QUESTION-1

SVM Code

```
import numpy as np
import matplotlib.pyplot as plt
from cvxopt import matrix
from cvxopt import solvers

class svm:

    def __init__(self,train_data,train_labels):

        self.__X=train_data
        self.__Y=np.array([train_labels,])
        self.__weight=[]
        self.__bias=None
        self.__split_for_plotting()

    def train(self):

        n=self.__X.shape[0]

        H=matrix(np.multiply((self.__Y.T @ self.__Y),(self.__X @
self.__X.T)).astype(np.float))
        f=matrix(np.array([-1]*n).astype(np.float),tc='d')
        A=matrix(-np.eye(n).astype(np.float))
        a=matrix(np.array([0.0]*n).astype(np.float))
        B=matrix(self.__Y.astype(np.float),tc='d')
        b=matrix(0.0)

        solvers.options['show_progress'] = False
        solution = solvers.qp(H,f,A,a,B,b)
        alphas = np.array(solution['x'])

        self.__weight=np.zeros_like(self.__X[0],dtype=float)
```

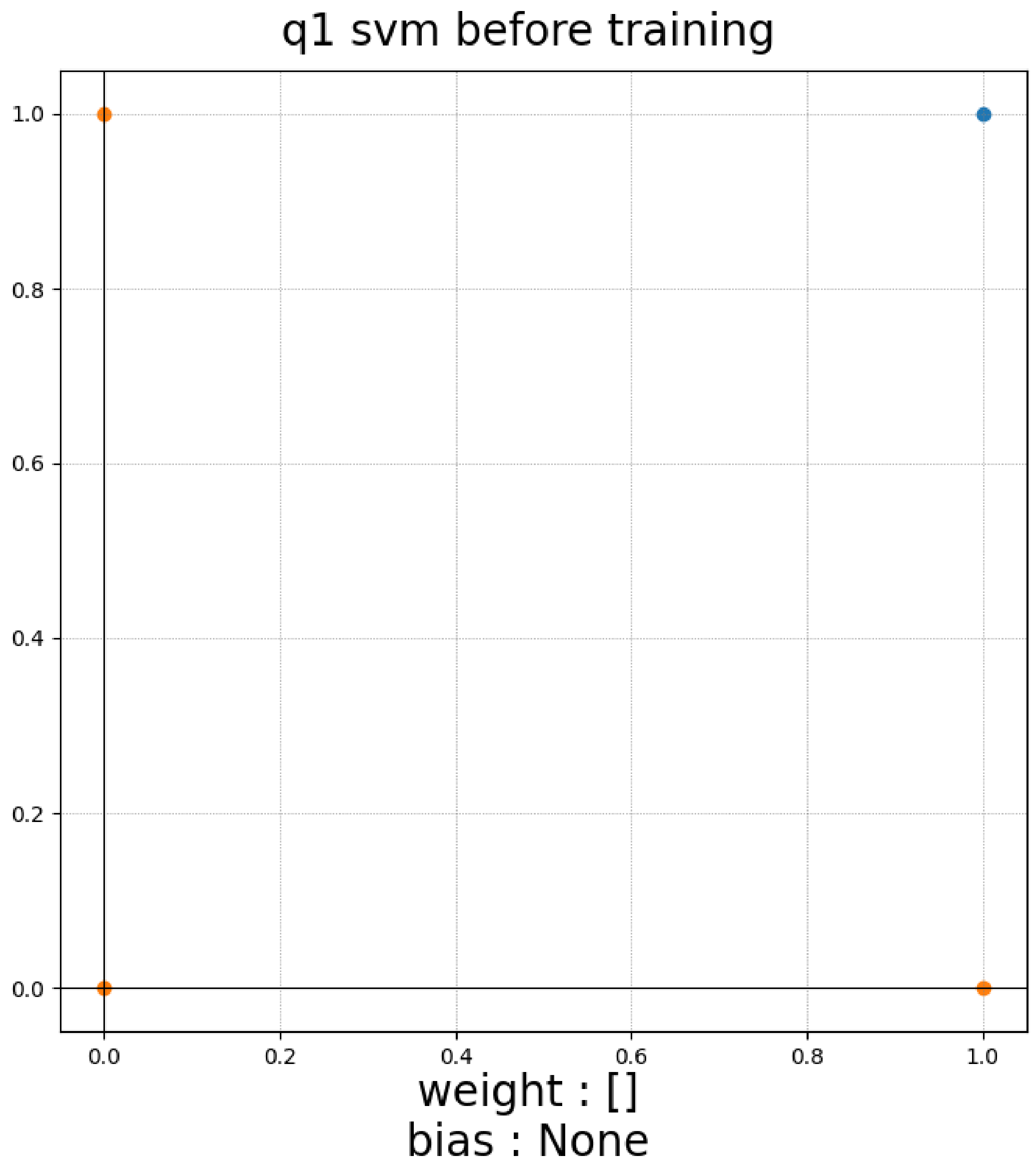
```
        for i,alpha in enumerate(alphas):
                self.__weight+=alpha*self.__Y[0][i]*self.__X[i]

        max_index=np.argmax(alphas)
        self.__bias = self.__Y[0][max_index] - self.__weight.T @
self.__X[max_index]

    def __split_for_plotting(self):

        self.x1=[]
        self.y1=[]
        self.x2=[]
        self.y2=[]
        for i,p in enumerate(self.__X):
            if self.__Y[0][i]==1:
                self.x1.append(p[0])
                self.y1.append(p[1])
            else:
                self.x2.append(p[0])
                self.y2.append(p[1])

    def show_plot(self,title=''):

        # styles
        plt.figure(figsize=(8,8))
        plt.figtext(0.5, 0.9, title, ha="center", fontsize=20)
        plt.axvline(0,color='black',linewidth=.8)
        plt.axhline(0,color='black',linewidth=.8)
        plt.grid(color='grey', linestyle=':', linewidth=.5)

        # plotting data
        plt.scatter(self.x2,self.y2)
        plt.scatter(self.x1,self.y1)

        # plotting decision boundary
        if np.any(self.__weight):
            a,b=self.__weight
            c=self.__bias
```

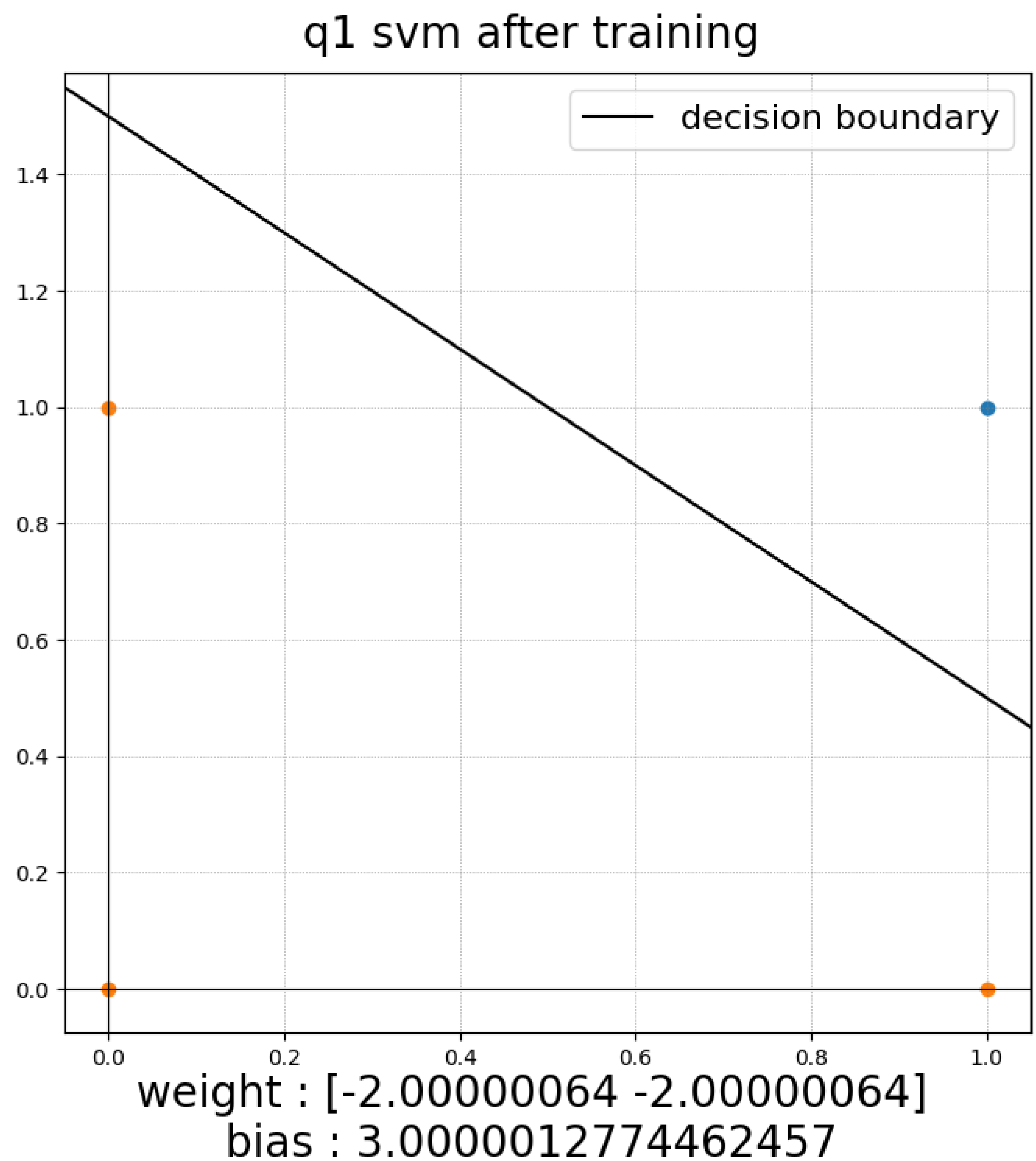
```
            if b==0:
                plt.axvline(-c/a, c='black', label='decision
boundary')
            else:
                y_intercept=-c/b
                slope=-a/b
                plt.axline((0,y_intercept), slope=slope, c='black',
label='decision boundary')
            plt.legend(loc='best',fontsize=16)

        plt.figtext(0.5, 0.01, f'weight : {self.__weight}\nbias :
{self.__bias}', ha="center", fontsize=20)
        title=title.replace(' ','_')
        plt.savefig(f'output_images/{title}.png')
        plt.show()

def demo(data,label,plot_title=''):

    a=svm(data,label)
    if plot_title!='':
        plot_title=plot_title+' '

    a.show_plot(plot_title+"svm before training")
    a.train()
    a.show_plot(plot_title+"svm after training")
```

Q1 Svm Demo

```
import svm,numpy as np

x1=[0,0,1,1]
x2=[0,1,0,1]

X=np.array(list(zip(x1,x2)))

Y_svm=np.array([1,1,1,-1])
```

```
svm.demo(X,Y_svm,plot_title="q1")
```





Q2

Perceptron

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

```
class Perceptron:

    def __init__(self,train_data,train_labels):

        assert (len(train_data)==len(train_labels)),"length of train_data and 
train_labels must match"
        self.__raw_data=train_data

        # append 1 to train data to get y

self.__train_data=np.append(train_data,np.array([[1]]*len(train_data)),axis=1)

        # check if train_labels is valid
        assert all(np.isin(train_labels,[0,1])),"'train_labels' should contain 
only 0s or 1s"
        self.train_labels=train_labels

        # negate y values from class 1
        for i,c in enumerate(train_labels):
            if c==1:
                self.__train_data[i]=-self.__train_data[i]

        # initialise weight vector, set default learning rate
        self.__weight=np.zeros_like(self.__train_data[0])
        self.__learning_rate=0.01
        self.__split_for_plotting()

    def __split_for_plotting(self):

        self.x1=[]
        self.y1=[]
        self.x2=[]
        self.y2=[]
        for i,p in enumerate(self.__raw_data):
            if self.train_labels[i]==1:
                self.x1.append(p[0])
                self.y1.append(p[1])
            else:
                self.x2.append(p[0])
                self.y2.append(p[1])

    # to manually set wait vector
    def set_weight(self,weight):
        if weight.shape!=self.__weight.shape:
```

