ASSIGNMENT-4

PATTERN RECOGNITION

R SHREJA COE18B043

RITHIC KUMAR N COE18B044

SREEDHAR ARUMUGAM COE18B051

05.05.2021

Train a single perceptron and SVM to learn an AND gate with two inputs x1 and x2. Assume that all the weights of the perceptron are initialized as 0. Show the calculation for each step and also draw the decision boundary for each updation.

In [ ]:

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

In [ ]:

```
from sympy.plotting import plot_implicit
from sympy import symbols
```

In [ ]:

```
# Append 1 to vector and make it -ve for one class (say 0)

def append_vector(df,class_1,class_2,pos_to_append): #assumption class label is last co
lumn , two features
    ext_vect=[]
    for _,row in df.iterrows():
        if row[df.columns[-1]]==int(class_1):
            ext_vect.append(-1)
            row[df.columns[0]]=-1* row[df.columns[0]]
            row[df.columns[1]]=-1* row[df.columns[1]]
        else:
            ext_vect.append(1)

    df.insert(pos_to_append,"bias",ext_vect,True)
```

In [ ]:

```
def move_sympyplot_to_axes(p, ax):
    backend = p.backend(p)
    backend.ax = ax
    backend._process_series(backend.parent._series, ax, backend.parent)
    backend.ax.spines['right'].set_color('none')
    backend.ax.spines['bottom'].set_position('zero')
    backend.ax.spines['top'].set_color('none')
    plt.close(backend.fig)
```

In [ ]:

```
def plot_eqn_and_points(df_train,weights): #works only for 2-feature # plot the origina
l points
#     df_train.plot.scatter(x=df_train.columns[0],y=df_train.columns[1])

    y,x = symbols('y x')
    eqn=y*weights[0]+x*weights[1]+weights[2]
    graph=plot_implicit(eqn, (x, -3, 3), (y, -3, 3),show=False)
    fig, ax = plt.subplots()

    move_sympyplot_to_axes(graph, ax)
    plt.setp(ax.yaxis.get_label(), 'rotation', 0)


    plt.scatter(df_train[df_train.columns[0]], df_train[df_train.columns[1]],c=df_train
[df_train.columns[-1]])
    plt.show()
#     plt.show()
#     plt.scatter(df_versicolor['petal.length'], df_versicolor['petal.width'],label="Ve
rsicolor")
```

In [ ]:

```
def perceptron_iter(df_train,learning_rate=1,weights=None):
    n=len(df_train)
    k=0
    count=0
    all_weights=[]
    if weights == None:
        weights=np.zeros(df_train.shape[1])
    weights=np.array(weights)
    while count != n:
        if weights.T @ np.array(df_train.iloc[k]) > 0:
            count+=1
        else:
            count=0
            weights += learning_rate * df_train.iloc[k]
            count+=1

        k=(k+1)%n
        print(weights)
        all_weights.append(weights.copy())
        # new_list = old_list.copy()
        # plot_eqn_and_points(df_train,weights)
    return weights,all_weights
```

In [ ]:

```
df_list=[[0,0,0],[0,1,0],[1,0,0],[1,1,1]]
```

In [ ]:

```
df_org=pd.DataFrame(df_list,columns=['x1','x2','output'])
```

In [ ]:

```
df=pd.DataFrame(df_list,columns=['x1','x2','output'])
```

In [ ]:

```
df
```

Out[ ]:

	x1	x2	output
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

In [ ]:

```
append_vector(df,0,1,2)
```

In [ ]:

```
df
```

Out[ ]:

	x1	x2	bias	output
0	0	0	-1	0
1	0	-1	-1	0
2	-1	0	-1	0
3	1	1	1	1

In [ ]:

```
df_train=df[df.columns[:-1]]
```

In [ ]:

```
df_train
```

Out[ ]:

	x1	x2	bias
0	0	0	-1
1	0	-1	-1
2	-1	0	-1
3	1	1	1

In [ ]:

```
w,w_a=perceptron_iter(df_train,1,weights=None)
```

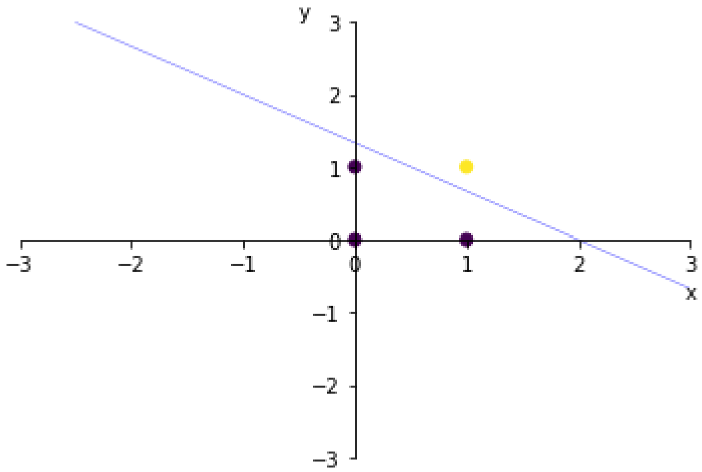
```
x1      0.0
x2      0.0
bias   -1.0
Name: 0, dtype: float64
x1      0.0
x2      0.0
bias   -1.0
Name: 0, dtype: float64
x1      0.0
x2      0.0
bias   -1.0
Name: 0, dtype: float64
x1      1.0
x2      1.0
bias    0.0
Name: 0, dtype: float64
x1      1.0
x2      1.0
bias   -1.0
Name: 0, dtype: float64
x1      1.0
x2      0.0
bias   -2.0
Name: 0, dtype: float64
x1      1.0
x2      0.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -1.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -1.0
Name: 0, dtype: float64
x1      2.0
x2      0.0
bias   -2.0
Name: 0, dtype: float64
x1      1.0
x2      0.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -2.0
Name: 0, dtype: float64
x1      1.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
```

```
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -2.0
Name: 0, dtype: float64
x1      3.0
x2      1.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      1.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -3.0
Name: 0, dtype: float64
x1      2.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      3.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      3.0
bias   -3.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
```

```
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
```

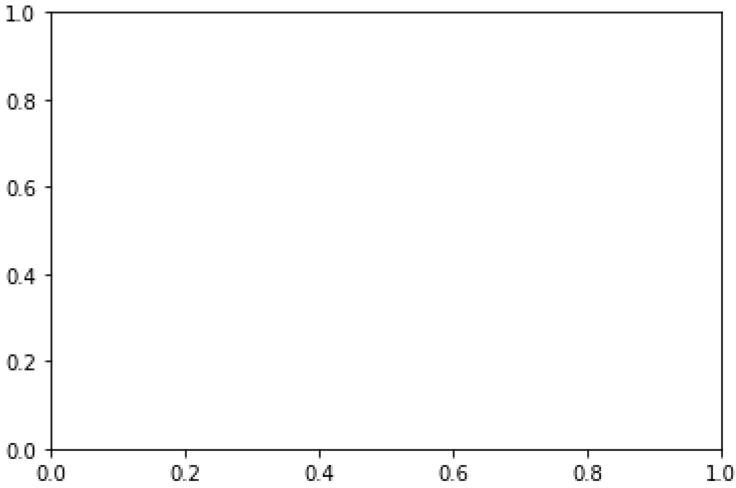
In [ ]:

