          4.55492424]]
weights are :  [ 0.          0.07806122 -0.10756098]
```




```
data :  [[ 7.80612245  0.        ]
 [20.10159196  0.        ]
 [ 1.70093202  0.        ]
 [ 0.65652412  0.        ]
 [ 1.20765129  0.        ]
 [ 3.17801339  0.        ]
 [ 3.40327803  1.12626467]
 [ 0.         10.75609756]
 [ 0.          5.85021521]
 [ 0.          5.64009662]
 [ 0.          2.965026  ]
 [ 0.         27.59203958]
 [ 0.          2.39705882]
 [ 0.          4.55492424]]
     fun: -0.3286372249072372
     jac: array([ 1.40095793,  5.18272276, -0.47683813, -0.79807049, -0.6285583 ,
       -0.02252679, -0.79806962,  7.0683139 ,  3.38833602,  3.23072284,
        1.22411136, 19.69721592,  0.79807048,  2.41671839])
 message: 'Optimization terminated successfully.'
```

```
      nfev: 116
       nit: 7
      njev: 7
    status: 0
   success: True
         x: array([0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.95210364e-01,
        0.00000000e+00, 0.00000000e+00, 3.34268643e-02, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.76848680e-14,
        3.28637228e-01, 0.00000000e+00])
lagrange multiplier are :  [0.00000000e+00 0.00000000e+00 0.00000000e+00 2.95210364e-
 0.00000000e+00 0.00000000e+00 3.34268643e-02 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 2.76848680e-14
 3.28637228e-01 0.00000000e+00]
bias is  0.7980700917977757
weight vector is :  [ 0.30757364 -0.75011527]
```



Question 4

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import minimize


w1 = []
w2 = []
w3 = []
setosa = []
virginica = []
versicolor = []

def data_processing():
    global w1
    global w2
```

```
        global w2
        global w3
        global setosa
        global virginica
        global versicolor
        data = pd.read_csv("iris.csv")
        print(data.head())
        # print(data.loc('51'))
        column = ['petal.length', 'sepal.width']
        setosa = data[column][0 : 50].copy()
        versicolor = data[column][50 : 100].copy()
        virginica = data[column][100 : 150].copy()
        print("Setosa overview : \n", setosa.head())
        print("Versicolor overview : \n", versicolor.head())
        print("Virginica overview : \n", virginica.head())
        w1 = []
        for index, row in setosa.iterrows():
            w1.append([row[0], row[1]])
        w1 = np.array(w1)

        w2 = []
        for index, row in versicolor.iterrows():
            w2.append([row[0], row[1]])
        w2 = np.array(w2)

        w3 = []
        for index, row in virginica.iterrows():
            w3.append([row[0], row[1]])
        w3 = np.array(w3)



    def perceptron():
        global w1
        global w2
        global w3
        global setosa
        global virginica
        global versicolor

        # print(data.loc('51'))
        f, ax = plt.subplots(figsize=(7, 7))
        c1, c2, c3 = "#3366AA", "#AA3333", 'm'

        ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
        ax.legend()
        ax.scatter(*w2.T, c=c2, marker="D", label = "w2")
        ax.legend()
        ax.scatter(*w3.T, c=c3, marker="X", label = "w2")
        ax.legend()



        data1 = []
        for i in range(len(w1)):
            data1.append([1, w1[i][0], w1[i][1]])
```

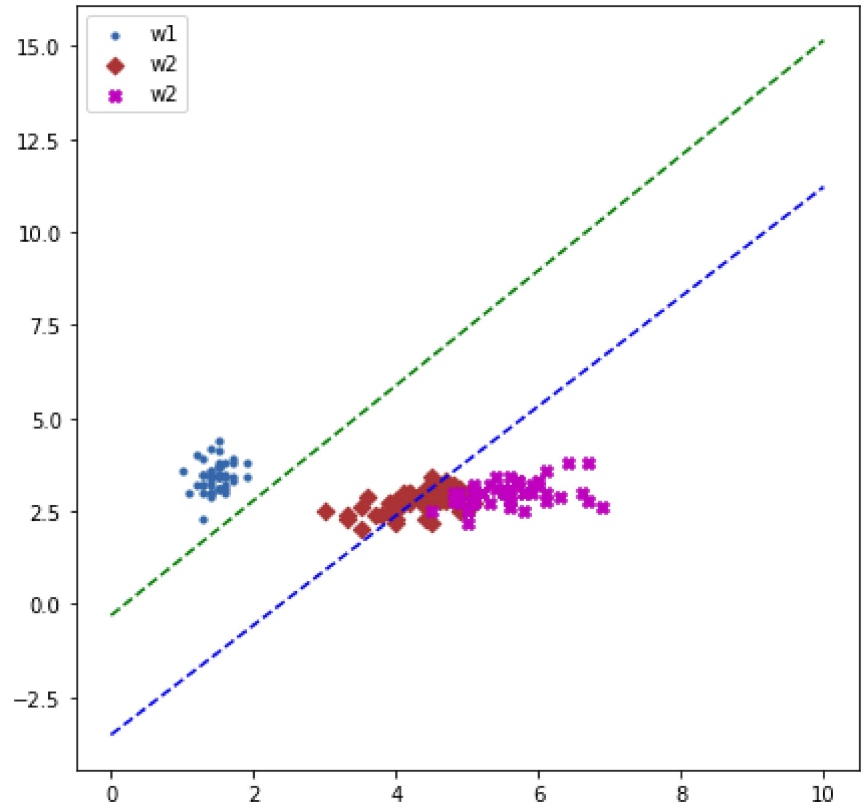
```
    data1.append([-1, -w2[i][0], -w2[i][1]])
    data1.append([-1, -w3[i][0], -w3[i][1]])
data1 = np.array(data1)

data2 = []
for i in range(len(w2)):
    data2.append([1, w2[i][0], w2[i][1]])
    data2.append([-1, -w3[i][0], -w3[i][1]])
    data2.append([-1, -w1[i][0], -w1[i][1]])
data2 = np.array(data2)

data3 = []
for i in range(len(w3)):
    data3.append([1, w3[i][0], w3[i][1]])
    data3.append([-1, -w1[i][0], -w1[i][1]])
    data3.append([-1, -w2[i][0], -w2[i][1]])
data3 = np.array(data3)

learning_rate = 0.01
k = 0
Q = 0                           #criterian theta
a = np.array([0, 0, 0])            #weights
missclasified_vectors = True
epoch = 1
print("Classifying Setosa from the rest .....")