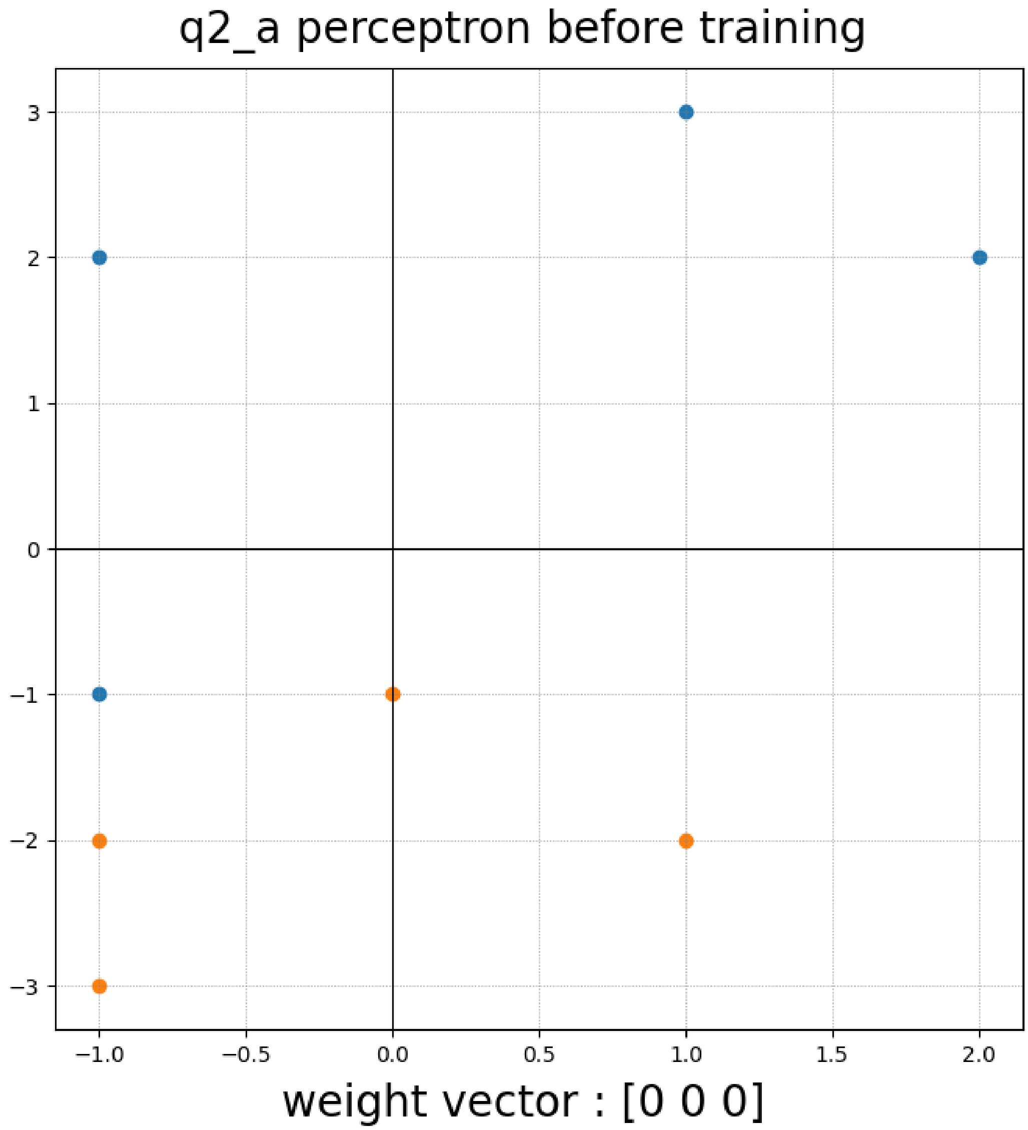
```
            raise ValueError(f"given weight vector must be of shape 1x(d+1),
1x{self.__weight.shape[0]} here")
        self.__weight=weight

    # to set learning rate
    def set_learning_rate(self,learning_rate):
        self.__learning_rate=learning_rate

    # to do 1 iteration of learning
    def train(self):
        gradient_of_Jp=np.zeros_like(self.__weight)
        for y in self.__train_data:
            if not self.__weight @ y > 0:
                gradient_of_Jp+=y
        print(f'gradient of Jp = {gradient_of_Jp}')
        print(f'new weight = old weight + learning rate * gradient of Jp\n
= {self.__weight} + {self.__learning_rate} * {gradient_of_Jp}')
        self.__weight= self.__weight + self.__learning_rate * gradient_of_Jp
        print(f"           = {self.__weight}\n")

    # to plot the data and the decision boundary
    def show_plot(self,title=''):

        # styles
        plt.figure(figsize=(8,8))
        plt.figtext(0.5, 0.9, title, ha="center", fontsize=20)
        plt.axvline(0,color='black',linewidth=.8)
        plt.axhline(0,color='black',linewidth=.8)
        plt.grid(color='grey', linestyle=':', linewidth=.5)

        # plotting data
        plt.scatter(self.x1,self.y1)
        plt.scatter(self.x2,self.y2)

        # plotting decision boundary
        if np.any(self.__weight[:-1]):
            a,b,c=self.__weight
            if b==0:
                plt.axvline(-c/a, c='black', label='decision boundary')
            else:
                y_intercept=-c/b
                slope=-a/b
                plt.axline((0,y_intercept), slope=slope, c='black',
label='decision boundary')
```

```python
            plt.legend(loc='best',fontsize=16)


        plt.figtext(0.5, 0.04, "weight vector : "+str(self.__weight), ha="center", fontsize=20)
        title=title.replace(' ','_')
        plt.savefig(f'output_images/{title}.png')
        plt.show()

    def get_weight(self):
        return self.__weight

def demo(data,label,plot_title='',learning_rate=None,weight=None):

    a=Perceptron(data,label)
    if learning_rate is not None:
        a.set_learning_rate(learning_rate)
    if weight is not None:
        a.set_weight(weight)
    if plot_title!='':
        plot_title=plot_title+' '

    a.show_plot(plot_title+"perceptron before training")
    i=1
    prev_weight=a.get_weight()
    while True:
        print(f"perceptron iteration {i}:\n")
        a.train()
        a.show_plot(plot_title+f"perceptron iteration {i}")
        i+=1
        if np.allclose(prev_weight,a.get_weight()):
            break
        prev_weight=a.get_weight()

    print("no significant change in weight vector after this iteration. stopping.")
```

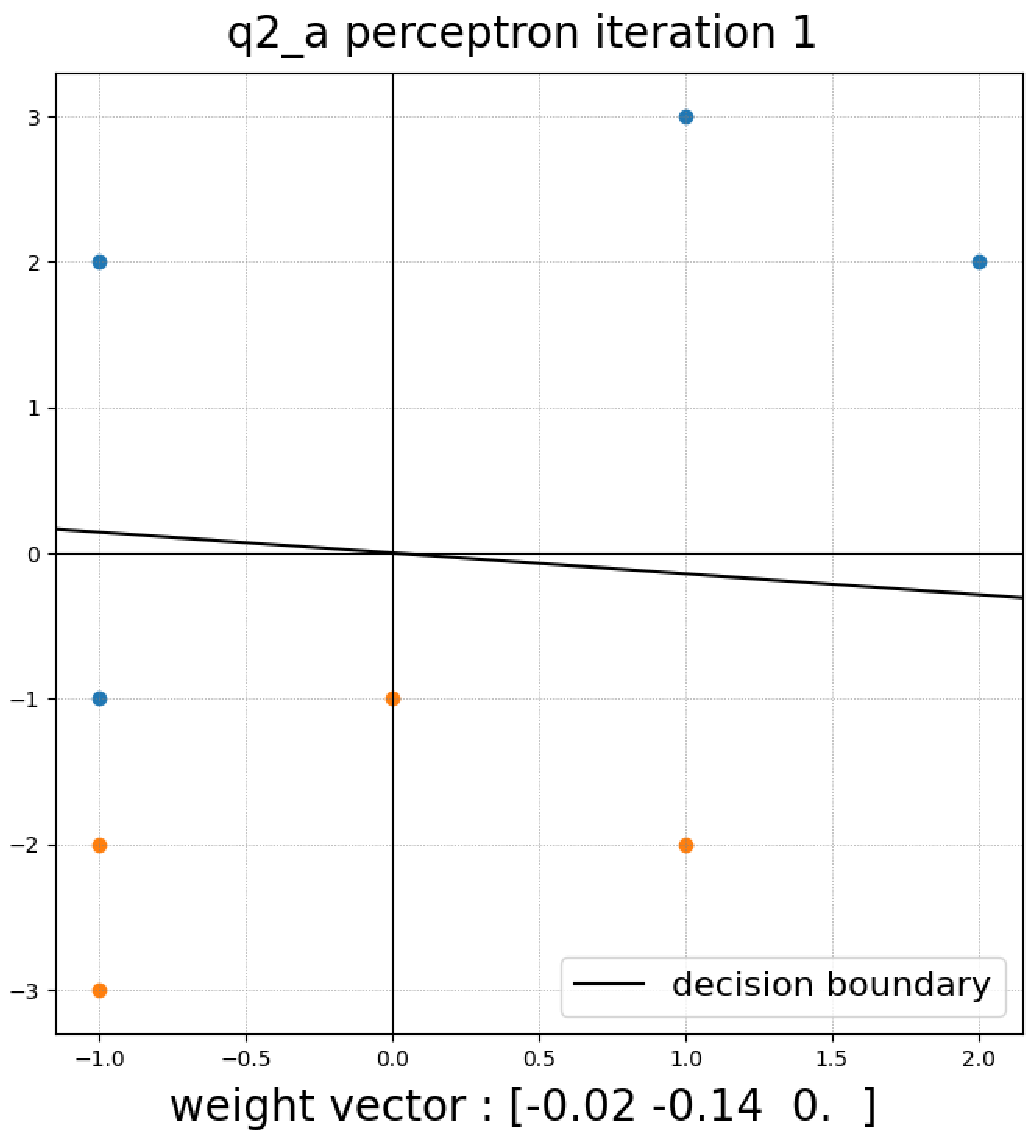
Q2 Demo (We use svm and perceptron classes defined as above)

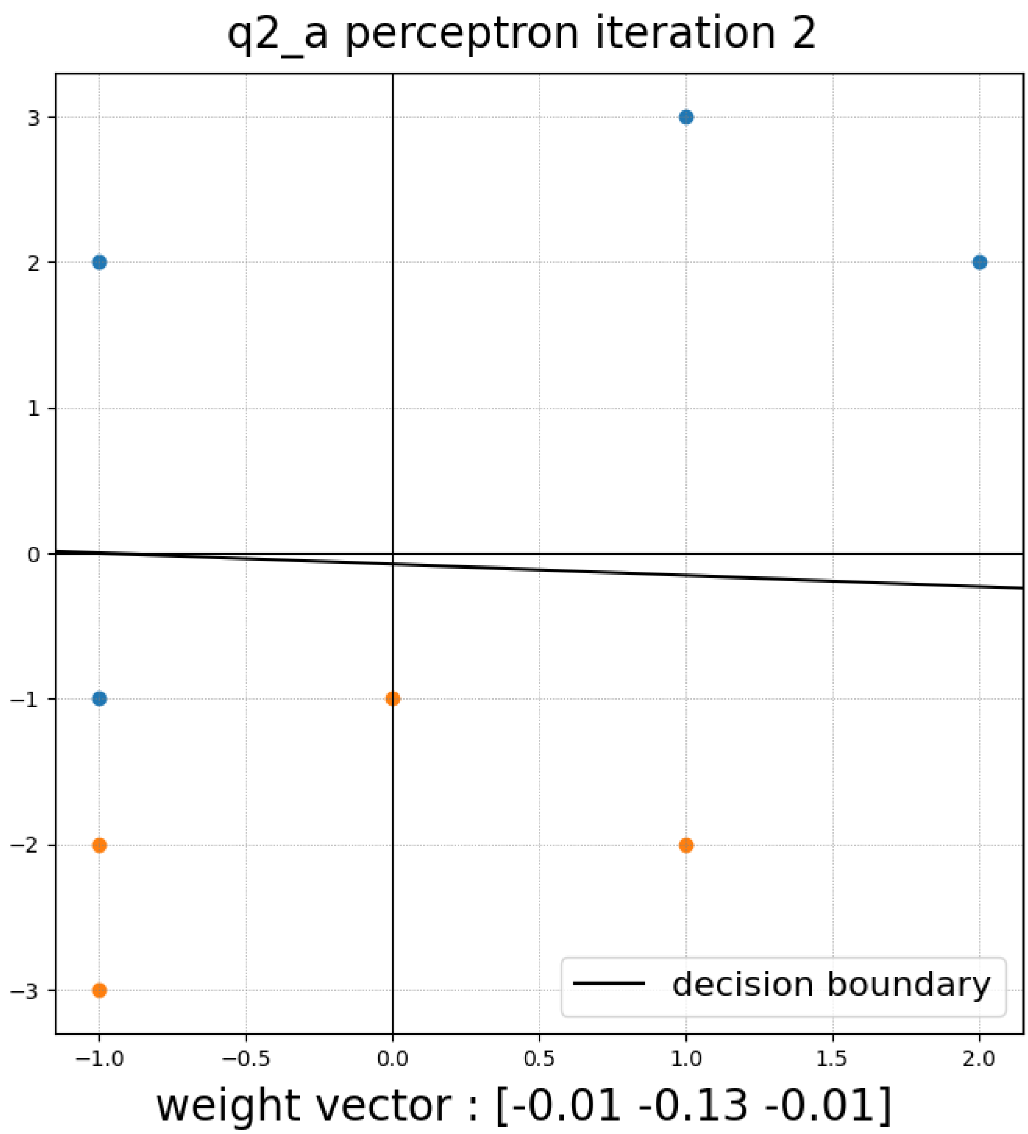
```python
import svm,numpy as np,perceptron
```