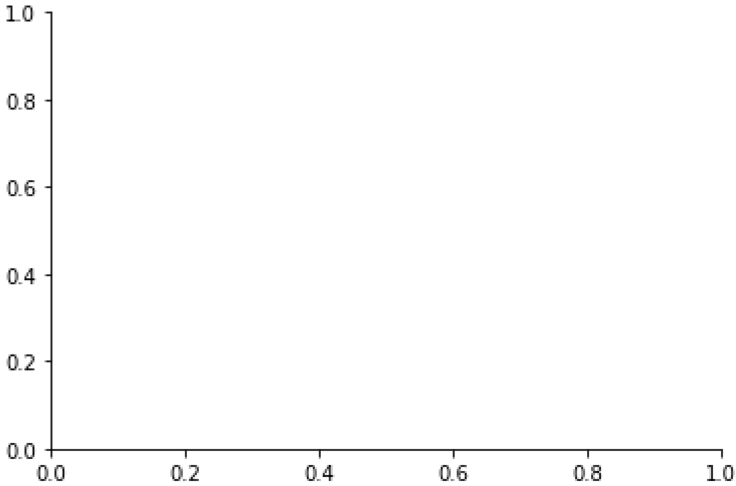
```
plot_eqn_and_points(df_org,w)
```

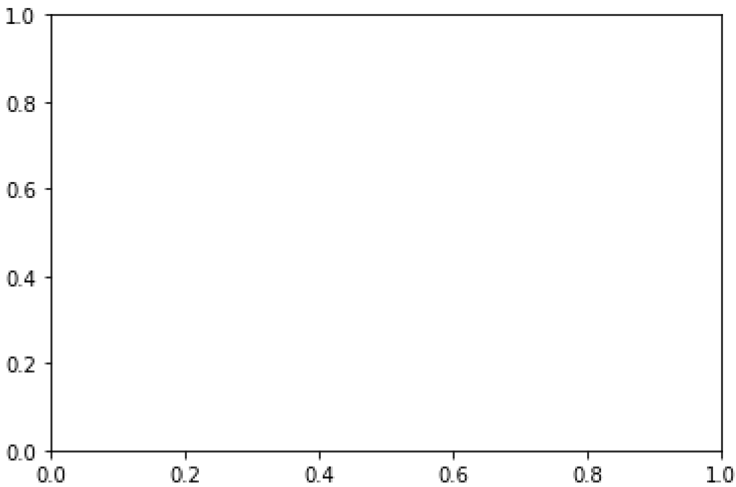


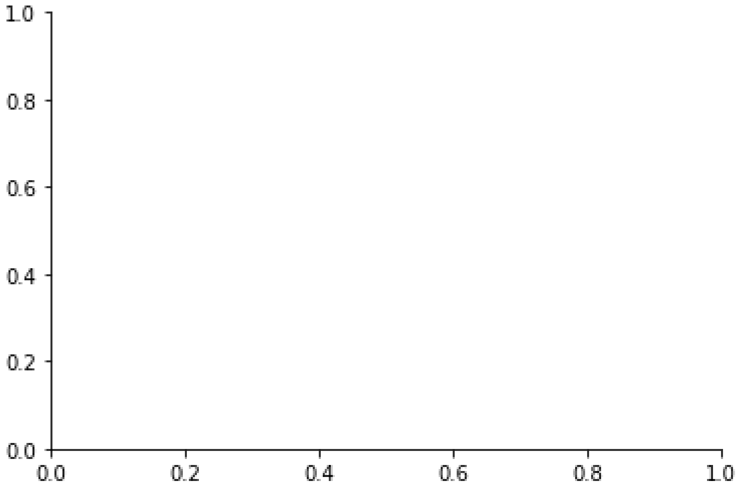
In [ ]:

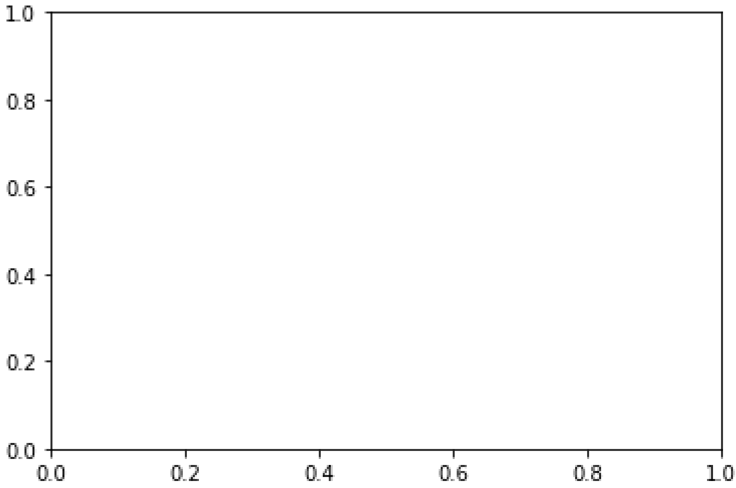
```
for i in w_a:
  try:
    plot_eqn_and_points(df_org,i)
  except:
    pass
```