while(((missclasified_vectors) or (not(k == 0))) and (epoch < 10)):
    if(k == 0):
        epoch = epoch + 1
        missclasified_vectors = False
    dot_product = np.dot(a, data1[k])
    if(dot_product <= 0):
        a = a + learning_rate * data1[k]
        missclasified_vectors = True
    k = (k + 1) % (len(data1))
print("weights for classifying setosa and the rest are : ", a)
x_vec = np.linspace(0, 10, 5)
y_vec = ((a[1] * x_vec) + a[0])/a[2]
y_vec = -y_vec
plt.plot(x_vec, y_vec, 'g--')



# a = np.array([0, 0, 0])            #weights
# missclasified_vectors = True
# epoch = 1
# while(((missclasified_vectors) or (not(k == 0))) and (epoch < 100)):
#     if(k == 0):
#         print("\n\n\nEpoch : ", epoch)
#         epoch = epoch + 1
#         missclasified_vectors = False
#     print("Data index : ", k)
#     print("weights : ", a)
#     print("Data point : ", np.array(data2[k]))
#     dot_product = np.dot(a, data2[k])
#     print("The dot product is : ", dot_product)
```

```
        #     if(dot_product <= 0):
        #         a = a + learning_rate * data2[k]
        #         missclasified_vectors = True
        #     k = (k + 1) % (len(data1))
        # print("weights are : ", a)
        # x_vec = np.linspace(0, 10, 5)
        # y_vec = ((a[1] * x_vec) + a[0])/a[2]
        # y_vec = -y_vec
        # # plt.plot(x_vec, y_vec, 'r--')

        a = np.array([0, 0, 0])            #weights
        missclasified_vectors = True
        epoch = 1
        print("Classifying Virginica from the rest ....")
        while(((missclasified_vectors) or (not(k == 0))) and (epoch < 100)):
            if(k == 0):
                epoch = epoch + 1
                missclasified_vectors = False
            dot_product = np.dot(a, data3[k])
            if(dot_product <= 0):
                a = a + learning_rate * data3[k]
                missclasified_vectors = True
            k = (k + 1) % (len(data1))
        print("weights for classifying virginica are : ", a)
        x_vec = np.linspace(0, 10, 5)
        y_vec = ((a[1] * x_vec) + a[0])/a[2]
        y_vec = -y_vec
        plt.plot(x_vec, y_vec, 'b--')


        plt.show()

    z = []
    data = []
    alpha = []
    c = 10
    # def non_linearly_seperable_func(alpha):
    #     global c
    #     value = -func(alpha)
    #     for i in range(len(alpha)):
    #         value -= (1/(2 * c)) * (alpha[i] ** 2)
    #     return (-value)



    def func(alpha):
        alpha = np.array(alpha)
        value = np.sum(alpha)
        for i in range(len(alpha)):
            for j in range(len(alpha)):
                if(i == j):
                    pass
                value -= 0.5 * (alpha[i] * alpha[j] * z[i] * z[j] * (data[i].T.dot(data[j])))
        return (-value)
```

```
    def constraint(alpha):
        global z
        sum = 0
        for alpha_i, z_i in zip(alpha, z):
            sum += alpha_i * z_i
        return(sum)

    def abs(num):
        if(num < 0):
            return(-num)
        else:
            return(num)

    def svm():
        global z
        global alpha
        global w1
        global w2
        global data
        z = np.concatenate([np.ones(50), -np.ones(100)], axis = 0)
        data1 = np.concatenate([w1, w2, w3], axis = 0)
        data2 = np.concatenate([w3, w1, w2])
        # print("data : ", data1)
        max_solution = 0