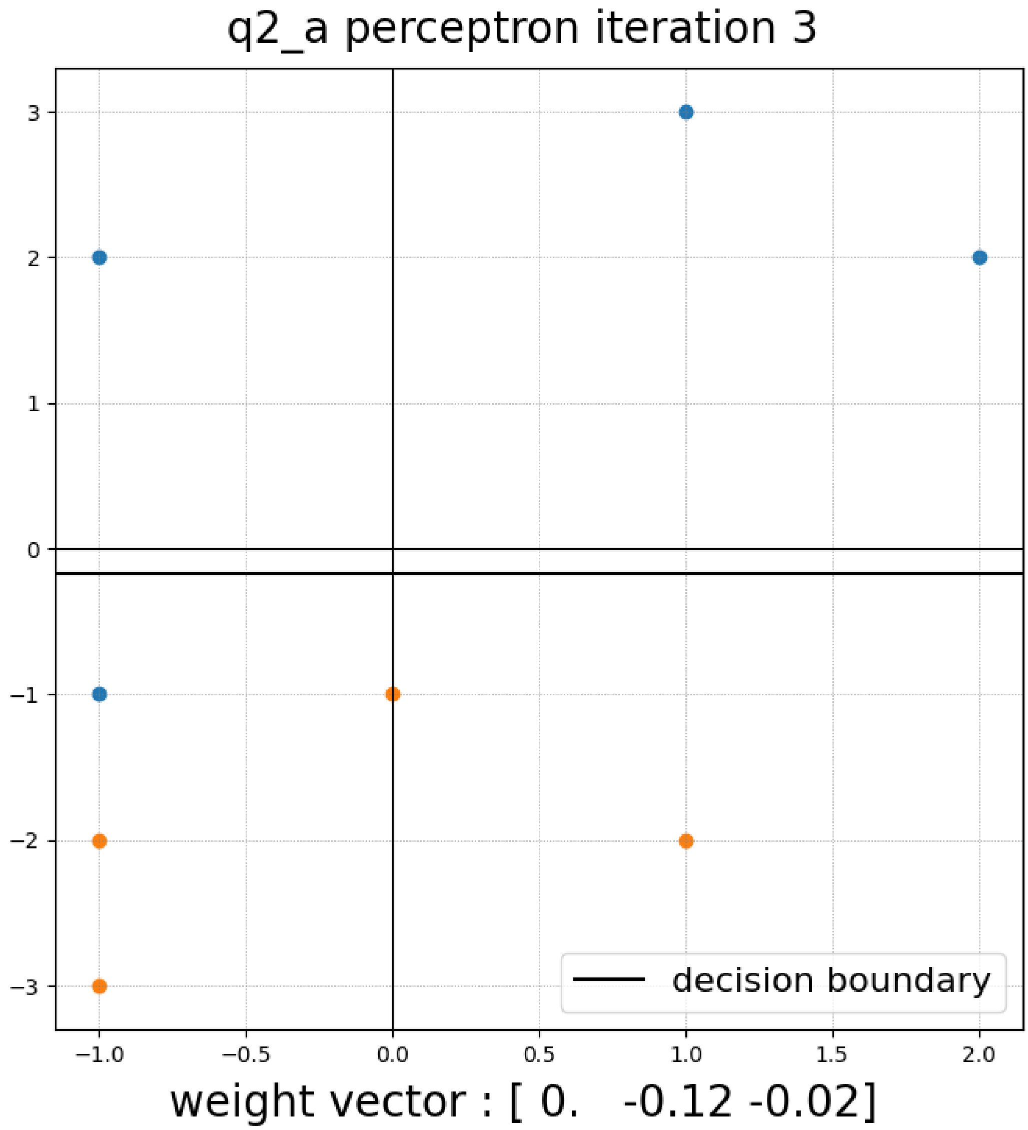
```
x1=[2,-1,-1,0,1,-1,1,-1]
x2=[2,-3,2,-1,3,-2,-2,-1]

X=np.array(list(zip(x1,x2)))

Y_svm =np.array([1,-1,1,-1,1,-1,-1,1])
Y_perc=np.array([1,0,1,0,1,0,0,1])

perceptron.demo(X,Y_perc,learning_rate=0.01,plot_title="q2_a")
perceptron.demo(X,Y_perc,learning_rate=0.5,plot_title="q2_b")

svm.demo(X,Y_svm,plot_title="q2")
```

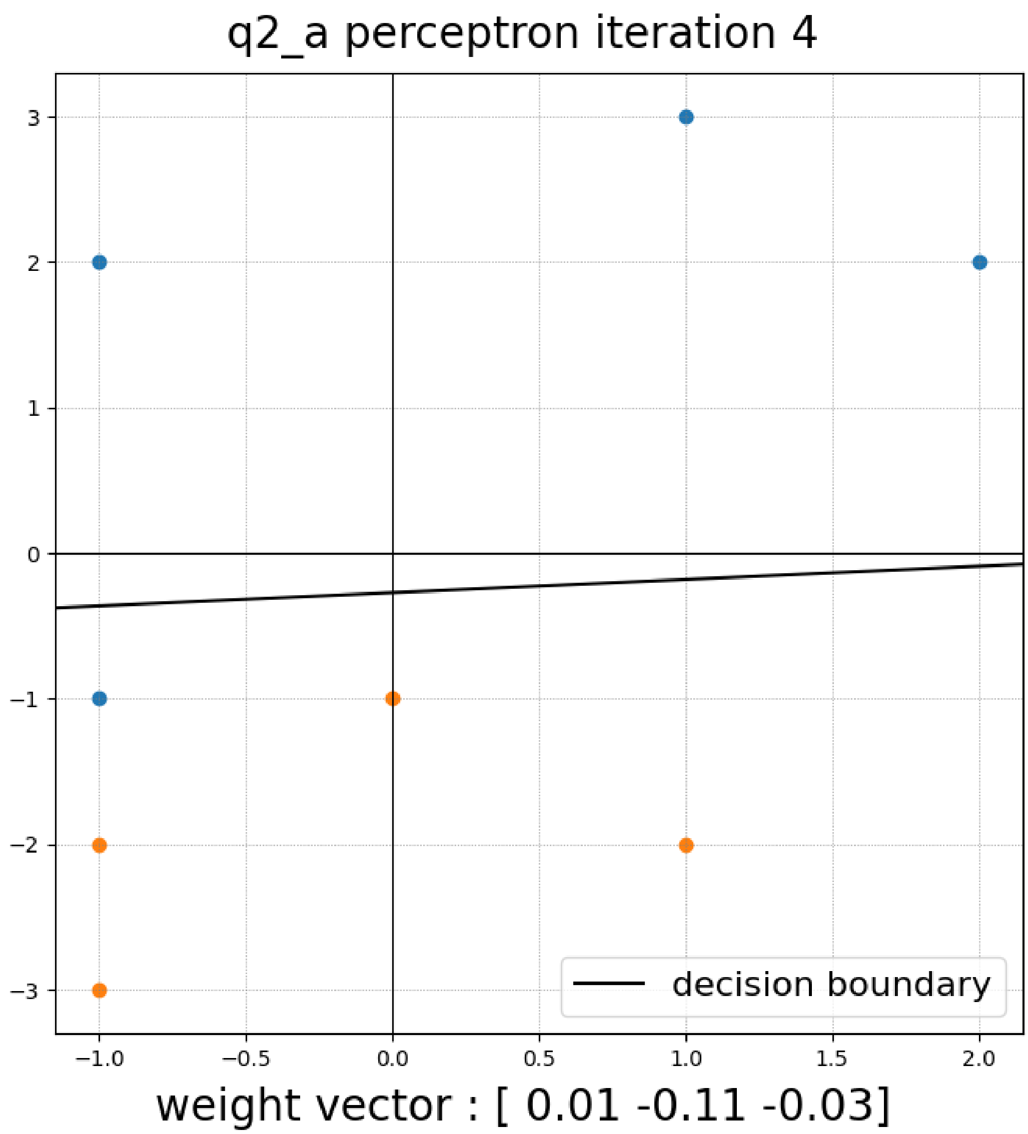


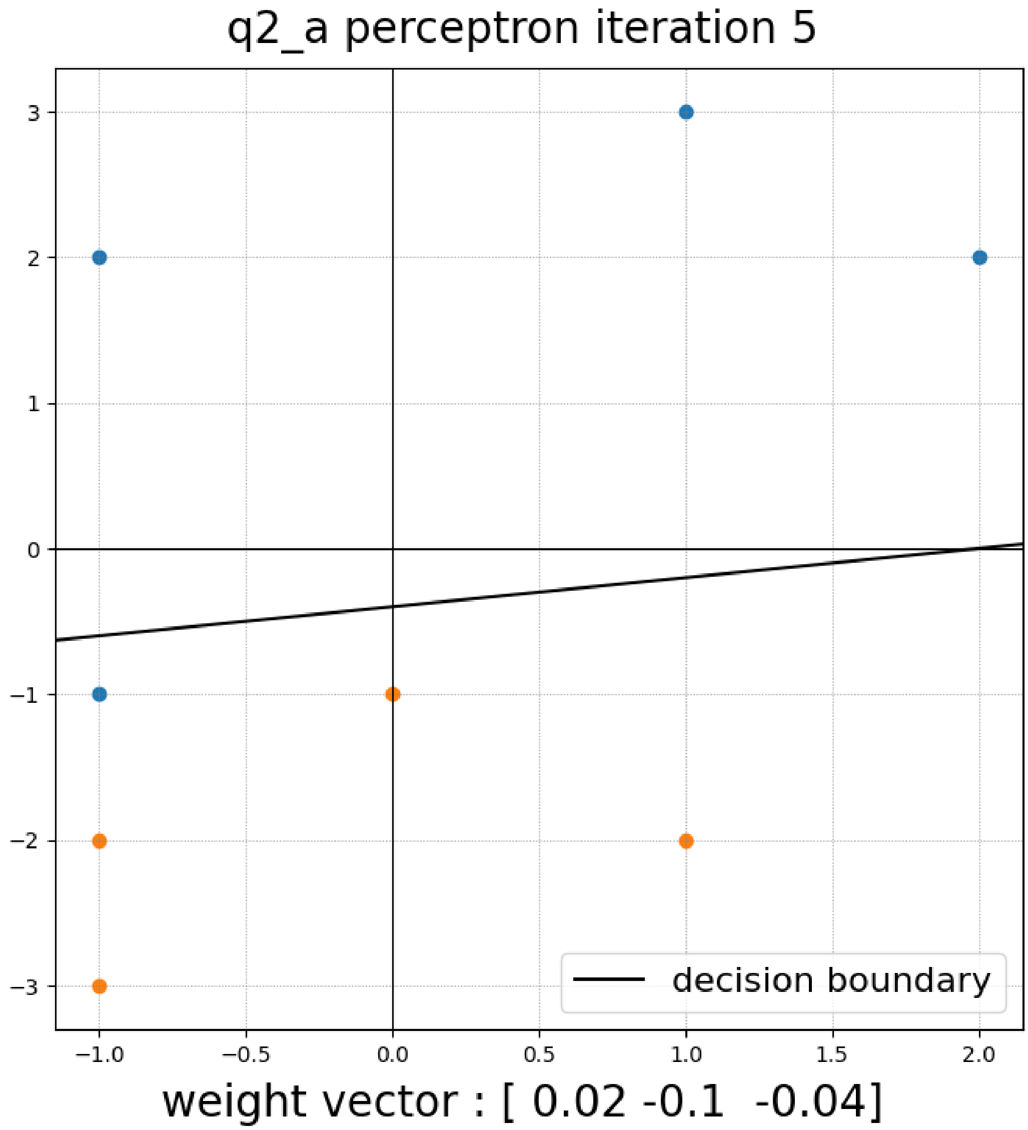
q2_a perceptron before training
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [0 0 0]

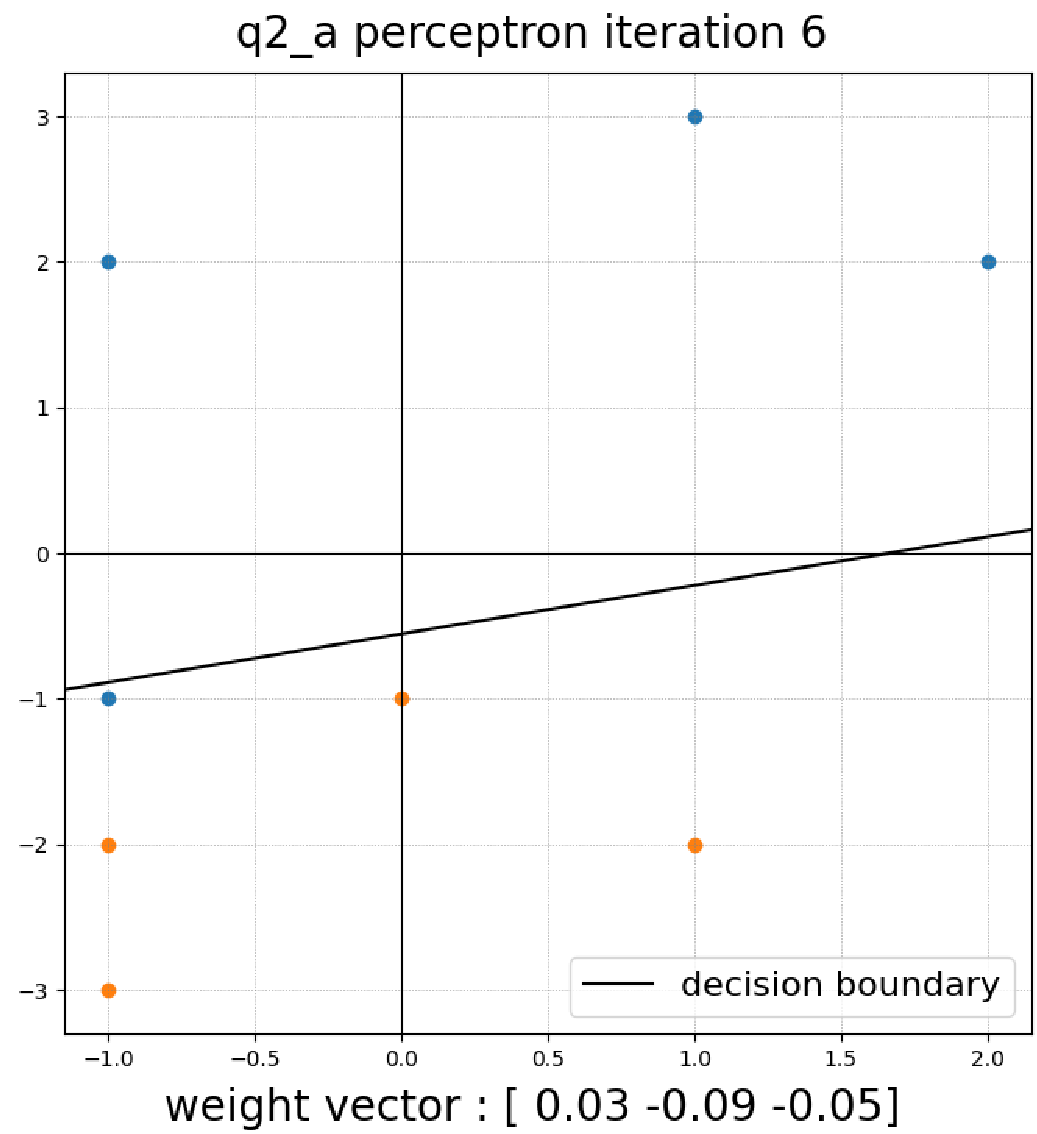

q2_a perceptron iteration 1
decision boundary
weight vector : [-0.02 -0.14 0.]