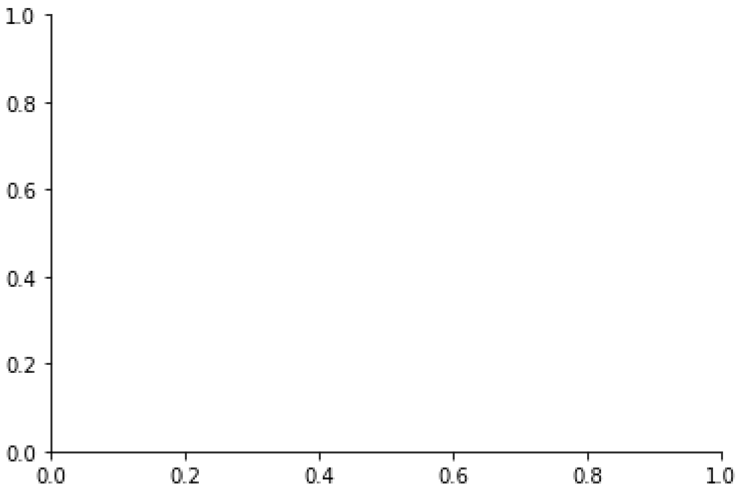

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

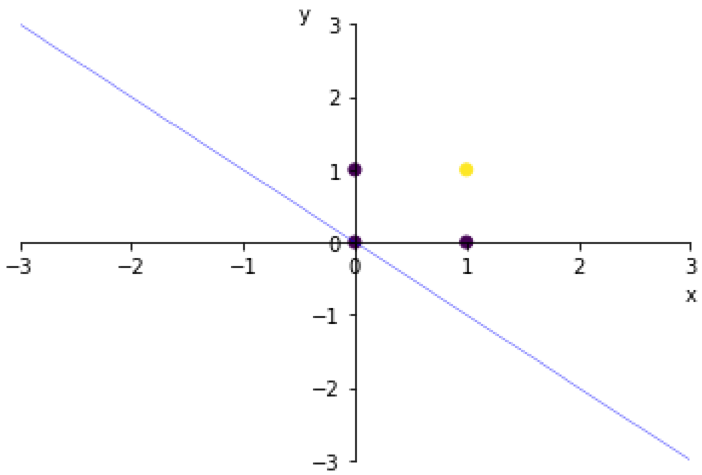

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

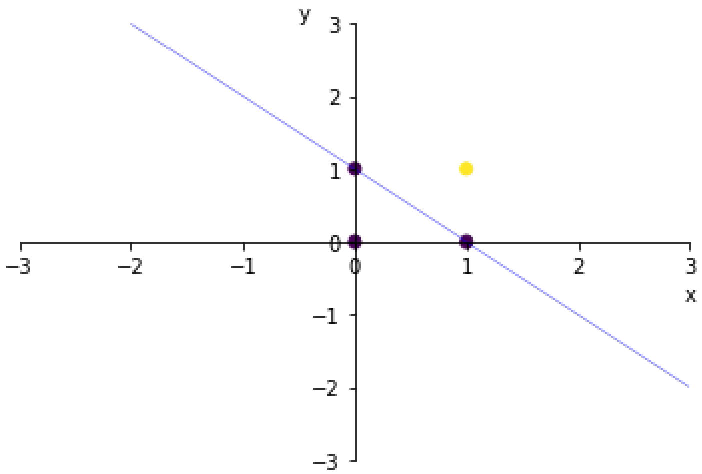

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

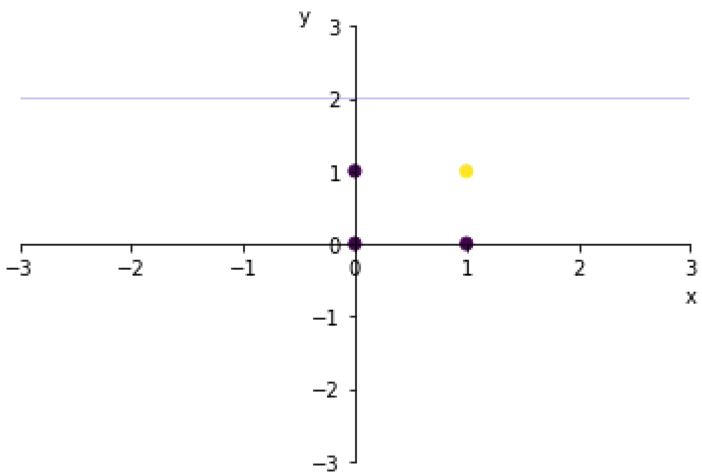

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



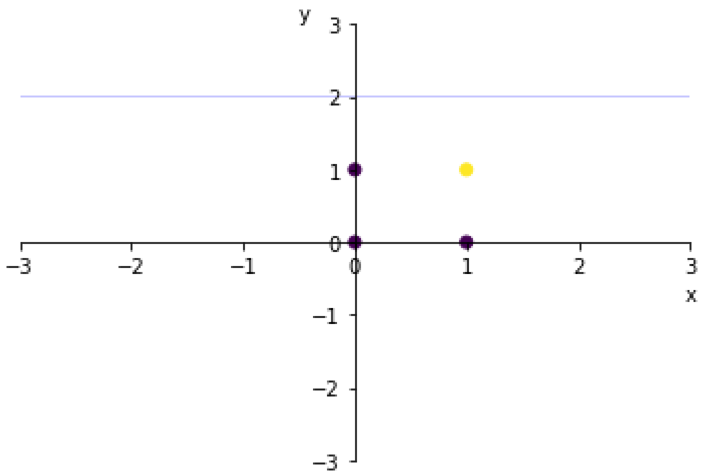
y
3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
x



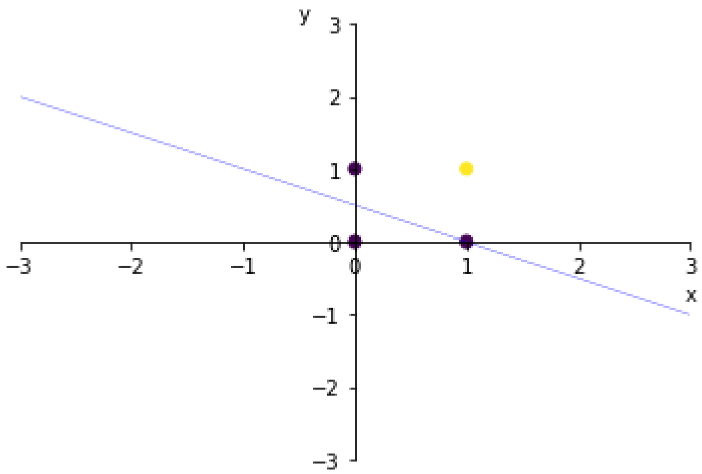
y
3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
x



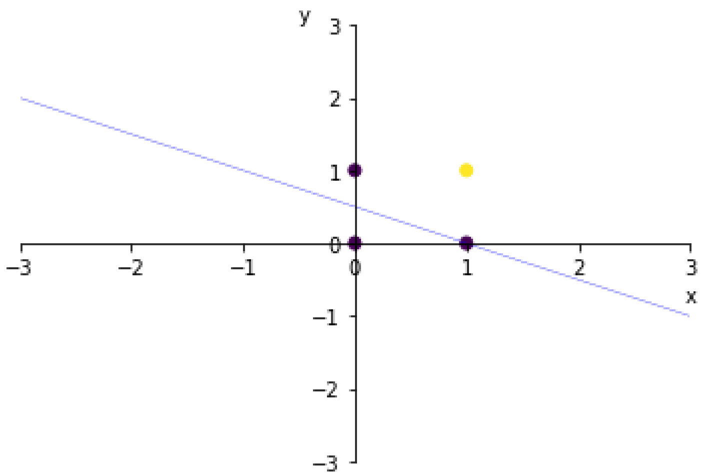
y
3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
x



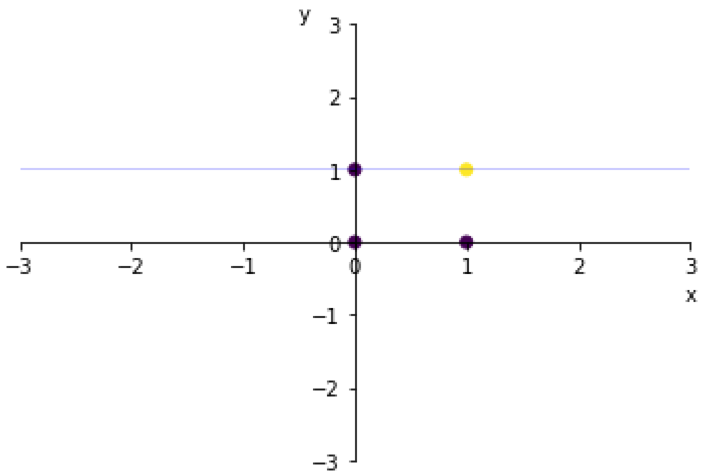
y
x



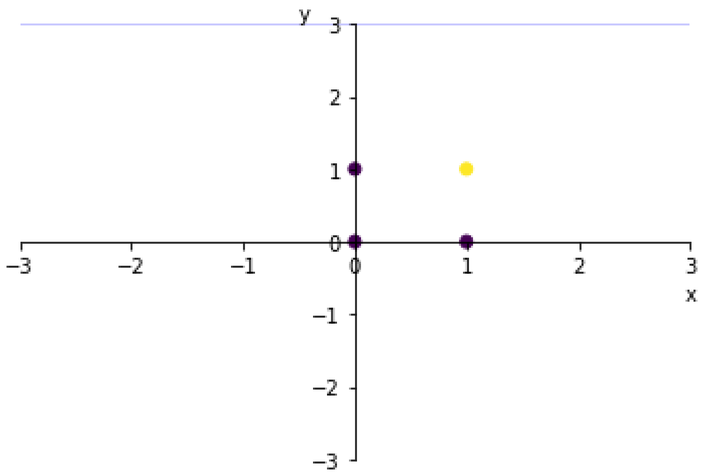
y
x



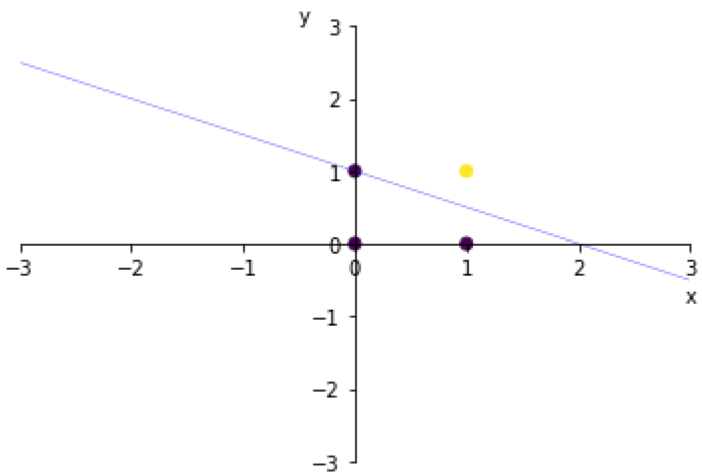
y
x



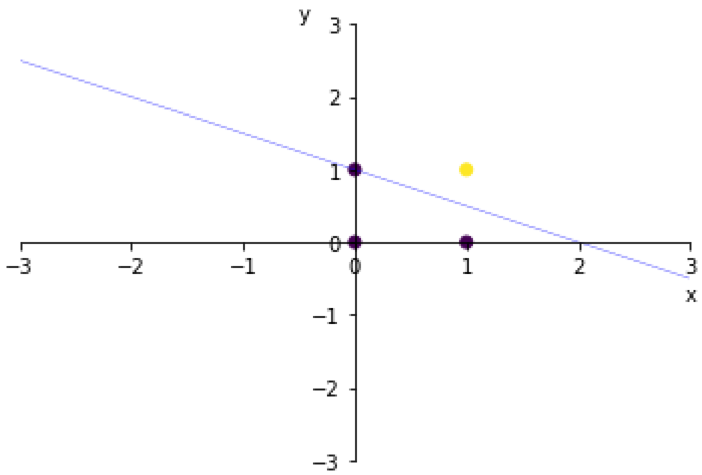
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x