        f, ax = plt.subplots(figsize=(7, 7))
        c1, c2 = "#3366AA", "#AA3333"
        for j in range(2):
            print("Computing the {}th svm for the data ... ".format(j + 1))
            if(j == 0):
                data = data1
            else:
                data = data2
            consrt = {'type' : 'eq', 'fun' : constraint}
            alpha = np.zeros(len(data))
            # print(len(alpha))
            if(j == 0):
                c = None
            else:
                c = 100
            b = (0.0000000000, c)
            bound = [b] * len(data)
            sol = minimize(func, alpha, method = 'SLSQP', constraints = consrt, bounds = bounc
            print(sol)
            alpha = sol.x

            print("lagrange multiplier are : ", alpha)

            a = np.zeros(len(data[0]))
            for i in range(len(data)):
                a += alpha[i]*z[i]*data[i]
            if(j == 1):
                error = alpha / c
            else:
                error = np.zeros(len(alpha))
            bias = []                                                    #finding the points that pass t
```

```
        for i in range(len(alpha)):
            for k in range(len(alpha)):
                if(i == k):
                    pass
                else:
                    if(abs(0.5 * (alpha[i] * alpha[k] * z[i] * z[k] * (data[i].T.dot(data[
                        bias.append((1 - error[i])/z[i] - a.T.dot(data[i]))

        bias = np.array(bias).mean()
        print("bias is ", bias)
        print("weight vector is : ", a)
        a = [bias, a[0] , a[1]]
        if(j == 0):
            x_vec = np.linspace(1, 8, 5)
        else:
            x_vec = np.linspace(3, 8, 5)

        y_vec = ((a[1] * x_vec) + a[0])/a[2]
        y_vec = -y_vec
        if(j == 0):
            plt.plot(x_vec, y_vec, 'r--')
        else:
            plt.plot(x_vec, y_vec, 'b--')
        y_vec = y_vec + (1 / a[2])
        plt.plot(x_vec, y_vec, 'g--')
        y_vec = y_vec - (2/ a[2])
        plt.plot(x_vec, y_vec, 'g--')

    ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
    ax.legend()
    ax.scatter(*w2.T, c=c2,s = 10, marker="D", label = "w2")
    ax.legend()
    ax.scatter(*w3.T, c='m',s = 10, marker="x", label = "w3")
    ax.legend()

    plt.show()


#  [ 3.33329562 -0.9517713 ]


if(__name__ == "__main__"):
    data_processing()
    perceptron()
    svm()
```

```
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
Setosa overview :
    petal.length  sepal.width
0           1.4          3.5
1           1.4          3.0
2           1.3          3.2
3           1.5          3.1
4           1.4          3.6
Versicolor overview :
     petal.length  sepal.width
50           4.7          3.2
51           4.5          3.2
52           4.9          3.1
53           4.0          2.3
54           4.6          2.8
Virginica overview :
      petal.length  sepal.width
100           6.0          3.3
101           5.1          2.7
102           5.9          3.0
103           5.6          2.9
104           5.8          3.0
Classifying Setosa from the rest .....
weights for classifying setosa and the rest are :  [ 0.01  -0.051  0.033]
Classifying Virginica from the rest ....
weights for classifying virginica are :  [-1.33   0.558 -0.379]
```



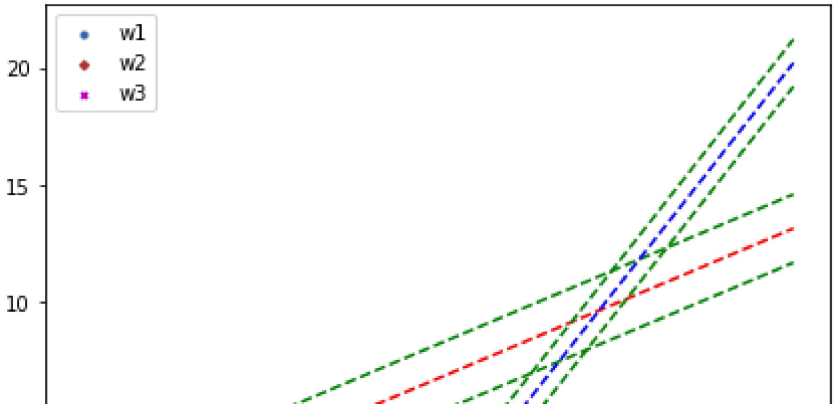

```
Computing the 1th svm for the data ...