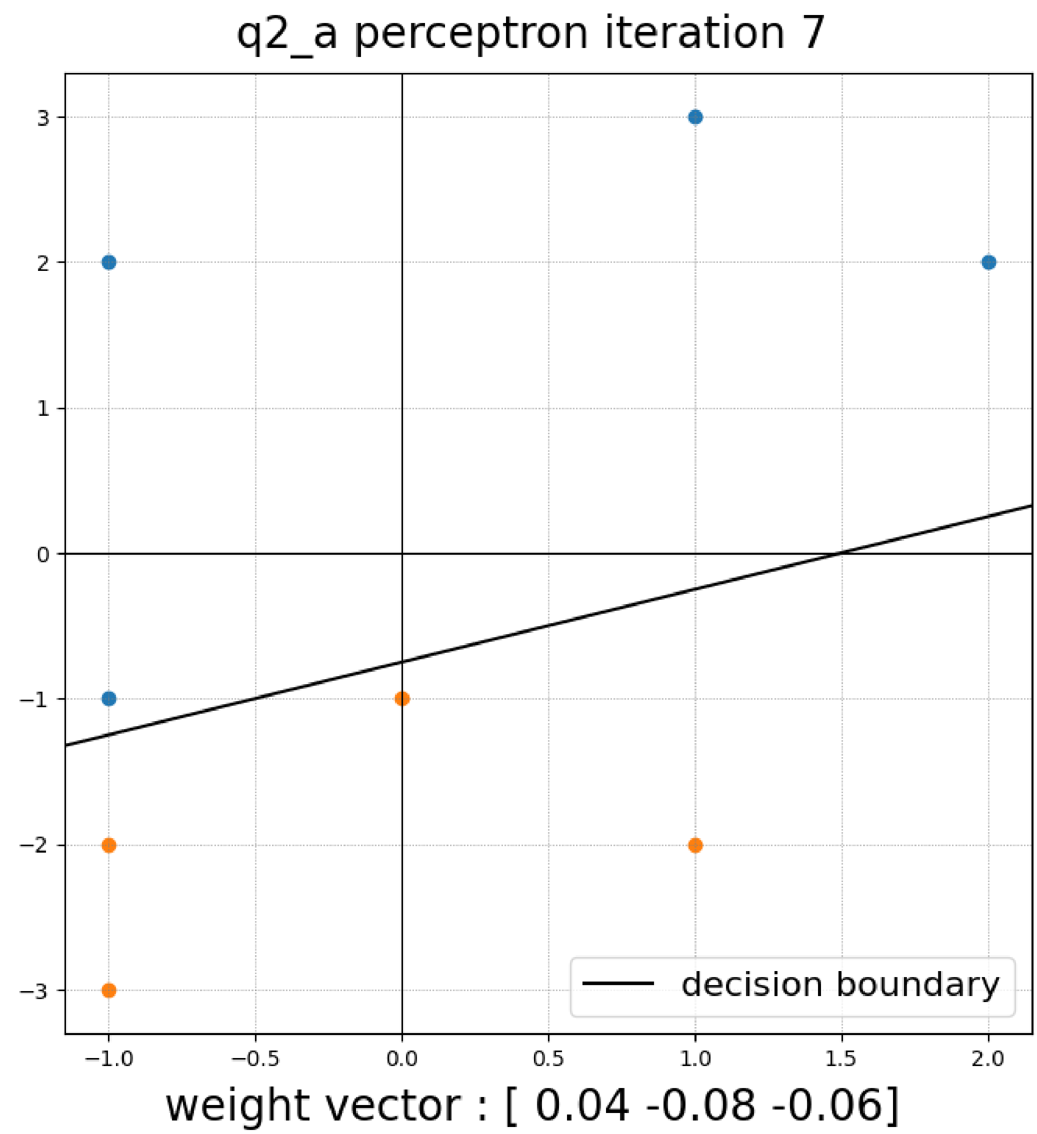

q2_a perceptron iteration 2
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [-0.01 -0.13 -0.01]

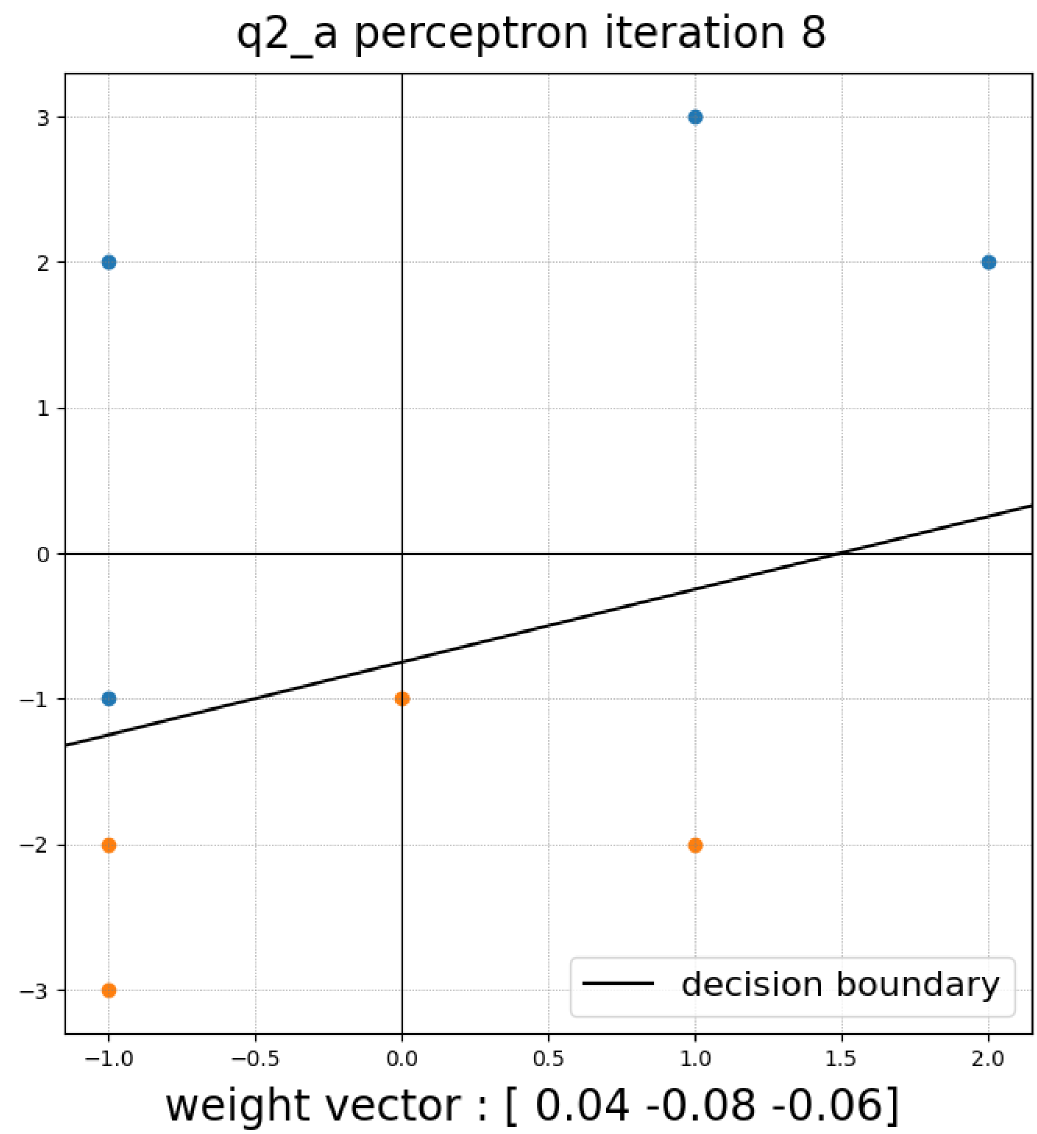

q2_a perceptron iteration 3
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0.   -0.12 -0.02]


q2_a perceptron iteration 4
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0.01 -0.11 -0.03]

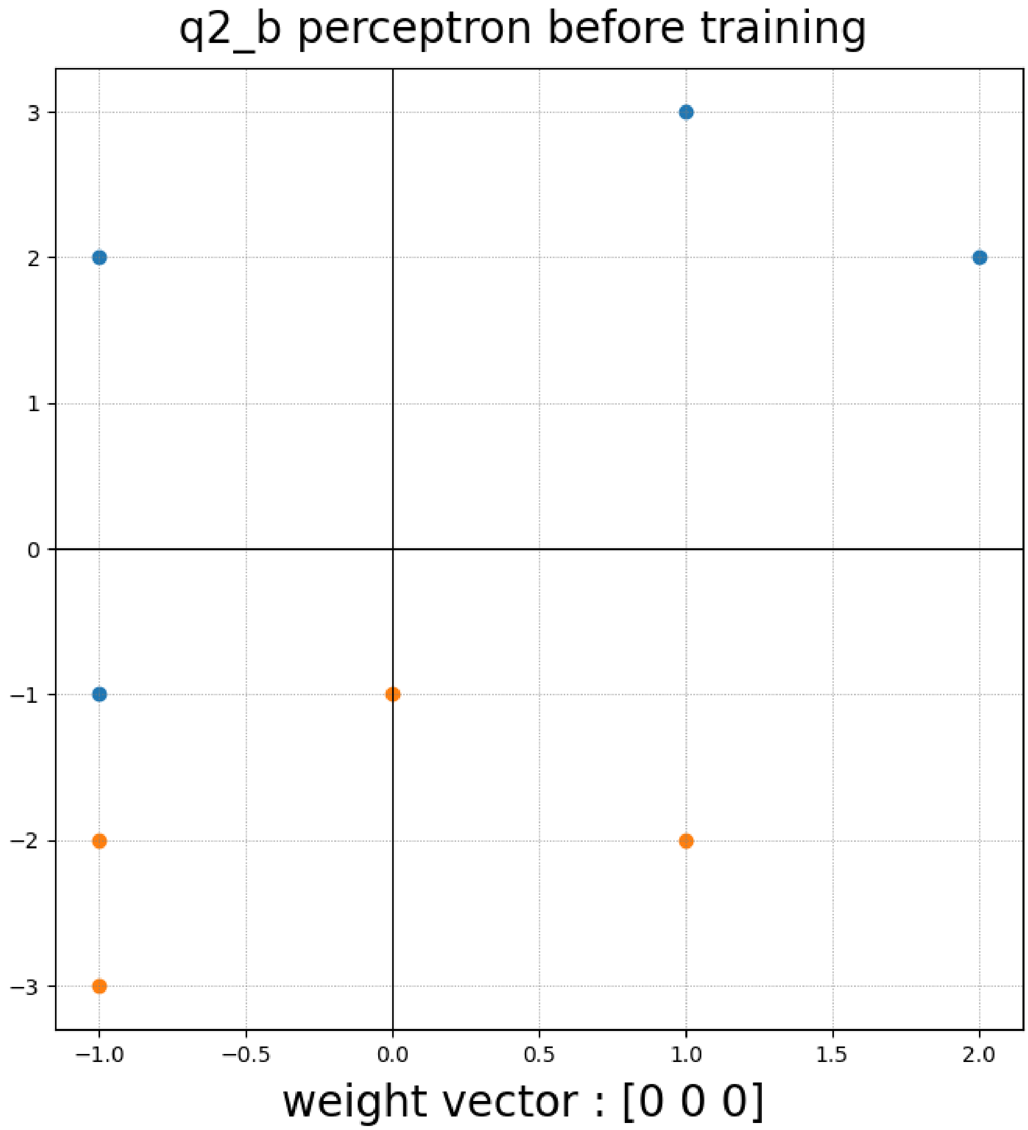

q2_a perceptron iteration 5
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0.02 -0.1 -0.04]

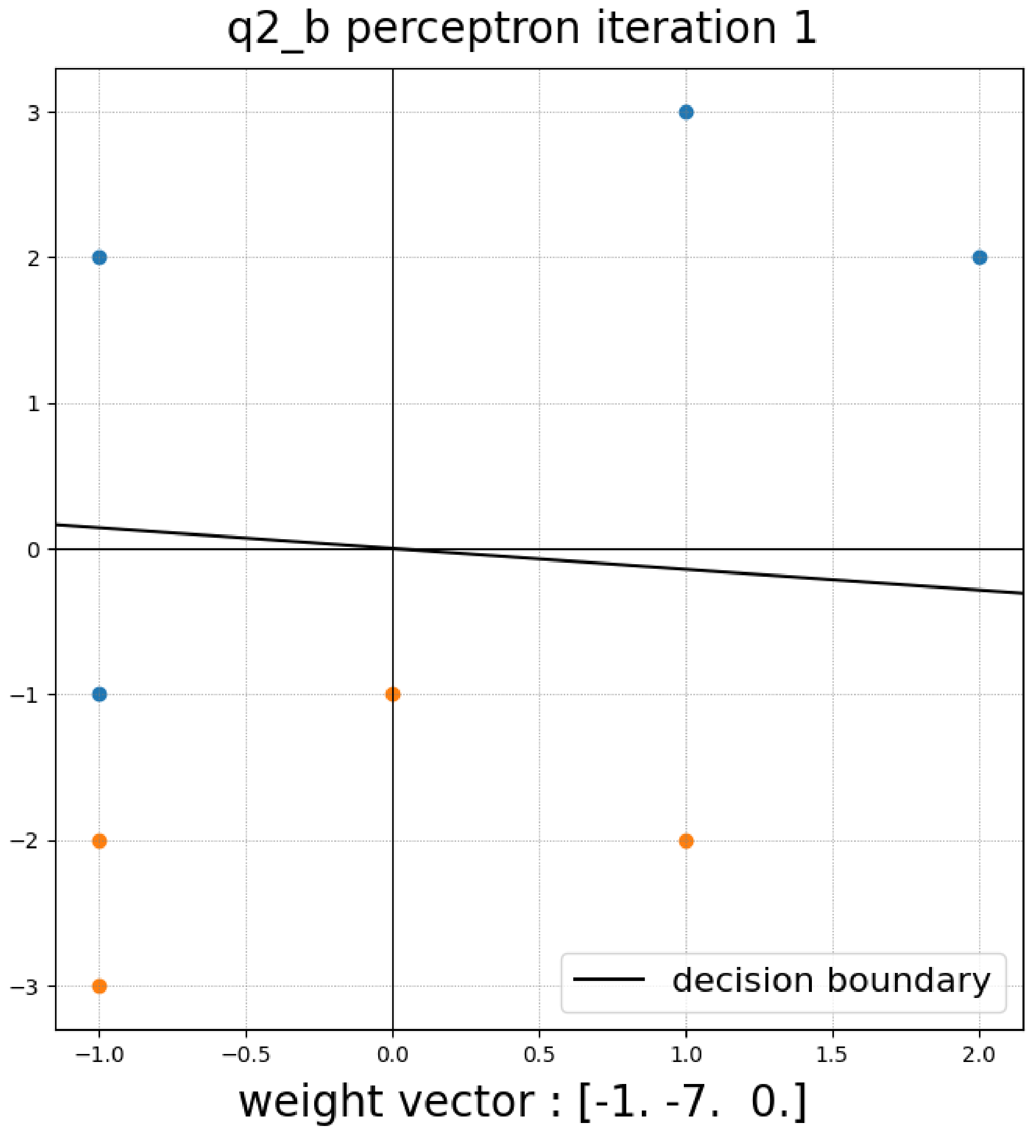

q2_a perceptron iteration 6
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0.03 -0.09 -0.05]