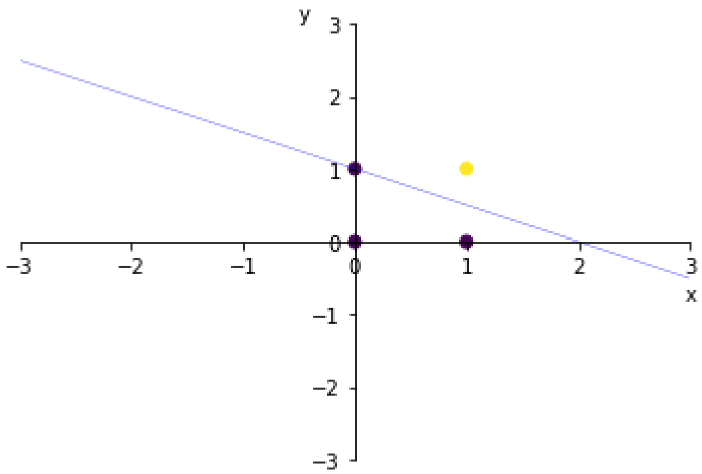


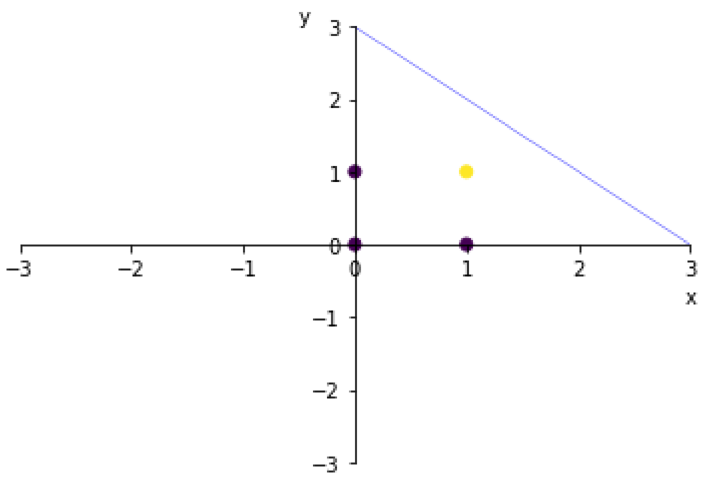
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x

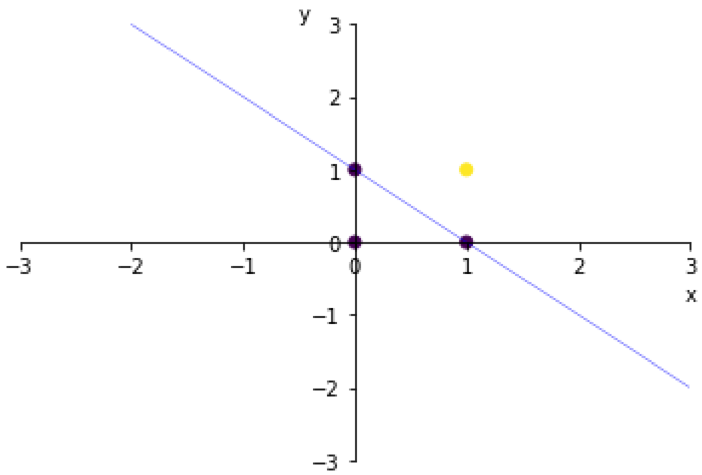


y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x

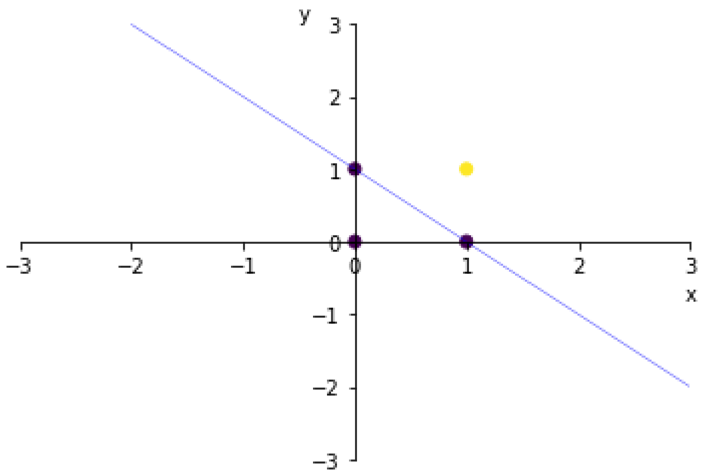

y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x


y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x

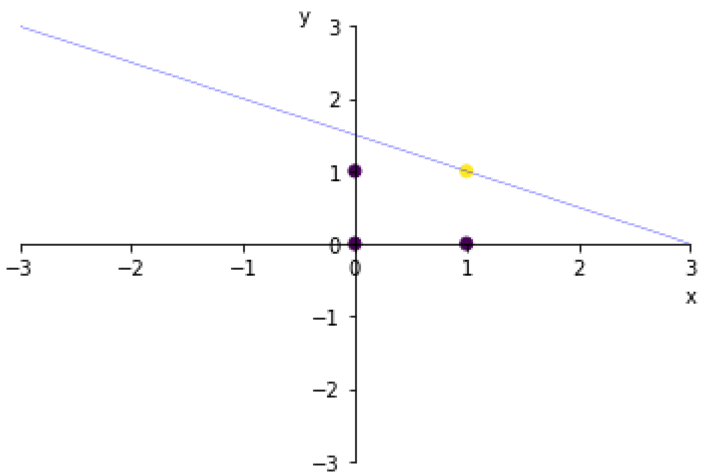

y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x



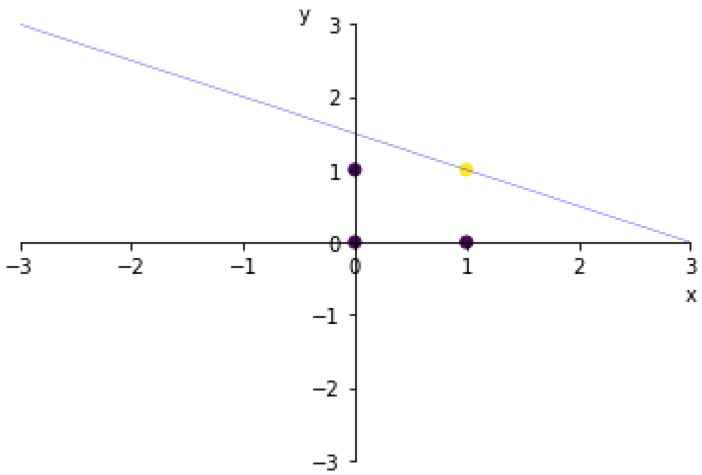
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x