     fun: -1.025306102404832
     jac: array([-0.36003101, -0.70292151, -0.44003224, -0.76007664, -0.29145312,
       -0.46291816, -0.42860913, -0.55434239, -0.77149963, -0.76007664,
       -0.34860814, -0.68007553, -0.70292151, -0.32572198,  0.23432565,
        0.13143861,  0.04001439, -0.36003101, -0.53149629, -0.28002989,
```

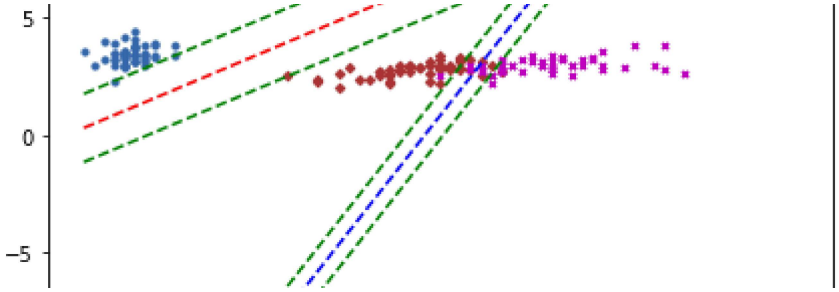
```
       -0.80580866, -0.34860802,  0.21147978, -0.87438679, -1.05727506,
       -0.95438778, -0.68007553, -0.48576427, -0.42860913, -0.81723166,
       -0.88580978, -0.55434239, -0.07429552,  0.12001562, -0.76007664,
       -0.31429899, -0.23429787, -0.291453  , -0.57718837, -0.55434239,
       -0.23429787, -1.05723512, -0.44003224, -0.61149728, -0.78296256,
       -0.70292151, -0.40576303, -0.56576526, -0.34860802, -0.49718738,
        2.71496022,  2.46349382,  3.03500462,  2.45203078,  2.86353934,
        2.73780632,  2.64638221,  1.50332057,  2.79496133,  2.05198526,
        2.02909923,  2.22345054,  2.5206089 ,  2.92069447,  1.53762949,
        2.40633869,  2.60065019,  2.30345166,  3.14927471,  2.18914139,
        2.84069335,  2.1091404 ,  3.44647312,  2.98927248,  2.4177618 ,
        2.47491682,  3.11500585,  3.229316  ,  2.66922808,  1.6176306 ,
        2.13198638,  2.00625324,  2.05198526,  3.56078327,  2.60065007,
        2.3263377 ,  2.78353834,  2.95496345,  2.09771729,  2.31487453,
        2.74922919,  2.72638321,  2.24629653,  1.57189858,  2.42918479,
        2.22345054,  2.29202855,  2.4177618 ,  1.05754292,  2.23487344,
        4.2809134 ,  3.56078337,  4.3609145 ,  4.05229303,  4.23518135,
        5.24104673,  2.94354042,  4.93242531,  4.57807168,  4.20091231,
        3.21789296,  3.81224968,  3.85798185,  3.57220629,  3.49220522,
        3.46935931,  3.85798183,  4.81815524,  5.89255854,  3.77794065,
        3.97229196,  3.24073896,  5.50393602,  3.3093169 ,  3.90371388,
        4.34949146,  3.11500573,  3.10358271,  4.12087111,  4.23518136,
        4.74953701,  4.44095571,  4.12087111,  3.49220522,  4.25802734,
        4.61238085,  3.70940258,  3.78940366,  2.97784957,  3.66367051,
        3.9151369 ,  3.28647101,  3.56078342,  4.22375834,  3.90371388,
        3.48078233,  3.57220629,  3.48078223,  3.45793632,  3.35504909])
message: 'Optimization terminated successfully.'
   nfev: 1219
    nit: 8
   njev: 8
 status: 0
success: True
      x: array([0.00000000e+00, 3.04543116e-15, 1.60848824e-15, 6.33990373e-16,
       1.14193421e-15, 4.11881815e-15, 7.01894223e-16, 1.72534032e-15,
       3.57064972e-15, 2.28507291e-15, 1.49198134e-16, 4.76229221e-15,
       2.67677713e-15, 0.00000000e+00, 7.07892049e-16, 0.00000000e+00,
       1.09490911e-16, 2.73206916e-15, 0.00000000e+00, 5.22449952e-17,
       0.00000000e+00, 6.02977814e-16, 1.20205645e-15, 0.00000000e+00,
       8.09882209e-01, 5.07681988e-15, 3.77650884e-15, 0.00000000e+00,
       3.02114074e-16, 2.08587007e-15, 0.00000000e+00, 5.27363589e-15,
       1.26480465e-15, 4.02222717e-15, 2.73289824e-15, 0.00000000e+00,
       0.00000000e+00, 1.22524569e-15, 2.62848359e-15, 0.00000000e+00,
       5.24938120e-16, 2.15565435e-01, 2.26451247e-15, 3.48358641e-15,
       0.00000000e+00, 1.21740638e-15, 1.53468273e-15, 1.01198516e-15,
       2.23422044e-15, 8.61836505e-16, 6.37810860e-16, 7.42297550e-16,
       0.00000000e+00, 5.48363794e-15, 1.80361888e-15, 1.70048854e-15,
       1.67181084e-15, 0.00000000e+00, 1.68873423e-15, 6.03770597e-15,
       4.75839314e-15, 6.02992738e-15, 3.05166540e-15, 3.23582712e-15,
       6.81740645e-15, 2.51267440e-15, 1.20571932e-15, 4.19018642e-15,
       0.00000000e+00, 5.43237165e-15, 1.71786356e-15, 4.04991209e-15,
       0.00000000e+00, 1.30479506e-16, 3.96596429e-15, 3.43309918e-15,