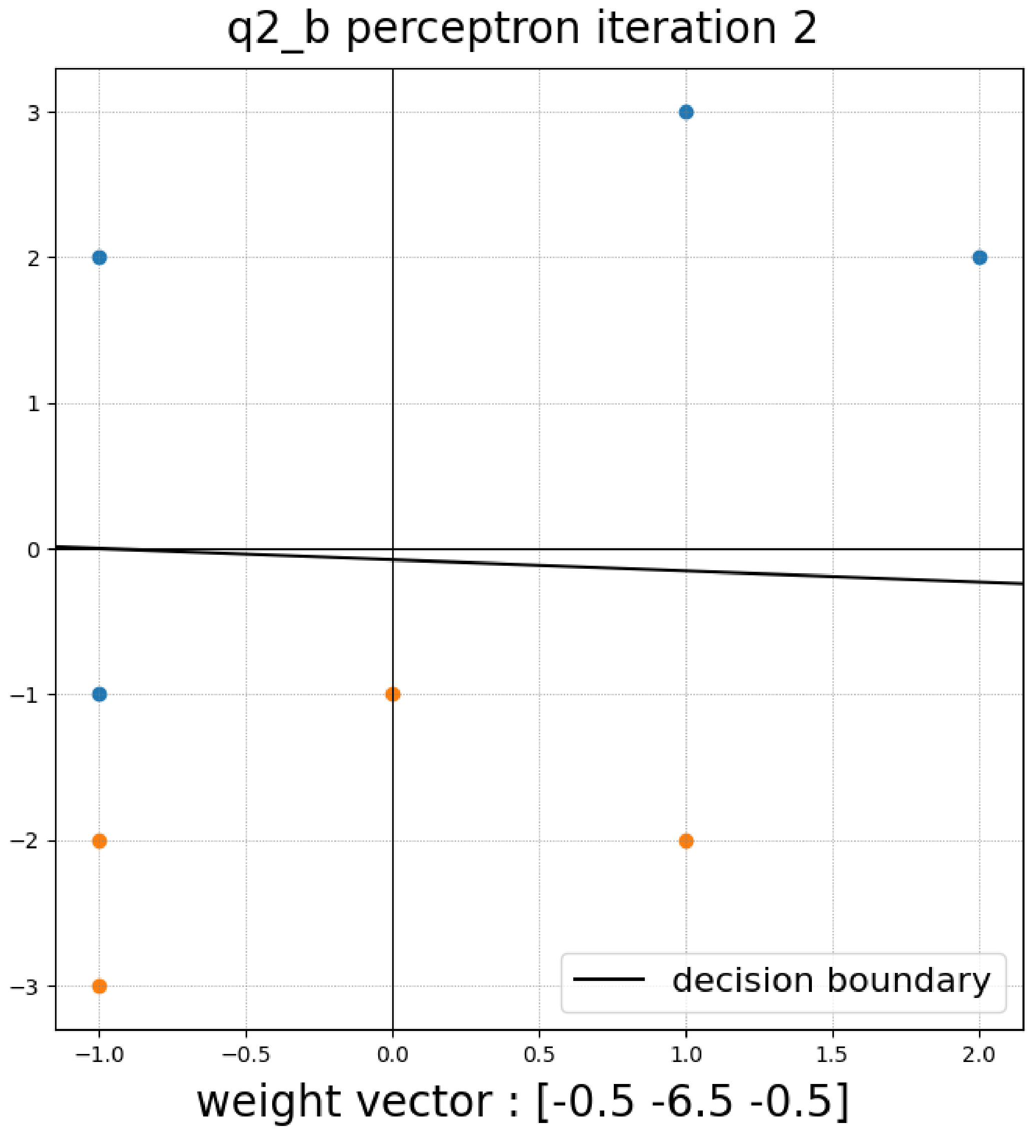

q2_a perceptron iteration 7
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0.04 -0.08 -0.06]

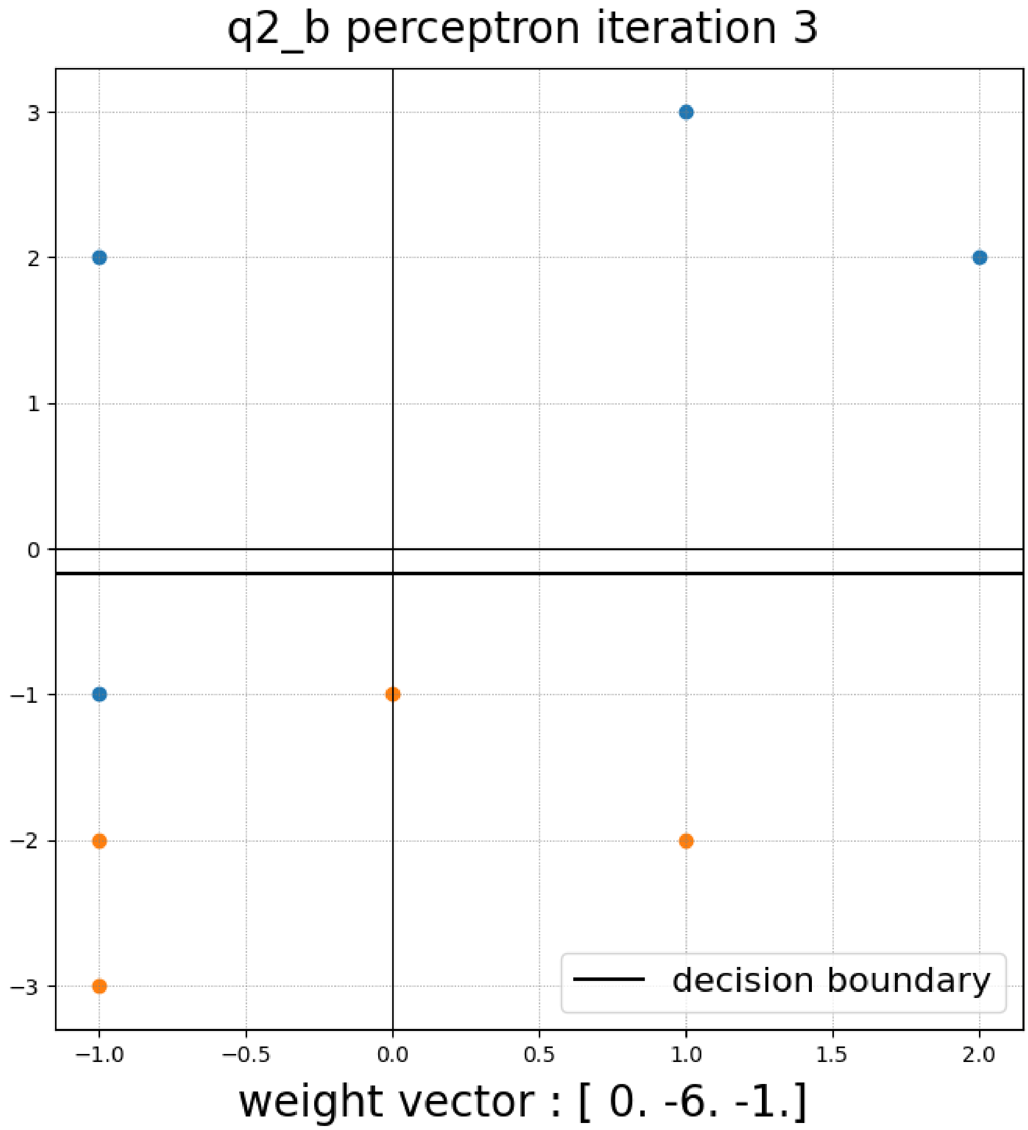


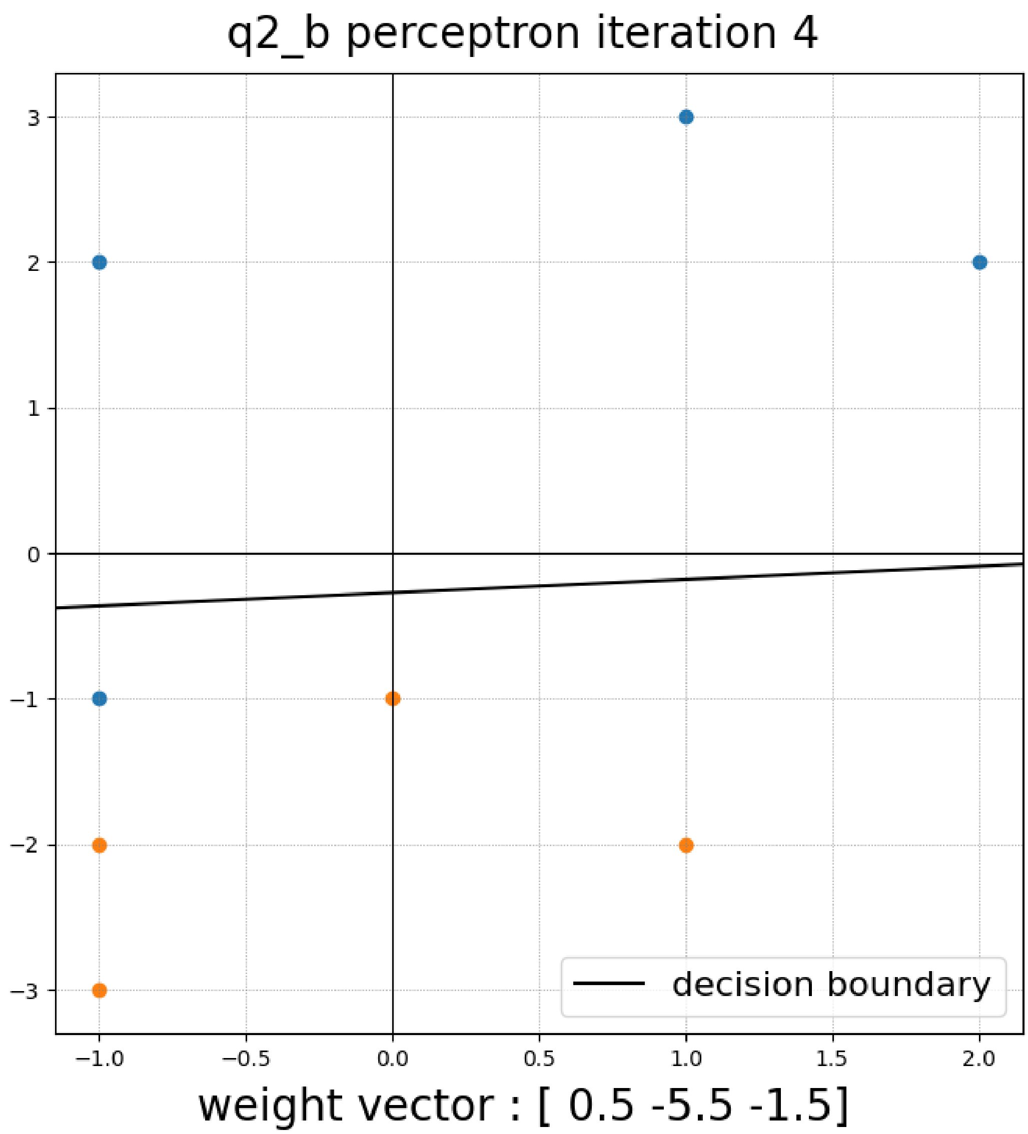
8 iterations for learning rate 0.01

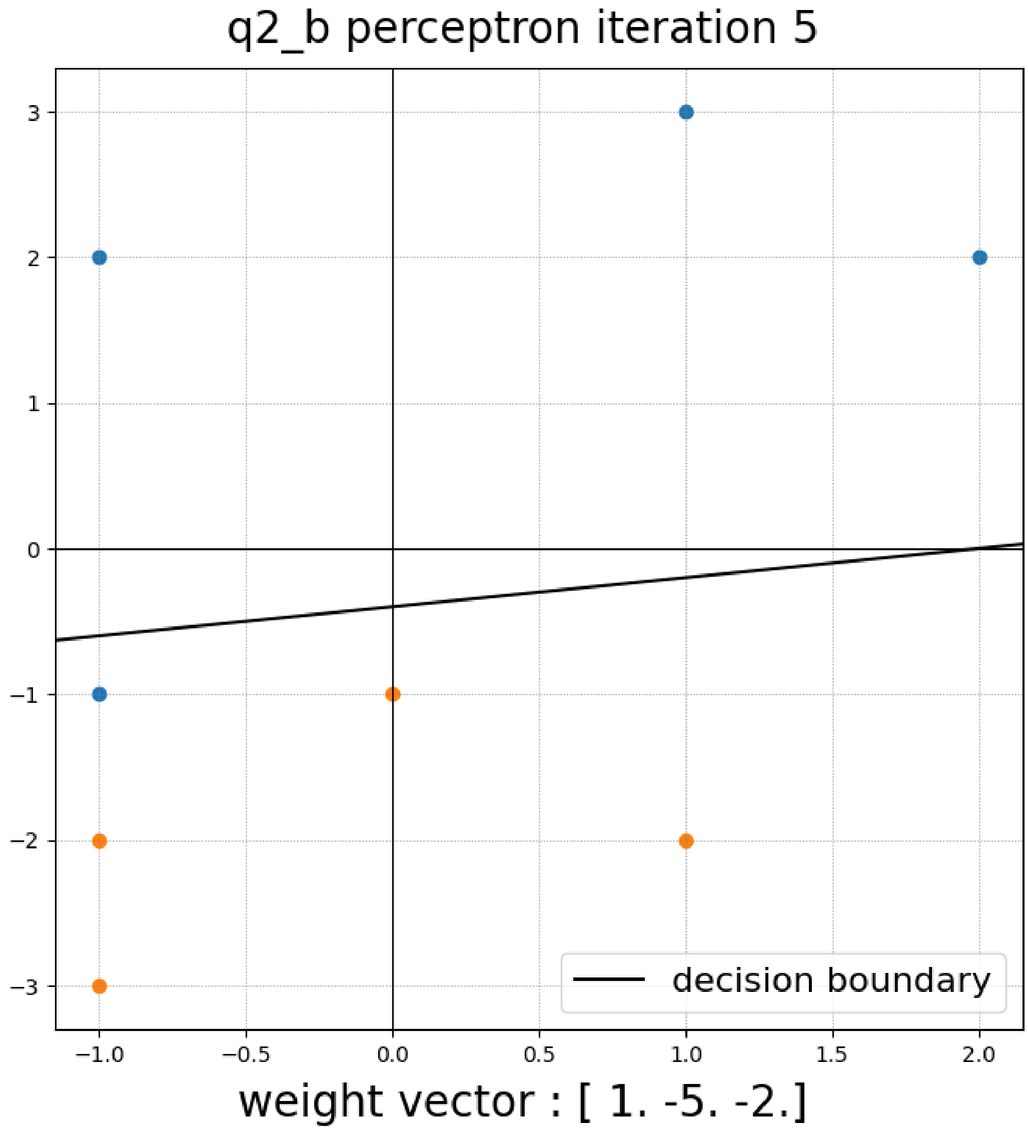

q2_b perceptron before training
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [0 0 0]