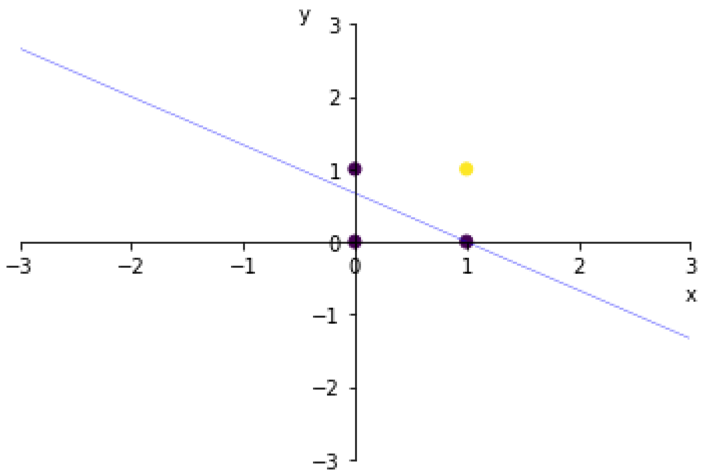
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x



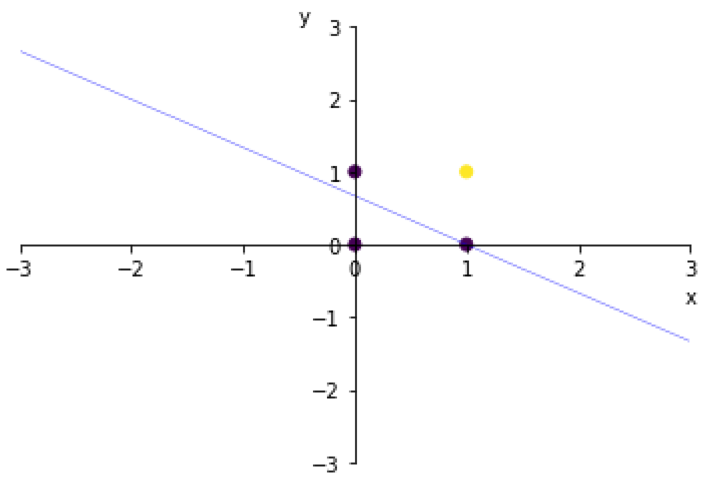
y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x



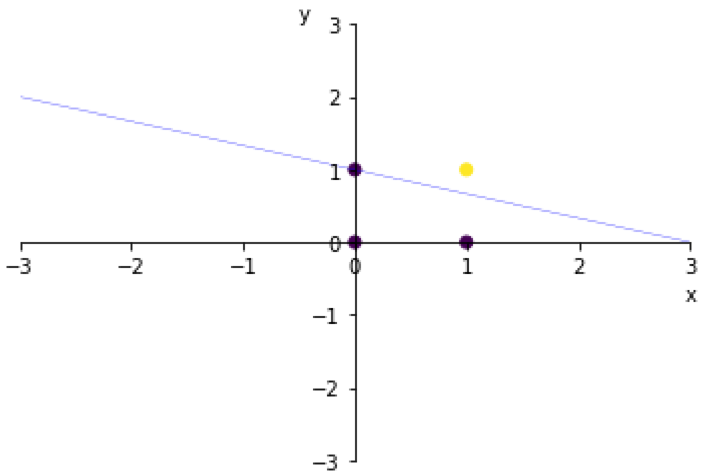
y
x
3
2
1
0
−1
−2
−3



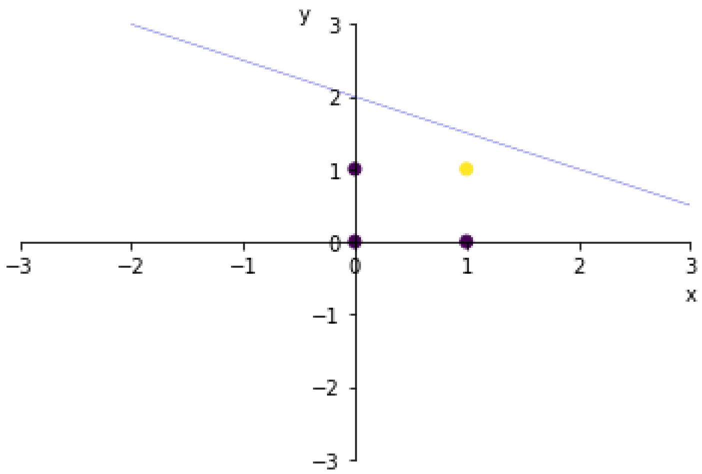
y
x
3
2
1
0
−1
−2
−3



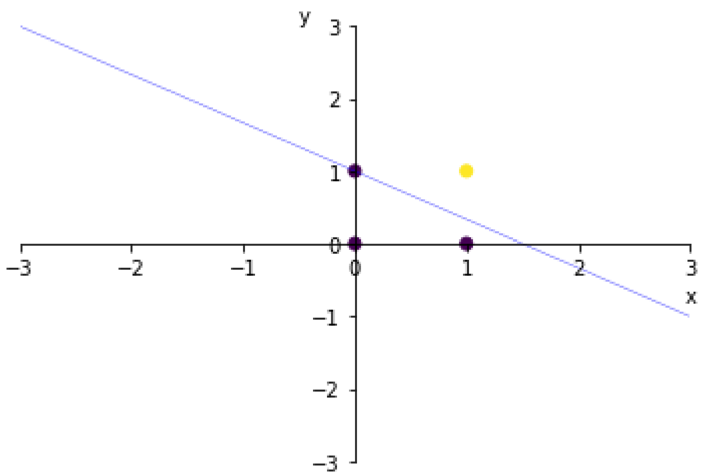
y
x
3
2
1
0
−1
−2
−3



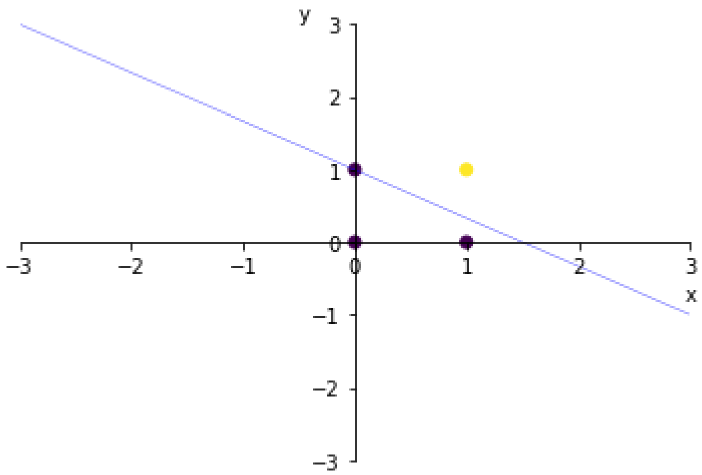
y
3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
x

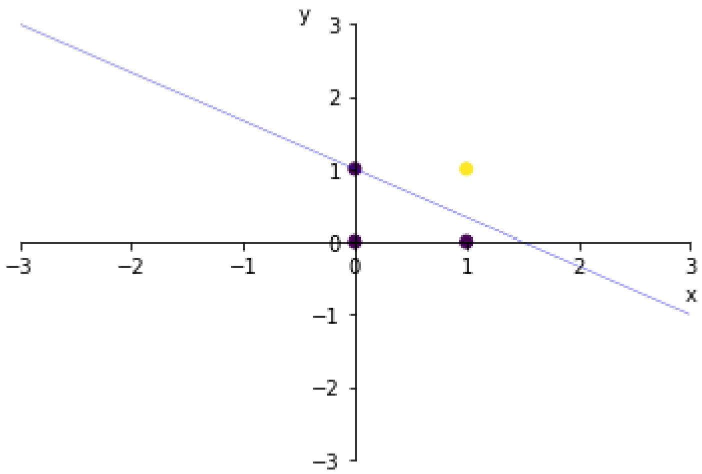


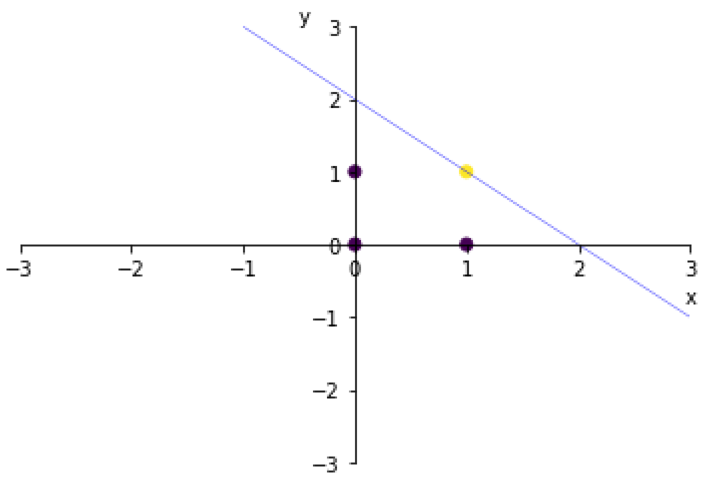
y
3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
x