       2.14768662e-15, 0.00000000e+00, 2.16293820e-15, 6.04339305e-15,
       4.87160763e-15, 5.13487477e-15, 5.35254902e-15, 0.00000000e+00,
       2.80738175e-15, 2.83674835e-15, 2.93010837e-15, 4.08186857e-15,
       4.29391959e-15, 4.34280005e-15, 2.82693748e-15, 3.19553611e-15,
       2.64147156e-15, 0.00000000e+00, 3.03520718e-15, 5.33244357e-15,
       3.14002658e-15, 4.81026263e-15, 1.02544764e+00, 4.01488086e-15,
       0.00000000e+00, 4.70472977e-15, 0.00000000e+00, 0.00000000e+00,
       8.73344405e-16, 1.39226687e-15, 3.31546489e-15, 0.00000000e+00,
       3.44386147e-16, 0.00000000e+00, 7.67306463e-16, 4.03417462e-17,
       0.00000000e+00, 0.00000000e+00, 2.38810991e-15, 0.00000000e+00,
       2.27849166e-16, 0.00000000e+00, 0.00000000e+00, 2.41482337e-15,
```

```
        2.27849166e-16, 0.00000000e+00, 0.00000000e+00, 2.41482337e-15,
        0.00000000e+00, 1.88762748e-15, 0.00000000e+00, 7.37318598e-17,
        0.00000000e+00, 0.00000000e+00, 2.69904919e-15, 1.58233956e-15,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 7.93230001e-16, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 1.53082604e-15, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        2.71385486e-16, 1.12354608e-15, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00])
lagrange multiplier are :  [0.00000000e+00 3.04543116e-15 1.60848824e-15 6.33990373e-
 1.14193421e-15 4.11881815e-15 7.01894223e-16 1.72534032e-15
 3.57064972e-15 2.28507291e-15 1.49198134e-16 4.76229221e-15
 2.67677713e-15 0.00000000e+00 7.07892049e-16 0.00000000e+00
 1.09490911e-16 2.73206916e-15 0.00000000e+00 5.22449952e-17
 0.00000000e+00 6.02977814e-16 1.20205645e-15 0.00000000e+00
 8.09882209e-01 5.07681988e-15 3.77650884e-15 0.00000000e+00
 3.02114074e-16 2.08587007e-15 0.00000000e+00 5.27363589e-15
 1.26480465e-15 4.02222717e-15 2.73289824e-15 0.00000000e+00
 0.00000000e+00 1.22524569e-15 2.62848359e-15 0.00000000e+00
 5.24938120e-16 2.15565435e-01 2.26451247e-15 3.48358641e-15
 0.00000000e+00 1.21740638e-15 1.53468273e-15 1.01198516e-15
 2.23422044e-15 8.61836505e-16 6.37810860e-16 7.42297550e-16
 0.00000000e+00 5.48363794e-15 1.80361888e-15 1.70048854e-15
 1.67181084e-15 0.00000000e+00 1.68873423e-15 6.03770597e-15
 4.75839314e-15 6.02992738e-15 3.05166540e-15 3.23582712e-15
 6.81740645e-15 2.51267440e-15 1.20571932e-15 4.19018642e-15
 0.00000000e+00 5.43237165e-15 1.71786356e-15 4.04991209e-15
 0.00000000e+00 1.30479506e-16 3.96596429e-15 3.43309918e-15
 2.14768662e-15 0.00000000e+00 2.16293820e-15 6.04339305e-15
 4.87160763e-15 5.13487477e-15 5.35254902e-15 0.00000000e+00
 2.80738175e-15 2.83674835e-15 2.93010837e-15 4.08186857e-15
 4.29391959e-15 4.34280005e-15 2.82693748e-15 3.19553611e-15
 2.64147156e-15 0.00000000e+00 3.03520718e-15 5.33244357e-15
 3.14002658e-15 4.81026263e-15 1.02544764e+00 4.01488086e-15
 0.00000000e+00 4.70472977e-15 0.00000000e+00 0.00000000e+00
 8.73344405e-16 1.39226687e-15 3.31546489e-15 0.00000000e+00
 3.44386147e-16 0.00000000e+00 7.67306463e-16 4.03417462e-17
 0.00000000e+00 0.00000000e+00 2.38810991e-15 0.00000000e+00
 2.27849166e-16 0.00000000e+00 0.00000000e+00 2.41482337e-15
 0.00000000e+00 1.88762748e-15 0.00000000e+00 7.37318598e-17
 0.00000000e+00 0.00000000e+00 2.69904919e-15 1.58233956e-15
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 7.93230001e-16 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 1.53082604e-15 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 2.71385486e-16 1.12354608e-15 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00]
bias is  1.0573509865503505
weight vector is :  [-1.25733167  0.6857809 ]
Computing the 2th svm for the data ...