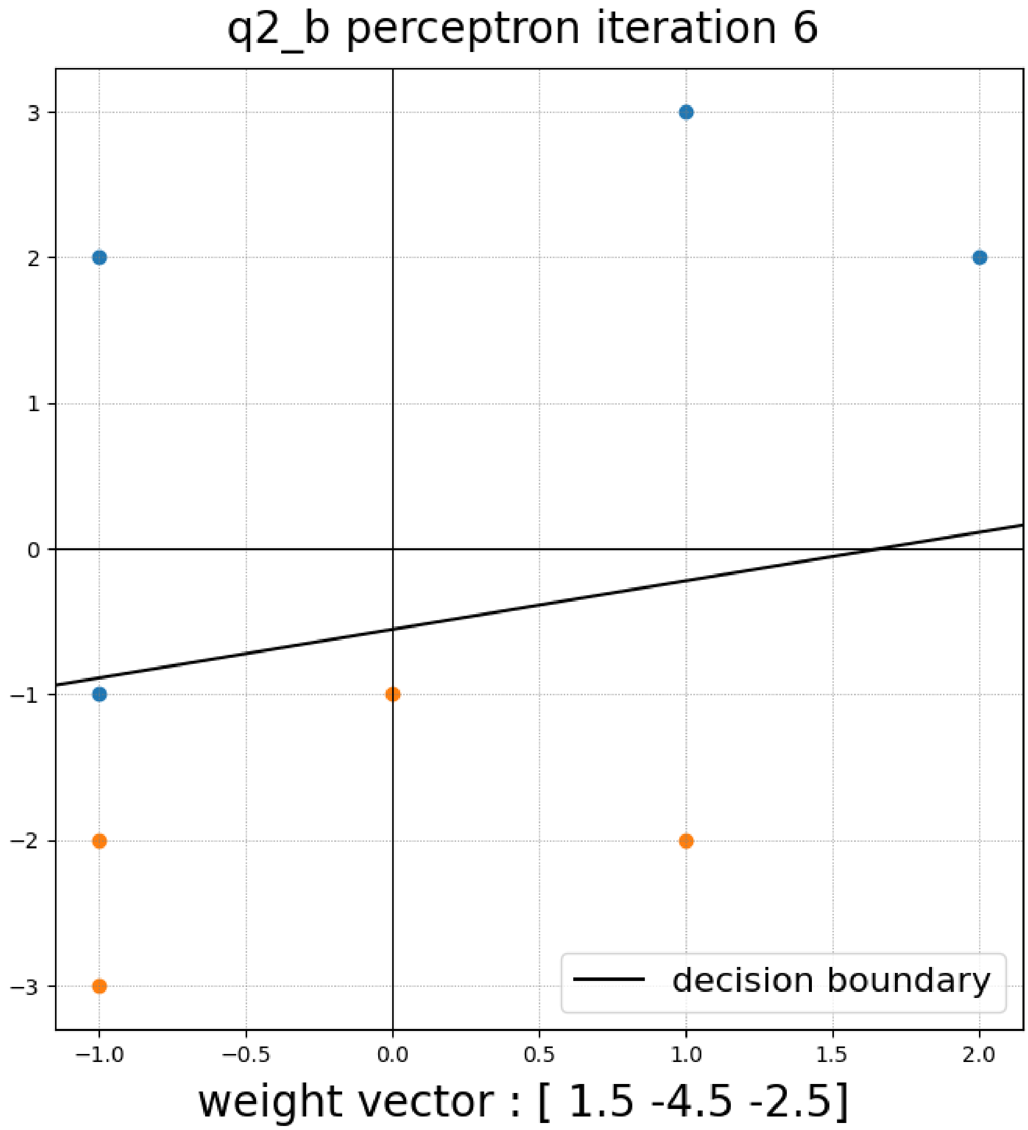

q2_b perceptron iteration 1
decision boundary
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [-1. -7. 0.]

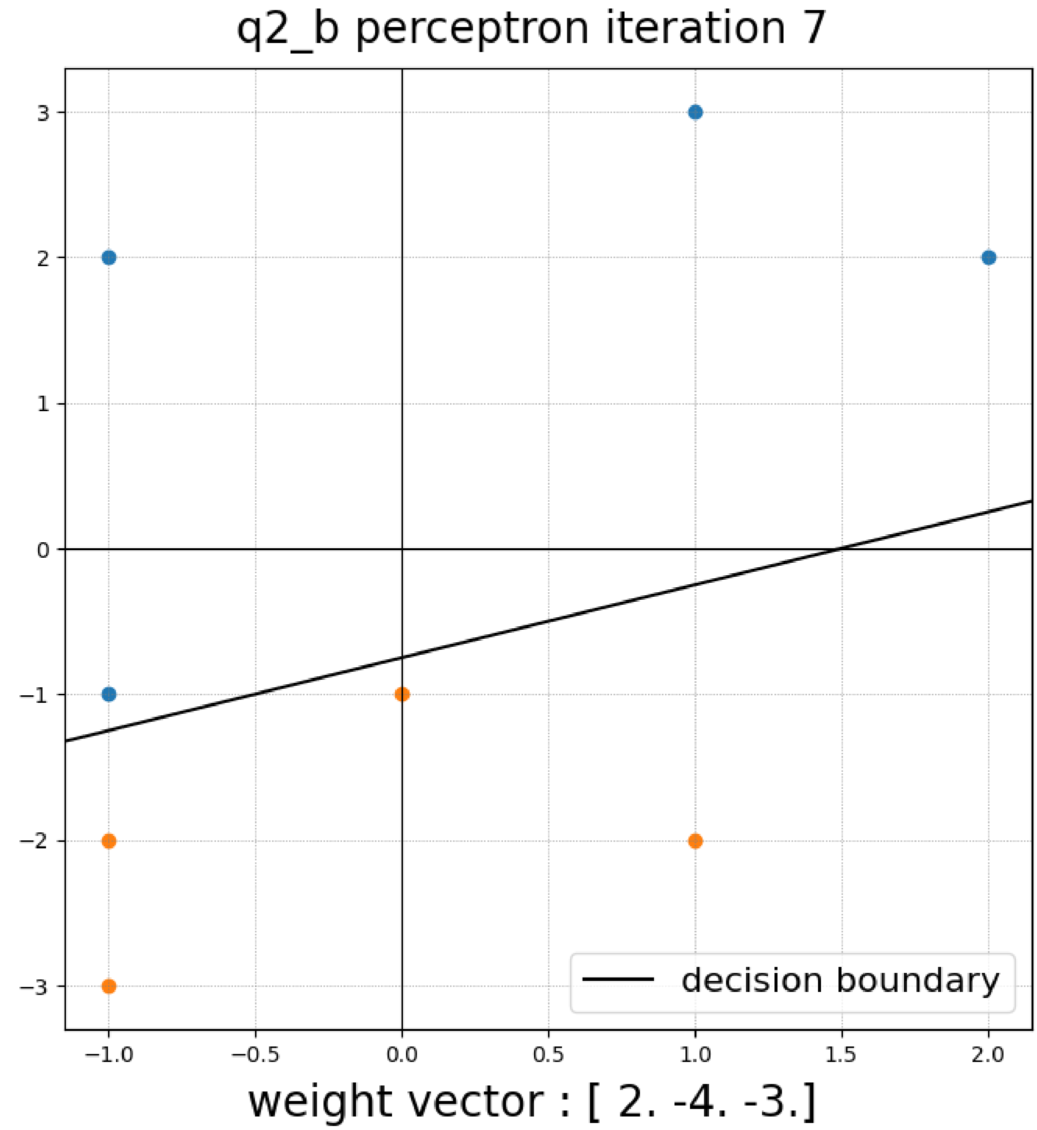

q2_b perceptron iteration 2
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [-0.5 -6.5 -0.5]

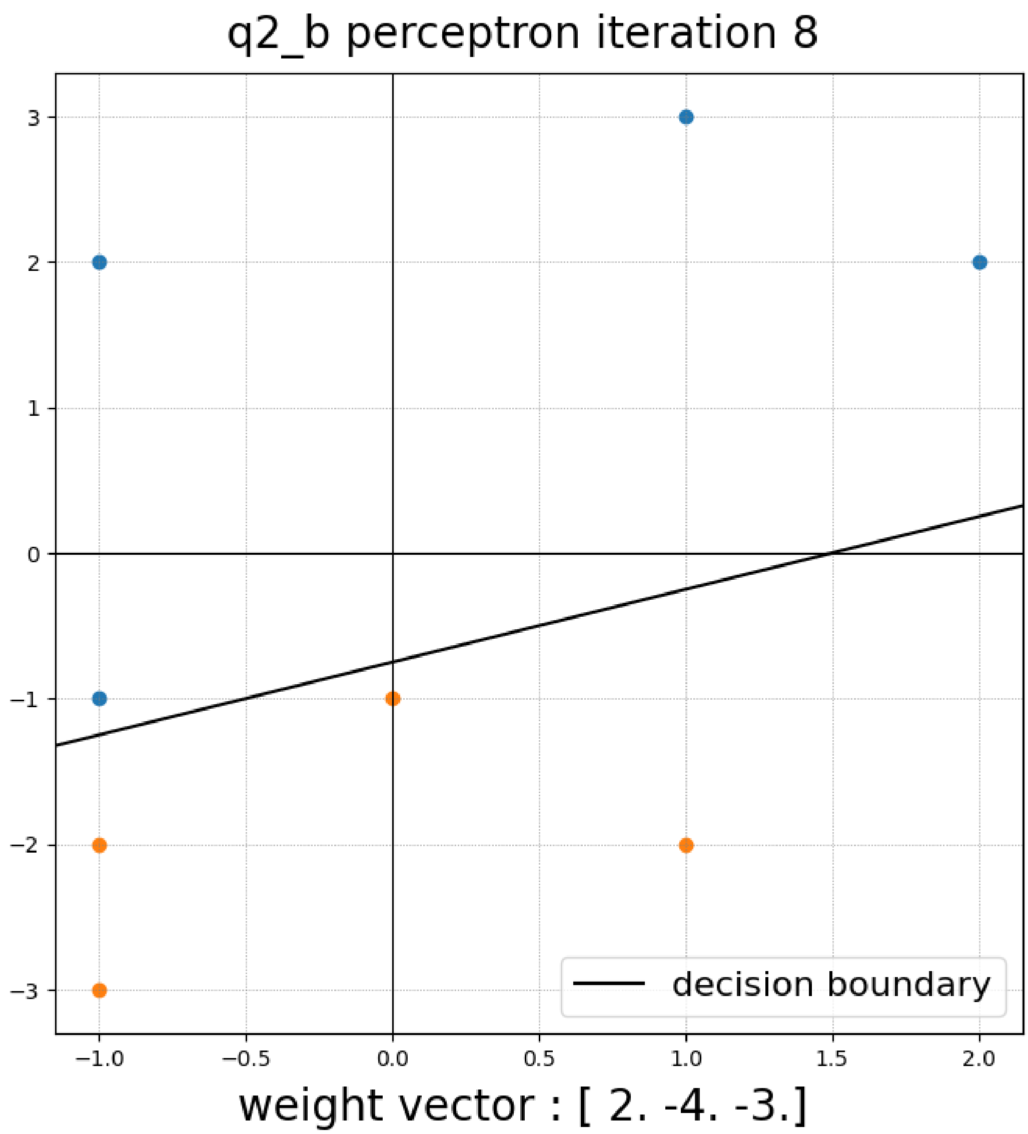

q2_b perceptron iteration 3
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0. -6. -1.]


q2_b perceptron iteration 4
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0.5 -5.5 -1.5]


q2_b perceptron iteration 5
decision boundary
weight vector : [1. -5. -2.]