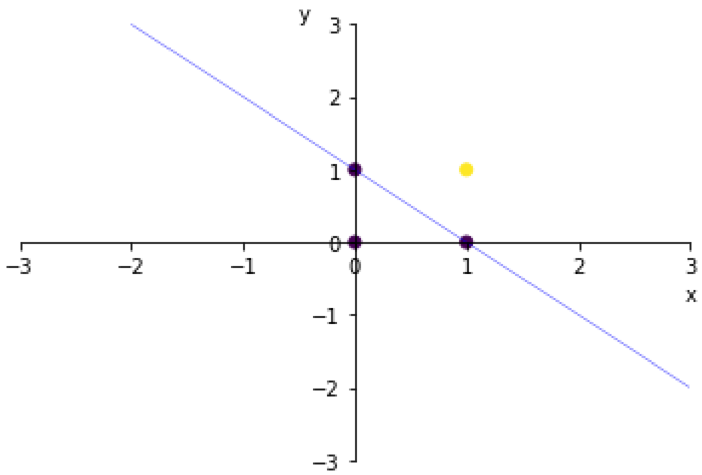


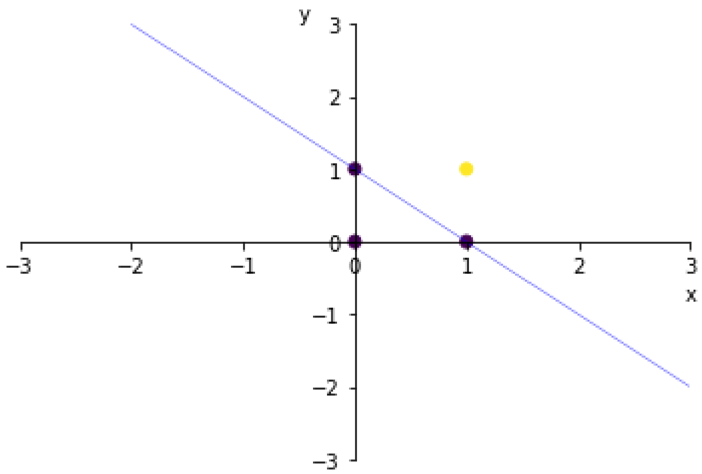
y
3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
x