     fun: -1568.2180312151793
     jac: array([ 28.71875   ,  24.375     ,  28.46875   ,  26.9375    ,
        27.9375    ,  32.3125    ,  21.25      ,  30.75      ,
        28.40625   ,  28.96875   ,  23.875     ,  25.46875   ,
        26.28125   ,  24.03125   ,  24.28125   ,  25.        ,
        26.28125   ,  32.0625    ,  34.375     ,  24.34375   ,
        27.1875    ,  23.21875   ,  33.09375   ,  23.3125    ,
        27.09375   ,  28.84375   ,  22.625     ,  22.9375    ,
        27.03125   ,  27.9375    ,  29.75      ,  30.4375    ,
        27.        ,  24.28125   ,  27.21875   ,  29.59375   ,
        26.4375    ,  26.1875    ,  22.4375    ,  25.625     ,
        26.71875   ,  23.96875   ,  24.375     ,  28.28125   ,
```

```
        27.09375   ,  24.625     ,  24.03125   ,  24.625     ,
        25.34375   ,  24.09375   ,  -5.1875    ,  -5.65625   ,
        -4.9375    ,  -6.125     ,  -5.09375   ,  -6.4375    ,
        -5.28125   ,  -5.8125    ,  -5.78125   ,  -6.125     ,
        -5.53125   ,  -6.375     ,  -5.65625   ,  -4.03125   ,
        -3.59375   ,  -4.84375   ,  -4.25      ,  -5.1875    ,
        -6.53125   ,  -5.4375    ,  -6.9375    ,  -5.53125   ,
        -2.90625   ,  -7.0625    ,  -8.0625    ,  -6.78125   ,
        -6.40625   ,  -5.75      ,  -5.28125   ,  -6.5625    ,
        -6.6875    ,  -5.8125    ,  -5.15625   ,  -4.5       ,
        -6.15625   ,  -4.375     ,  -4.625     ,  -5.09375   ,
        -5.15625   ,  -5.8125    ,  -4.65625   ,  -5.8125    ,
        -4.9375    ,  -6.28125   ,  -7.65625   ,  -5.65625   ,
        -6.        ,  -5.5       ,  -5.53125   ,  -5.40625   ,
       -23.625     , -22.53125   , -24.84375   , -20.65625   ,
       -23.5       , -22.90625   , -23.53125   , -16.75      ,
       -23.375     , -19.75      , -18.25      , -21.0625    ,
       -20.75      , -23.90625   , -17.90625   , -22.09375   ,
       -22.75      , -20.8125    , -23.53125   , -19.9375    ,
       -24.1875    , -20.15625   , -25.40625   , -24.03125   ,
       -21.765625  , -22.171875  , -24.578125  , -25.515625  ,
       -22.84375   , -17.65625   , -19.484375  , -18.953125  ,
       -19.765625  , -26.328125  , -22.72264099, -22.33647156,
       -23.75395203, -22.88919067, -20.5294342 , -20.50289917,
       -22.55270386, -23.2787323 , -20.39073181, -16.83444214,
       -21.39077759, -21.05430603, -21.16645813, -21.7538147 ,
       -15.0042572 , -20.75376892])
message: 'Optimization terminated successfully.'
   nfev: 7766
    nit: 51
   njev: 51
 status: 0
success: True
      x: array([7.39069483e-19, 5.50932906e-17, 3.80975736e-19, 1.93596952e-19,
       4.38747726e-19, 3.52105884e-19, 1.00000000e+02, 4.26400887e-19,
       1.11934151e-12, 1.19469715e-18, 1.00000000e+02, 1.02801530e-18,
       5.05869636e-19, 6.80013732e+00, 1.60078527e-16, 1.03878736e-10,
       4.83179503e-19, 1.27036400e-18, 1.95774205e-12, 1.13400103e-17,
       7.49574351e-19, 1.00000000e+02, 1.92082035e-12, 1.00000000e+02,
       7.73323022e-19, 5.95402536e-19, 1.00000000e+02, 1.00000000e+02,
       2.13179128e-12, 3.78261811e-19, 1.27758317e-19, 1.33317212e-18,
       9.81580150e-13, 1.47277037e-17, 5.73826515e-12, 4.15436761e-19,
       1.85542027e-19, 4.90568483e-19, 1.00000000e+02, 5.74855015e-19,
       5.09826784e-19, 6.15062550e+01, 3.90199386e-17, 5.48723162e-19,
       9.39650420e-19, 2.21098324e-12, 5.86992541e+00, 7.67262232e-19,