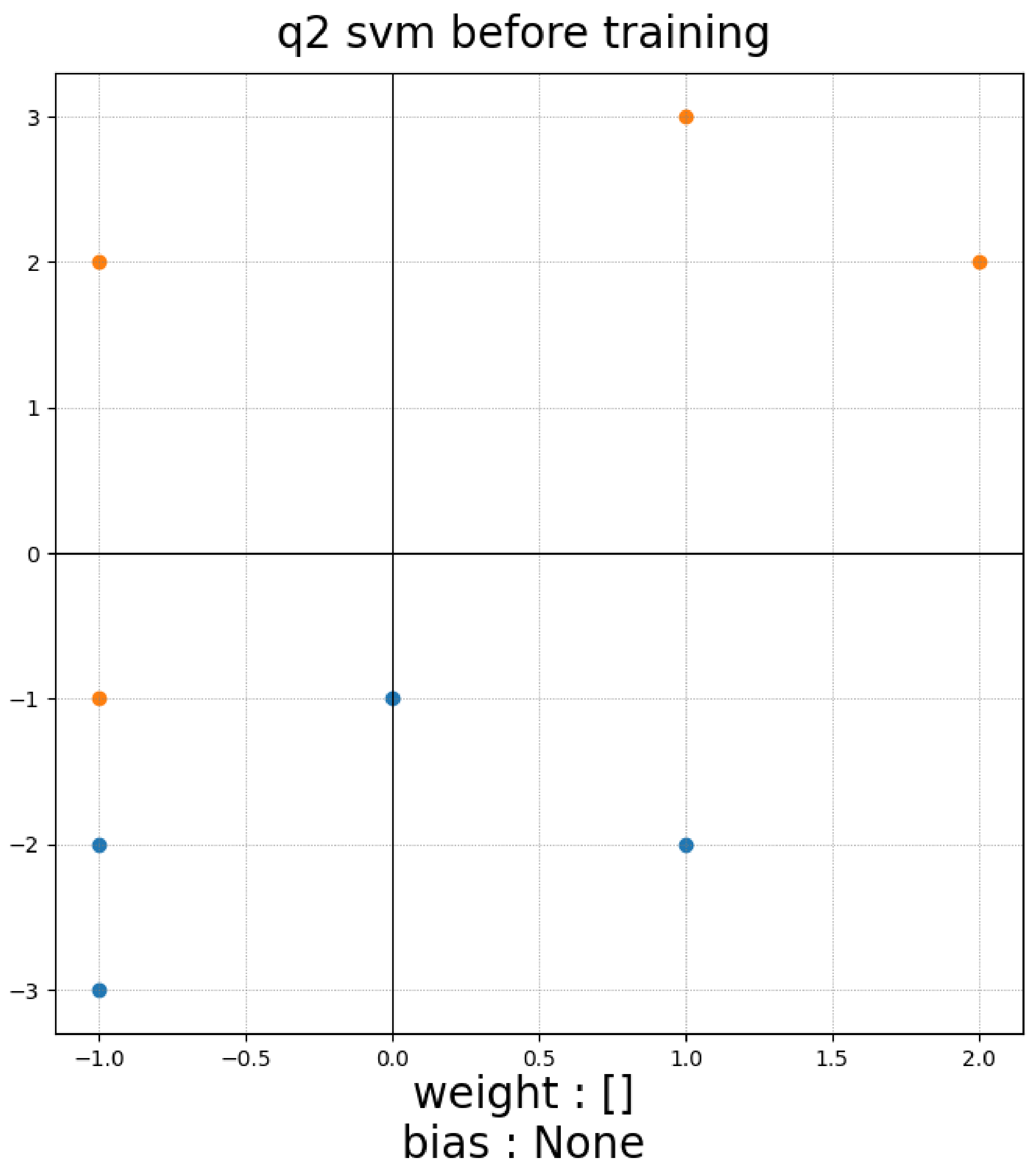

q2_b perceptron iteration 6
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [1.5 -4.5 -2.5]

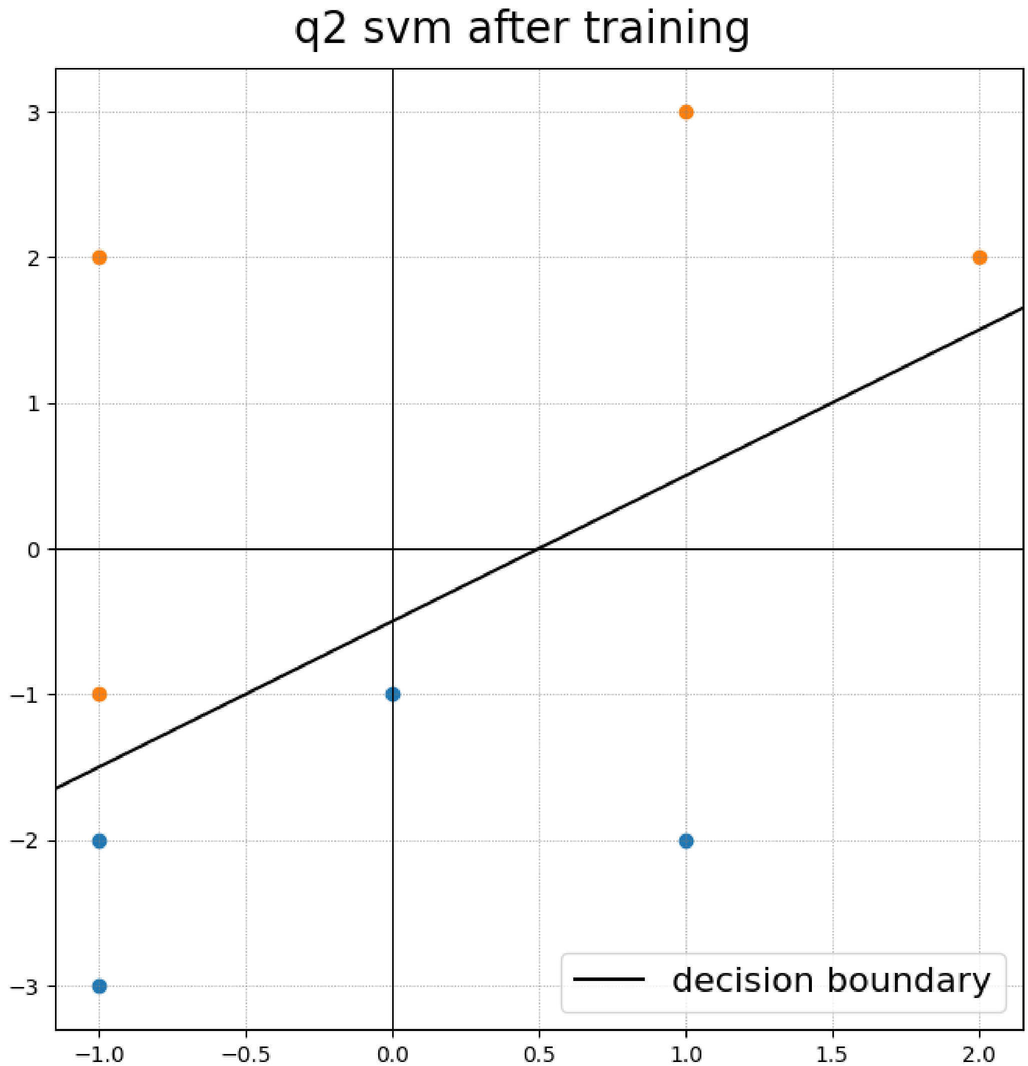

q2_b perceptron iteration 7
decision boundary
weight vector : [2. -4. -3.]



8 iterations for learning rate 0.5

SVM op:


q2 svm before training
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight : []
bias : None

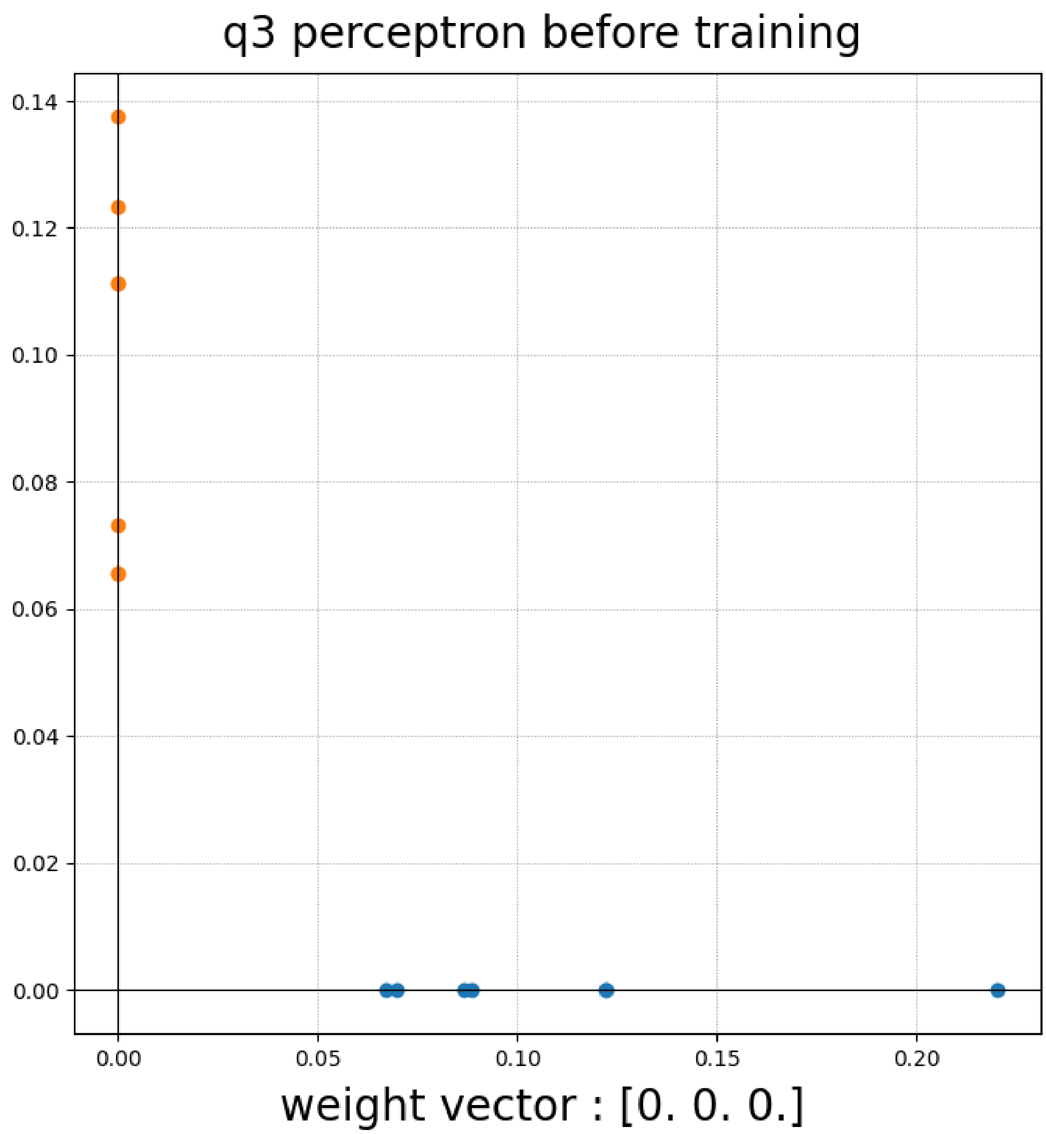


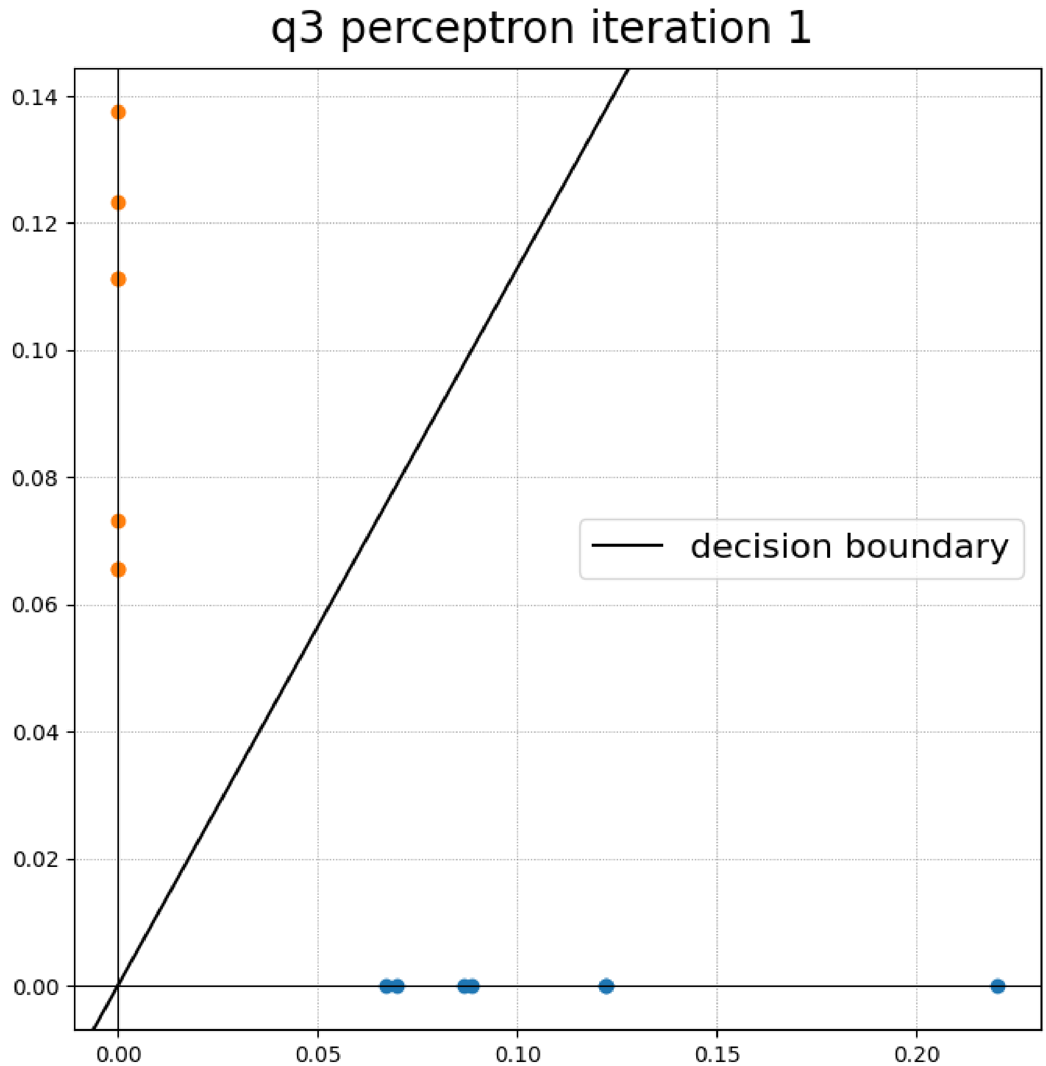
weight : [-2.00000023 2.00000031]
bias : 1.0000000792363046