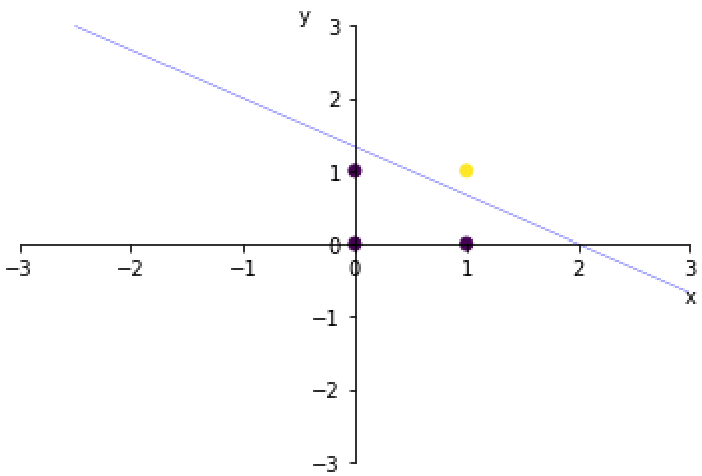

y
x
3
2
1
0
−1
−2
−3

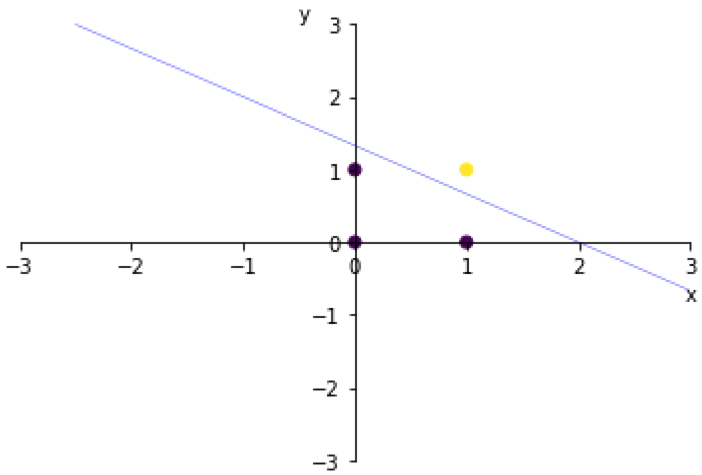

y
x
3
2
1
0
−1
−2
−3


y
x
3
2
1
0
−1
−2
−3

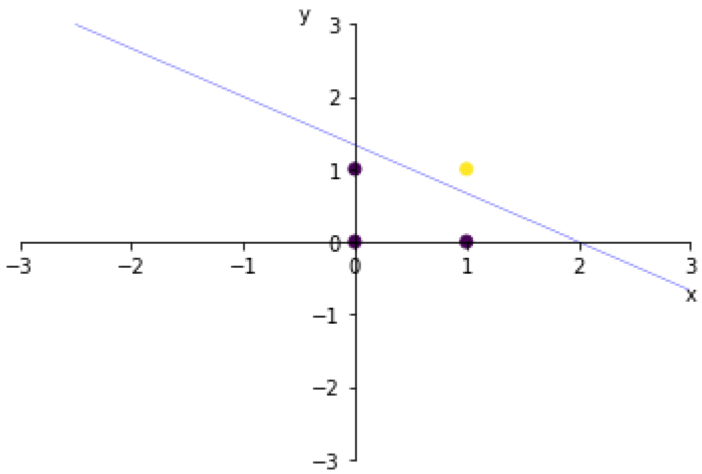

y
x


y
x

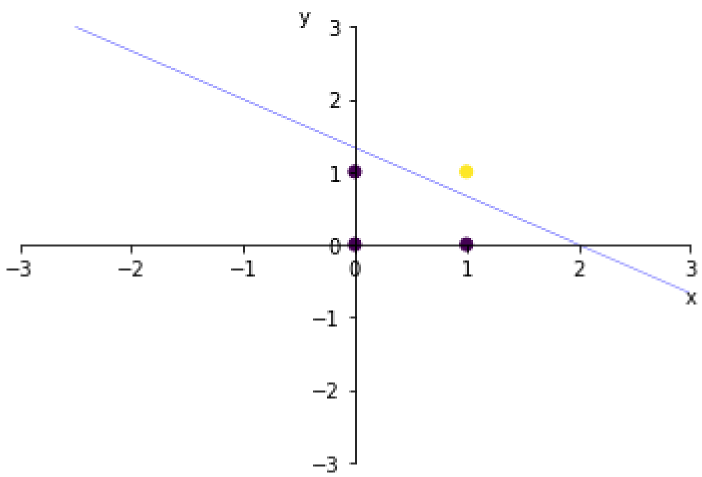

y
x



y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x



y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x



y
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
x

In [ ]:

```
w
```

Out[ ]:

```
x1      3.0
x2      2.0
bias   -4.0
Name: 0, dtype: float64
```

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)
```

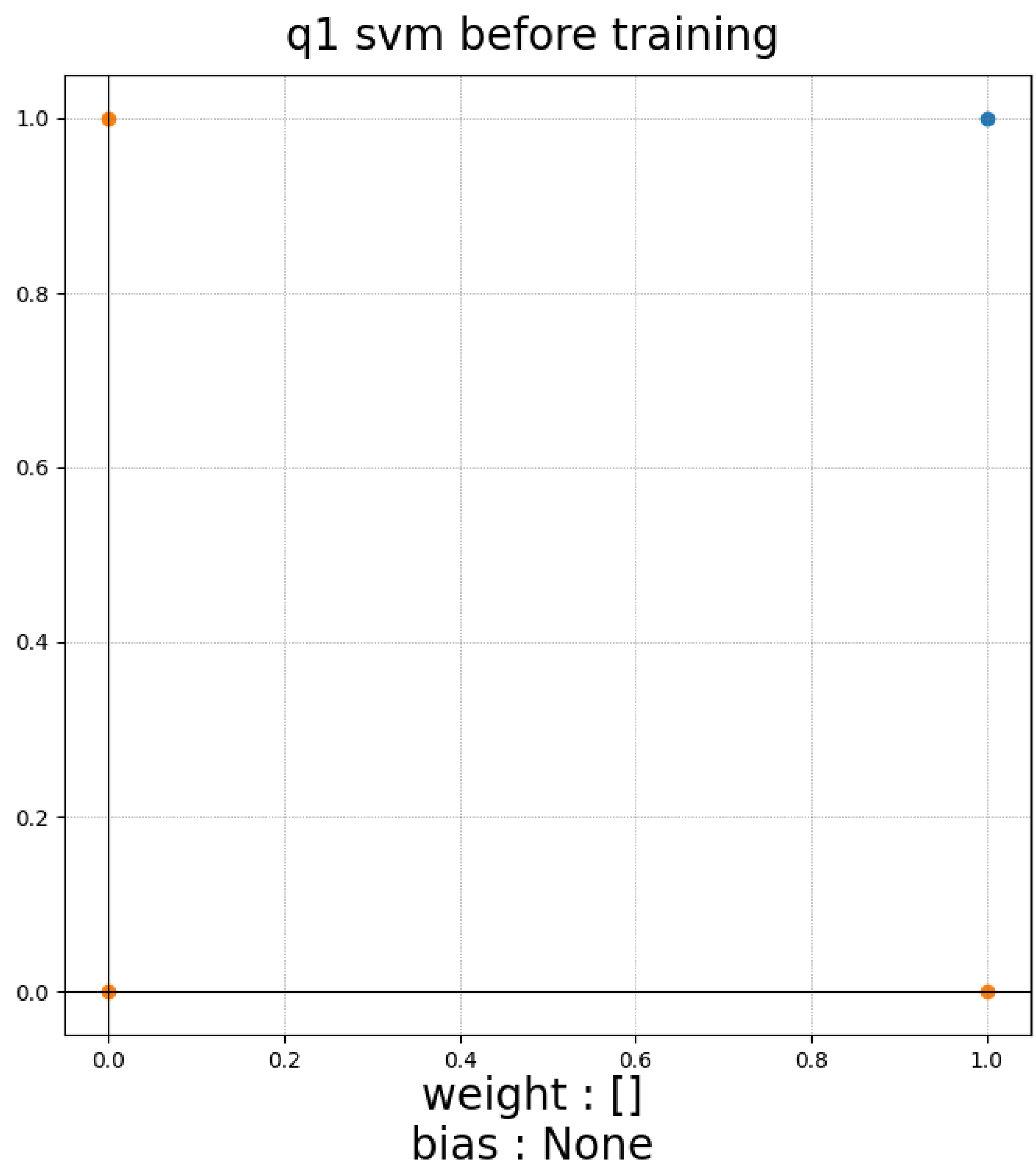
```
        for i,alpha in enumerate(alphas):
                self.__weight+=alpha*self.__Y[0][i]*self.__X[i]

        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
```

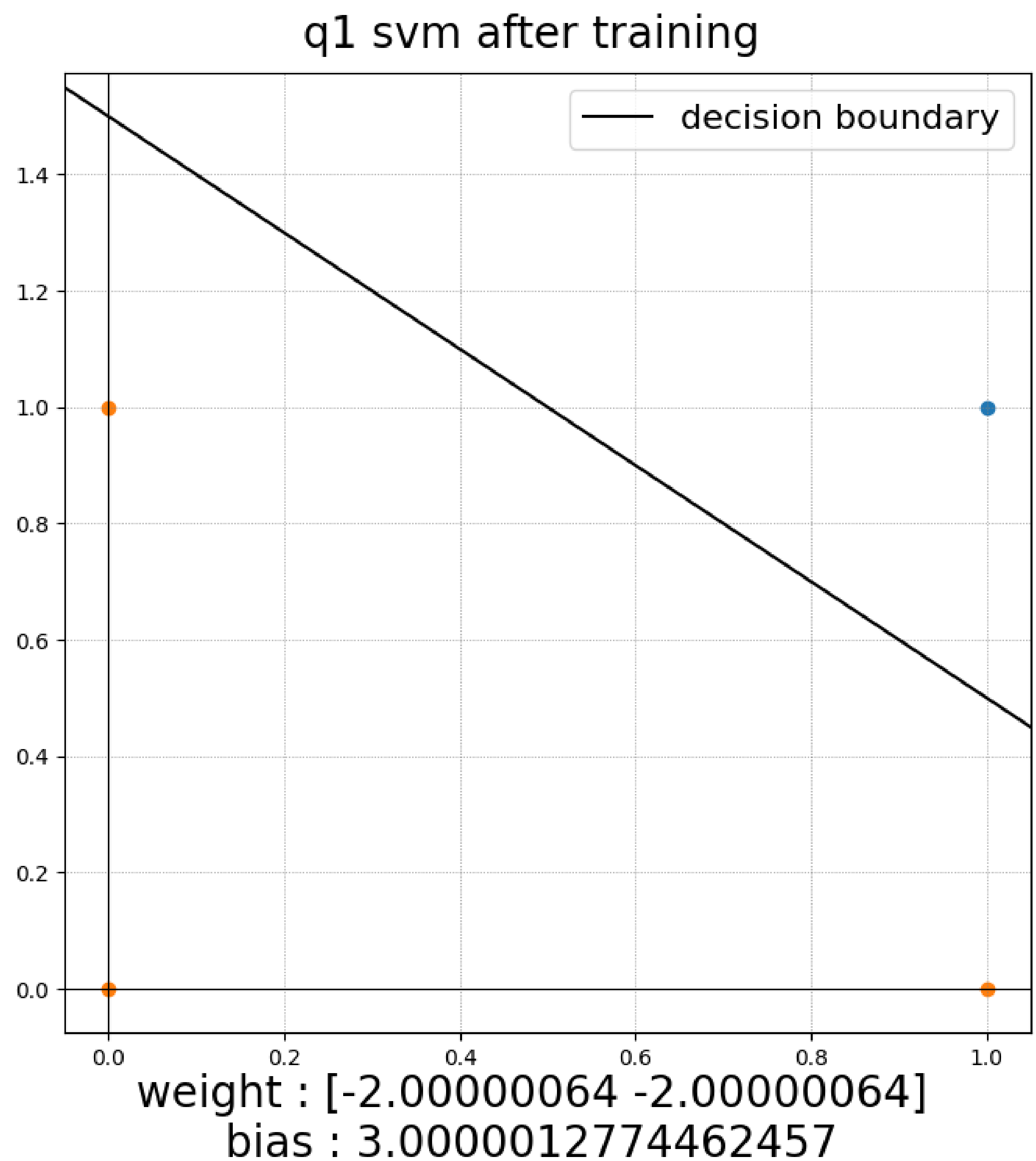
```
            if b==0:
                plt.axvline(-c/a, c='black', label='decision
boundary')
            else:
                y_intercept=-c/b
                slope=-a/b
                plt.axline((0,y_intercept), slope=slope, c='black',
label='decision boundary')
            plt.legend(loc='best',fontsize=16)