       3.29542786e-11, 1.77344061e+01, 6.92605978e-12, 1.29602483e-19,
       5.28925578e-12, 1.02035919e-19, 7.40055725e-12, 1.27768658e-11,
       6.37605896e-12, 3.63654677e-12, 2.98334456e-19, 8.76941595e-20,
       8.40573403e-12, 3.19770517e-12, 1.40020450e-19, 2.16490862e-19,
       1.51610478e-11, 1.96452043e-11, 1.48472769e-11, 9.82278992e-12,
       9.17877329e-12, 9.95607591e-12, 3.46583023e-12, 6.11643275e-12,
       6.32944744e-12, 3.96520866e-12, 4.64814180e-13, 1.65143241e-19,
       3.24822817e-12, 3.53138321e-12, 7.25320683e-12, 8.53940610e-20,
       5.16735781e-20, 3.67363490e-12, 1.83466767e-11, 2.08658137e-11,
       4.90371358e-13, 8.47985781e-13, 4.77893128e-12, 8.50535245e-12,
       2.52349025e-19, 5.41513001e-12, 6.36071362e-12, 1.01686701e-18,
       1.37742036e-12, 5.60548262e-12, 7.16207362e-12, 1.58991172e-13,
       9.21464222e-12, 3.02973094e-20, 6.61956068e-12, 0.00000000e+00,
       9.28809357e-09, 2.53270373e-18, 1.00000000e+02, 9.88537374e-19,
       4.72637070e-16, 1.35472457e-11, 2.37246698e-10, 1.09604538e-18,
       9.50341424e-10, 7.56094724e-19, 1.48366917e-18, 3.07615718e-19,
       0.00000000e+00, 9.19107238e+01, 4.22689566e-19, 5.62125247e-19,
```

```
       0.00000000e+00, 9.19107238e+01, 4.22689566e-19, 5.62125247e-19,
       1.86969129e-18, 7.54603510e-19, 4.93784242e-16, 9.87908779e-19,
       1.00000000e+02, 5.43651957e-19, 1.00000000e+02, 1.00000000e+02,
       4.81101291e-19, 7.35850242e-19, 1.00000000e+02, 1.00000000e+02,
       2.84643248e-11, 8.10239886e-19, 1.13361796e-18, 1.08261333e-18,
       7.55857840e-19, 1.00000000e+02, 1.82062766e-18, 2.07431718e-18,
       6.08353378e-08, 3.83329008e-12, 4.20364080e-19, 9.82319147e-19,
       1.31663673e-11, 3.27596088e-18, 8.53723526e-19, 1.23814802e-18,
       7.42126758e-19, 3.17331182e-19, 4.74904785e-19, 4.97168244e-19,
       9.35618190e-19, 7.04578538e-19])
lagrange multiplier are :  [7.39069483e-19 5.50932906e-17 3.80975736e-19 1.93596952e-
 4.38747726e-19 3.52105884e-19 1.00000000e+02 4.26400887e-19
 1.11934151e-12 1.19469715e-18 1.00000000e+02 1.02801530e-18
 5.05869636e-19 6.80013732e+00 1.60078527e-16 1.03878736e-10
 4.83179503e-19 1.27036400e-18 1.95774205e-12 1.13400103e-17
 7.49574351e-19 1.00000000e+02 1.92082035e-12 1.00000000e+02
 7.73323022e-19 5.95402536e-19 1.00000000e+02 1.00000000e+02
 2.13179128e-12 3.78261811e-19 1.27758317e-19 1.33317212e-18
 9.81580150e-13 1.47277037e-17 5.73826515e-12 4.15436761e-19
 1.85542027e-19 4.90568483e-19 1.00000000e+02 5.74855015e-19
 5.09826784e-19 6.15062550e+01 3.90199386e-17 5.48723162e-19
 9.39650420e-19 2.21098324e-12 5.86992541e+00 7.67262232e-19
 3.29542786e-11 1.77344061e+01 6.92605978e-12 1.29602483e-19
 5.28925578e-12 1.02035919e-19 7.40055725e-12 1.27768658e-11
 6.37605896e-12 3.63654677e-12 2.98334456e-19 8.76941595e-20
 8.40573403e-12 3.19770517e-12 1.40020450e-19 2.16490862e-19
 1.51610478e-11 1.96452043e-11 1.48472769e-11 9.82278992e-12
 9.17877329e-12 9.95607591e-12 3.46583023e-12 6.11643275e-12
 6.32944744e-12 3.96520866e-12 4.64814180e-13 1.65143241e-19
 3.24822817e-12 3.53138321e-12 7.25320683e-12 8.53940610e-20
 5.16735781e-20 3.67363490e-12 1.83466767e-11 2.08658137e-11
 4.90371358e-13 8.47985781e-13 4.77893128e-12 8.50535245e-12
 2.52349025e-19 5.41513001e-12 6.36071362e-12 1.01686701e-18
 1.37742036e-12 5.60548262e-12 7.16207362e-12 1.58991172e-13
 9.21464222e-12 3.02973094e-20 6.61956068e-12 0.00000000e+00