Q3

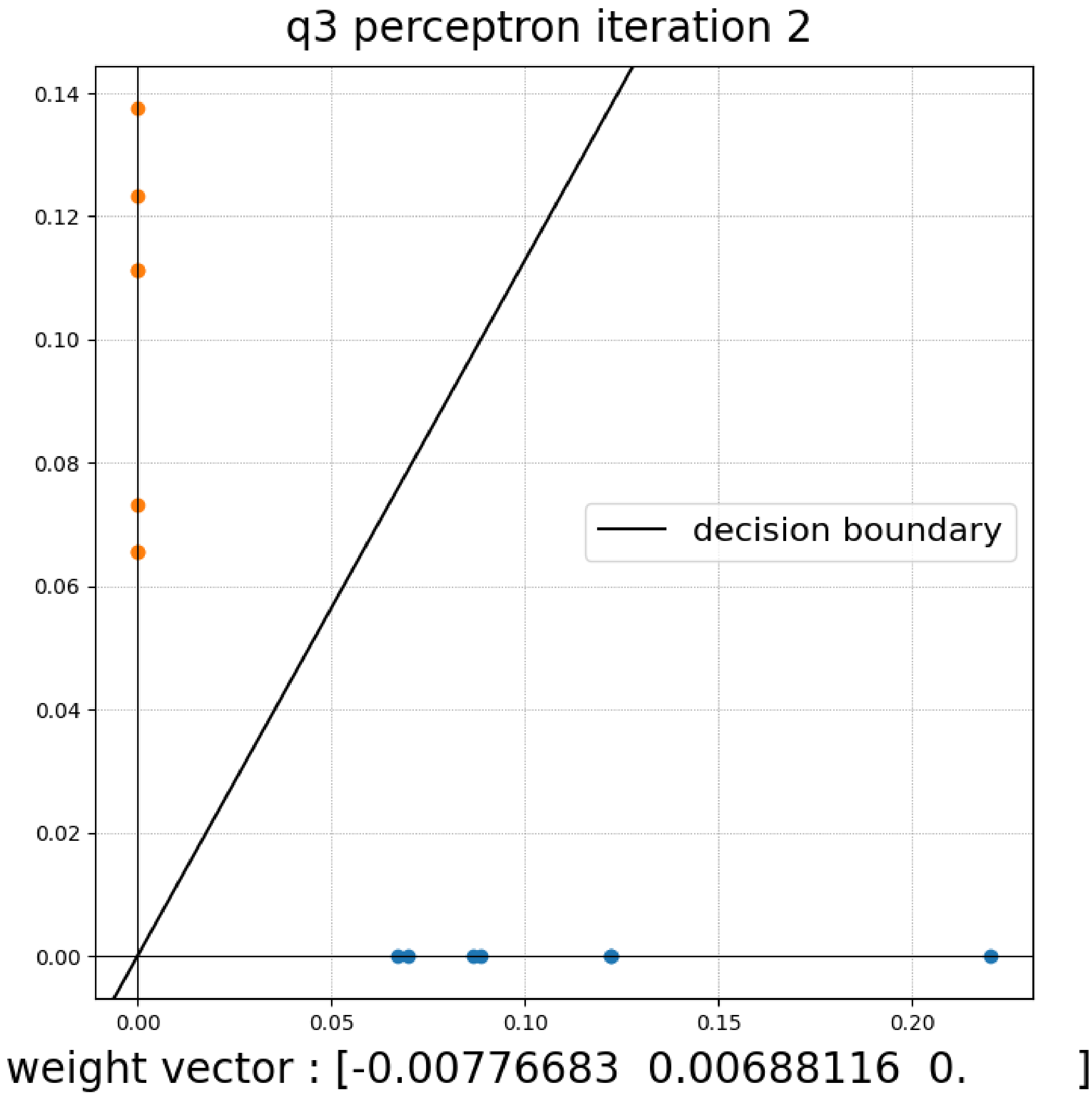
```
import numpy as np, cv2, svm, perceptron

# reading images
images=[]
for i in range(1,15):
    images.append(cv2.imread(f'poly_images/poly{i}.png'))

# for img in images:
#   cv2.imshow("image",img)
#   cv2.waitKey(0)

# choosing features:
def greenish_pixels(image):
    X,Y=image.shape[:2]
    value=0
    for x in range(X):
        for y in range(Y):
            b,g,r=image[x,y]
            # if int(g)>int(b)+int(r):
            if g>b and g>r:
                value+=1
    return value/X/Y

def reddish_pixels(image):
    X,Y=image.shape[:2]
    value=0
    for x in range(X):
        for y in range(Y):
            b,g,r=image[x,y]
            # if int(g)>int(b)+int(r):
            if r>b and r>g:
                value+=1
    return value/X/Y

x1=[]
```

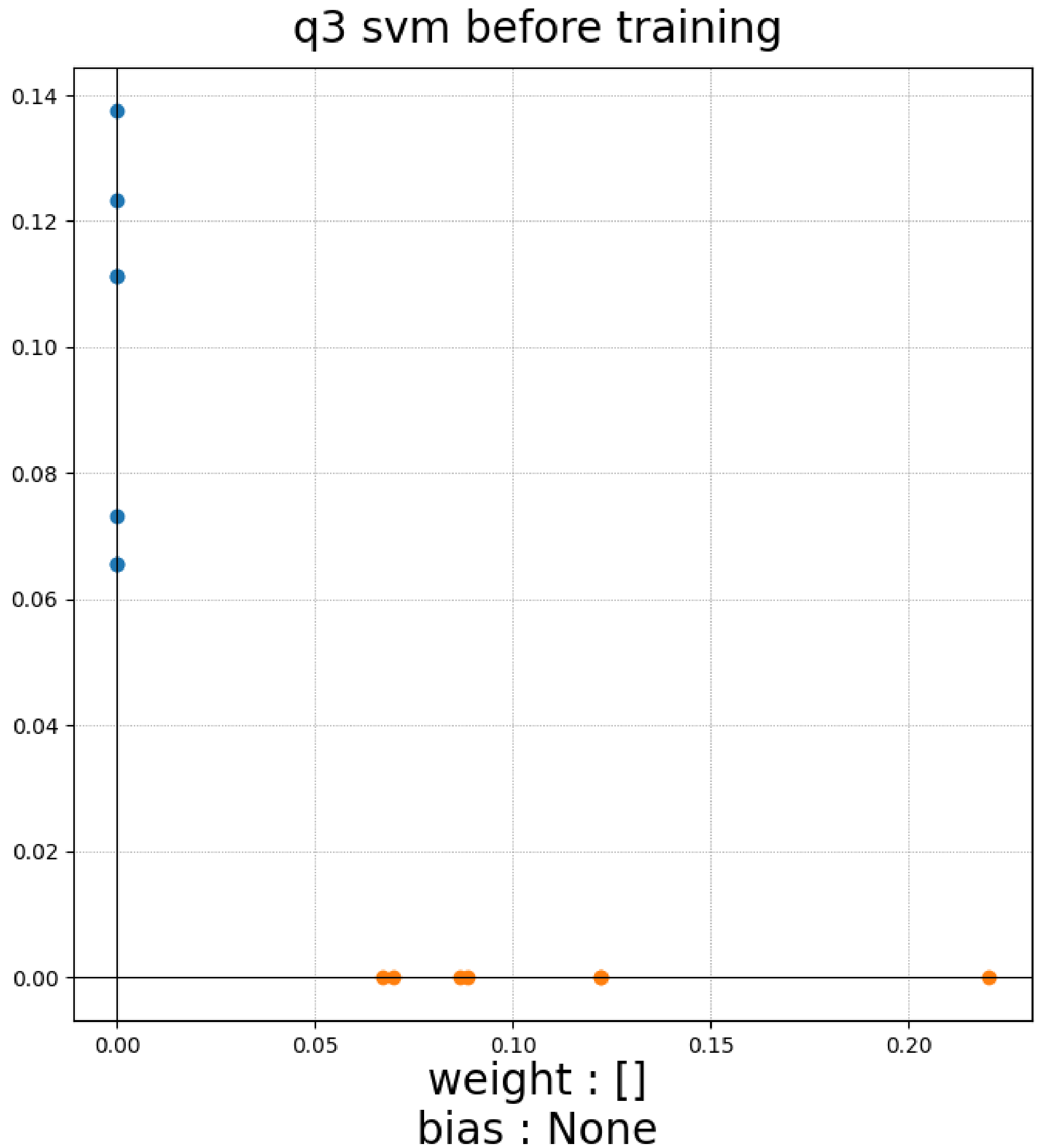
```
x2=[]

# extracting features:
for image in images:
    x1.append(greenish_pixels(image))
    x2.append(reddish_pixels(image))

# training
X=np.array(list(zip(x1,x2)))
Y_perc=np.array([1,1,1,1,1,1,1, 0, 0, 0, 0, 0, 0, 0])
Y_svm =np.array([1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1])

svm.demo(X,Y_svm,plot_title="q3")

perceptron.demo(X,Y_perc,learning_rate=0.01,plot_title="q3")
```

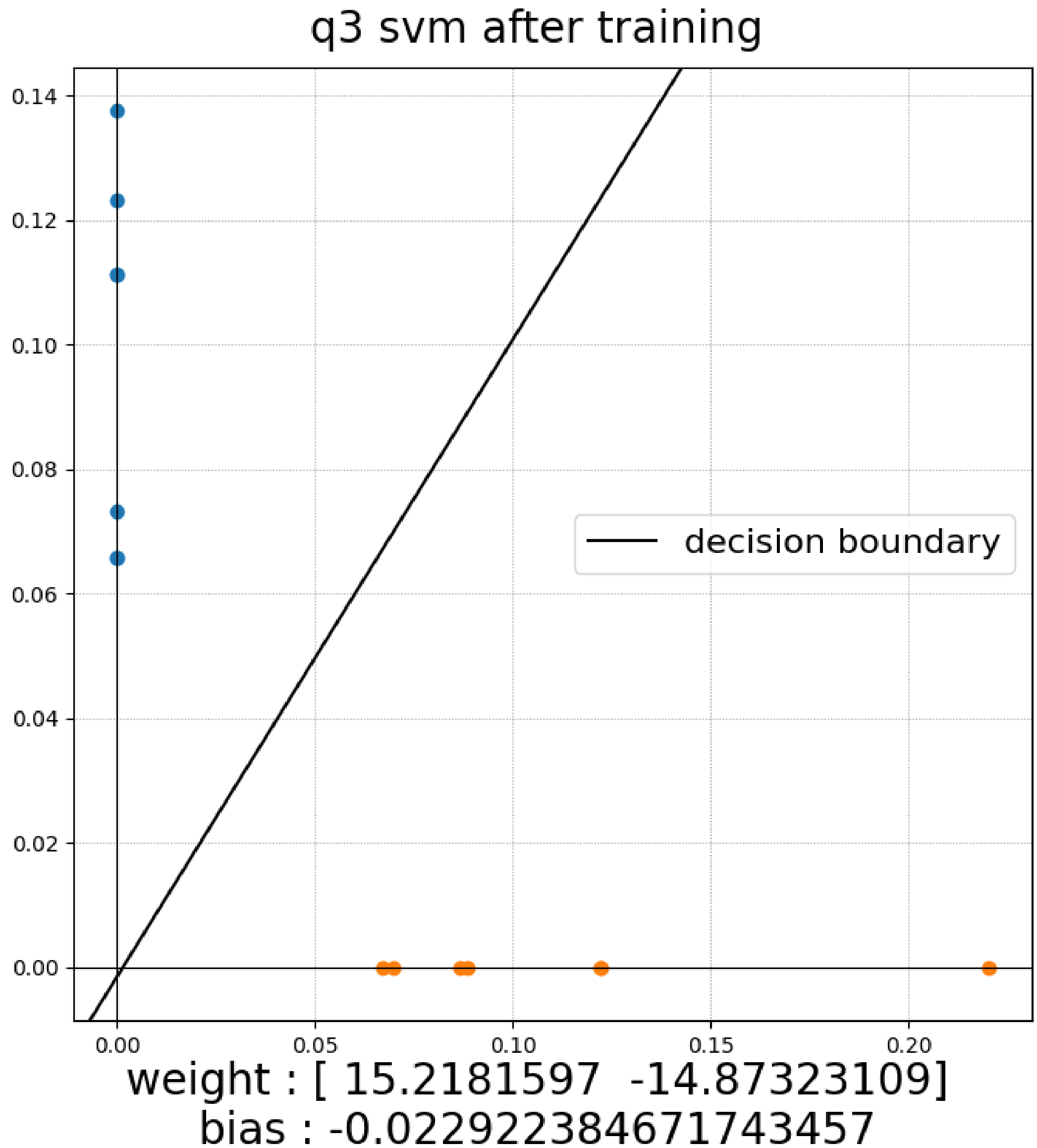

q3 perceptron before training
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.05
0.10
0.15
0.20
weight vector : [0. 0. 0.]



weight vector : [-0.00776683 0.00688116 0.   ]


q3 perceptron iteration 2
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.05
0.10
0.15
0.20
decision boundary
weight vector : [-0.00776683 0.00688116 0.   ]


q3 svm before training
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.05
0.10
0.15
0.20
weight : []
bias : None


q3 svm after training
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.05
0.10
0.15
0.20
decision boundary
weight : [15.2181597 -14.87323109]
bias : -0.022922384671743457