 9.28809357e-09 2.53270373e-18 1.00000000e+02 9.88537374e-19
 4.72637070e-16 1.35472457e-11 2.37246698e-10 1.09604538e-18
 9.50341424e-10 7.56094724e-19 1.48366917e-18 3.07615718e-19
 0.00000000e+00 9.19107238e+01 4.22689566e-19 5.62125247e-19
 1.86969129e-18 7.54603510e-19 4.93784242e-16 9.87908779e-19
 1.00000000e+02 5.43651957e-19 1.00000000e+02 1.00000000e+02
 4.81101291e-19 7.35850242e-19 1.00000000e+02 1.00000000e+02
 2.84643248e-11 8.10239886e-19 1.13361796e-18 1.08261333e-18
 7.55857840e-19 1.00000000e+02 1.82062766e-18 2.07431718e-18
 6.08353378e-08 3.83329008e-12 4.20364080e-19 9.82319147e-19
 1.31663673e-11 3.27596088e-18 8.53723526e-19 1.23814802e-18
 7.42126758e-19 3.17331182e-19 4.74904785e-19 4.97168244e-19
 9.35618190e-19 7.04578538e-19]
bias is  -23.92371797076165
weight vector is :  [ 5.49728308 -0.9933336 ]
```



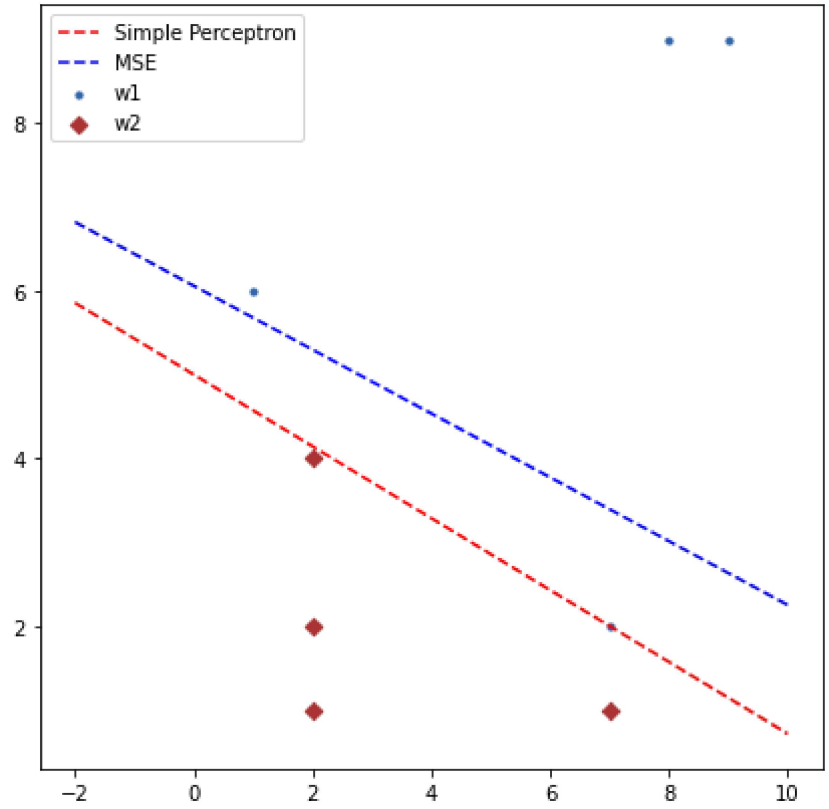


Question 5

```
# import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
# import pandas as pd

def perceptron():
    w1 = np.array([[1, 1, 6], [1, 7, 2], [1, 8, 9], [1, 9, 9]])
    w2 = np.array([[1, 2, 1], [1, 2, 2], [1, 2, 4], [1, 7, 1]])
    data = np.concatenate([w1, -w2], axis = 0)
    print("Data : \n", data)
    w1 = np.array([[1, 6], [7, 2], [8, 9], [9, 9]])
    w2 = np.array([[2, 1], [2, 2], [2, 4], [7, 1]])
    total_epoch = [0, 0]
    print("Classification using a Simple Pereptron  ......   ")
    for learning_rate in [0.1]:
        k = 0
        Q = 0                           #criterian theta
        a = np.array([0.1, 0.1, 0.1])           #weights
        missclasified_vectors = True
        epoch = 1
        while((missclasified_vectors) or not(k == 0)):
            if(k == 0):
                epoch = epoch + 1
                missclasified_vectors = False
            dot_product = np.dot(a, data[k])
            if(dot_product <= 0):
                a = a + learning_rate * data[k]
                missclasified_vectors = True
            k = (k + 1) % (len(data))
    print("The final weight vector is : ", a)
    f, ax = plt.subplots(figsize=(7, 7))
    c1, c2 = "#3366AA", "#AA3333"
    ax.scatter(*w1.T, c=c1,s = 10, label = "w1")
    ax.legend()
    ax.scatter(*w2.T, c=c2, marker="D", label = "w2")
    ax.legend()
    x_vec = np.linspace(-2, 10, 5)
    y_vec = ((a[1] * x_vec) + a[0])/a[2]
    y_vec = -y_vec
    plt.plot(x_vec, y_vec, 'r--', label = "Simple Perceptron")
    plt.legend()
    print("Classification using MSE prcodure of Pereptron  ......   ")
    b = np.array([1, 1, 1, 1, 1, 1, 1, 1])
```

```
    a = np.linalg.lstsq(data, b.transpose())
    a = np.array(a[0])
    print("The final weight vector is : ", a)
    x_vec = np.linspace(-2, 10, 5)
    y_vec = ((a[1] * x_vec) + a[0])/a[2]
    y_vec = -y_vec
    plt.plot(x_vec, y_vec, 'b--', label = "MSE")
    plt.legend()
    plt.show()

if(__name__ == '__main__'):
    perceptron()
```

```
Data :
 [[ 1  1  6]
 [ 1  7  2]
 [ 1  8  9]
 [ 1  9  9]
 [-1 -2 -1]
 [-1 -2 -2]
 [-1 -2 -4]
 [-1 -7 -1]]
Classification using a Simple Pereptron  ......
The final weight vector is :  [-3.5  0.3  0.7]
Classification using MSE prcodure of Pereptron  ......
The final weight vector is :  [-1.18701981  0.07460574  0.19591589]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:43: FutureWarning: `rcor
To use the future default and silence this warning we advise to pass `rcond=None`, tc
```



Double-click (or enter) to edit