        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:
```

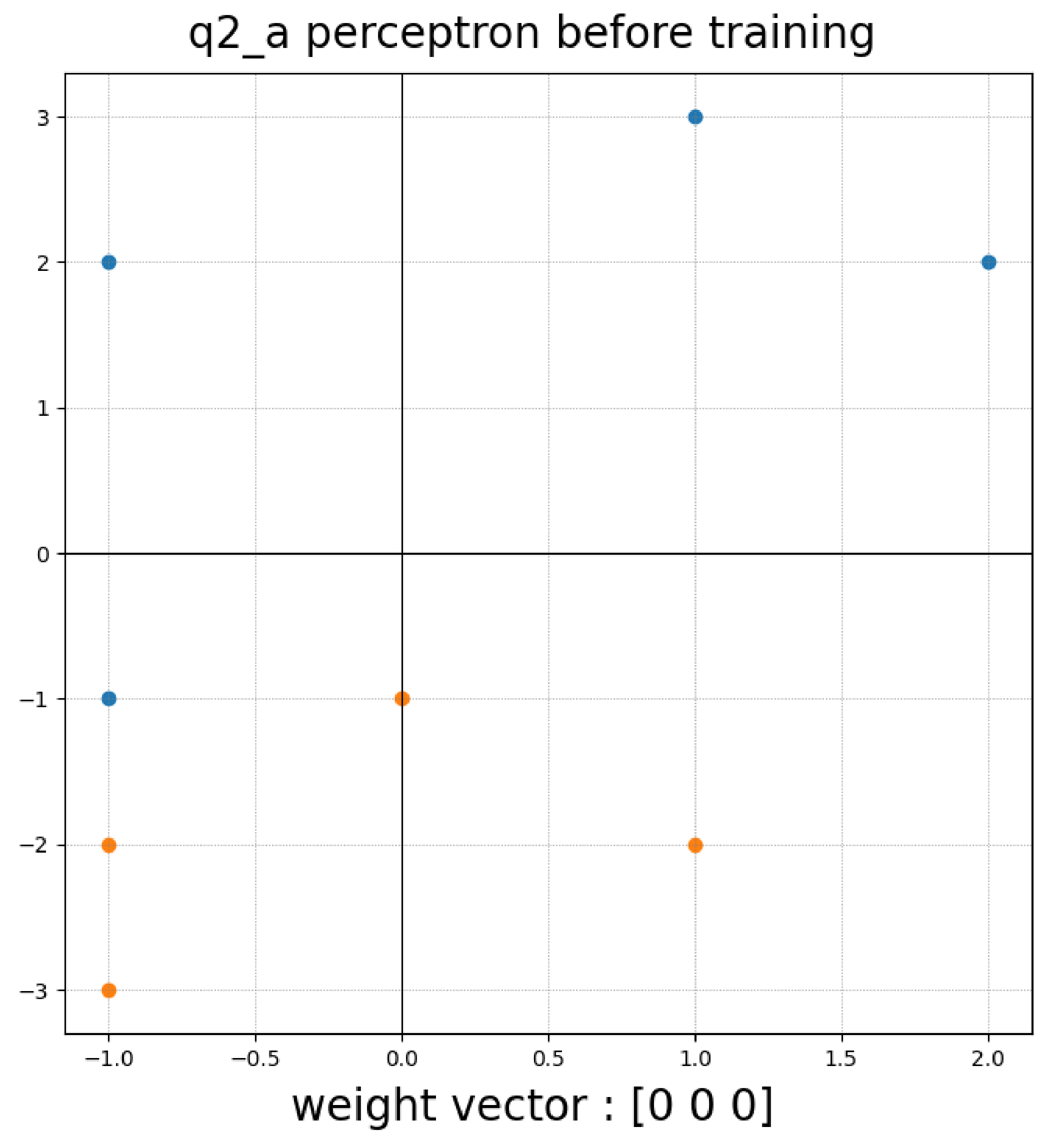
```
            raise ValueError(f"given weight vector must be of shape 1x(d+1),
1x{self.__weight.shape[0]} here")
        self.__weight=weight

    # 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')
```

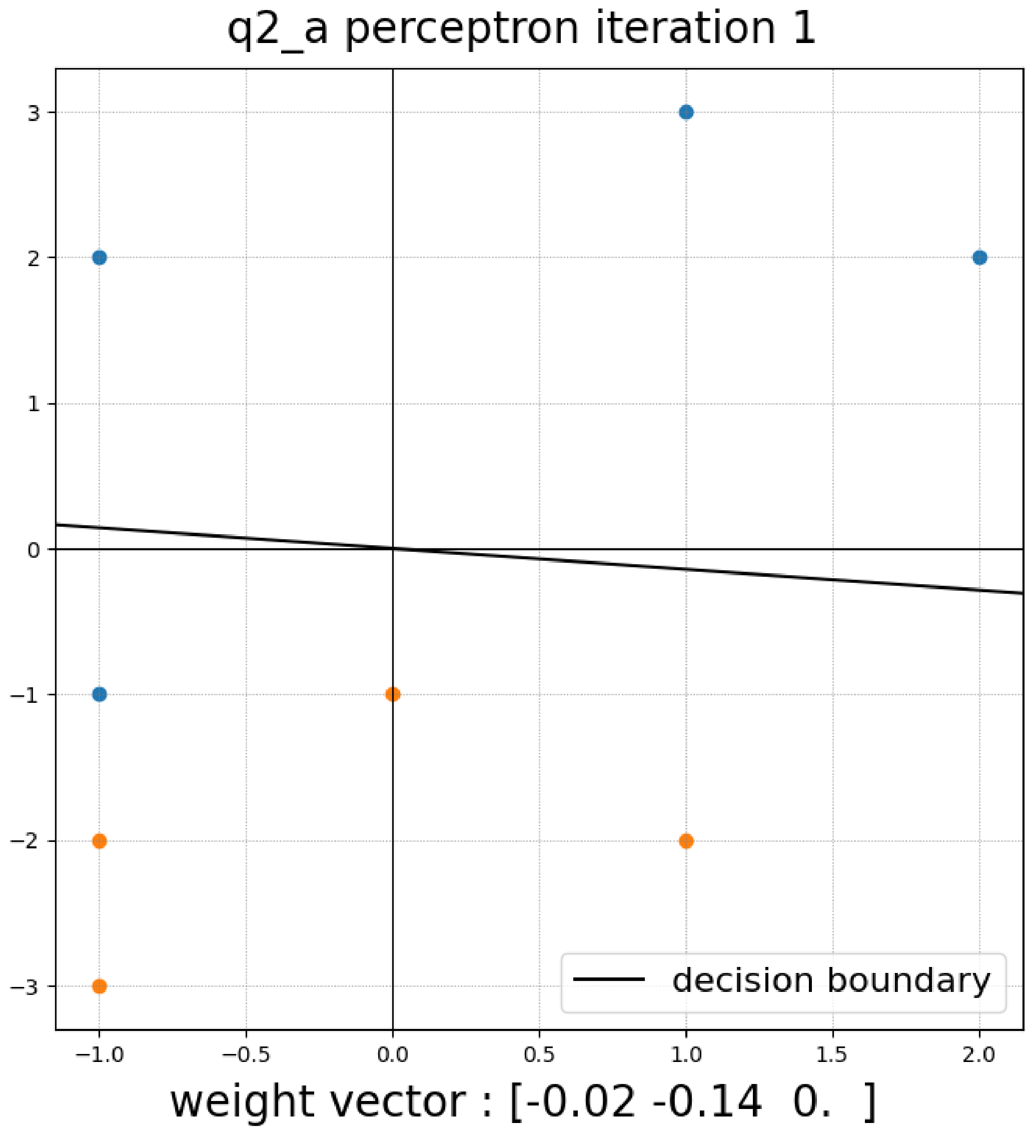
```
            plt.legend(loc='best',fontsize=16)


        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.")
```

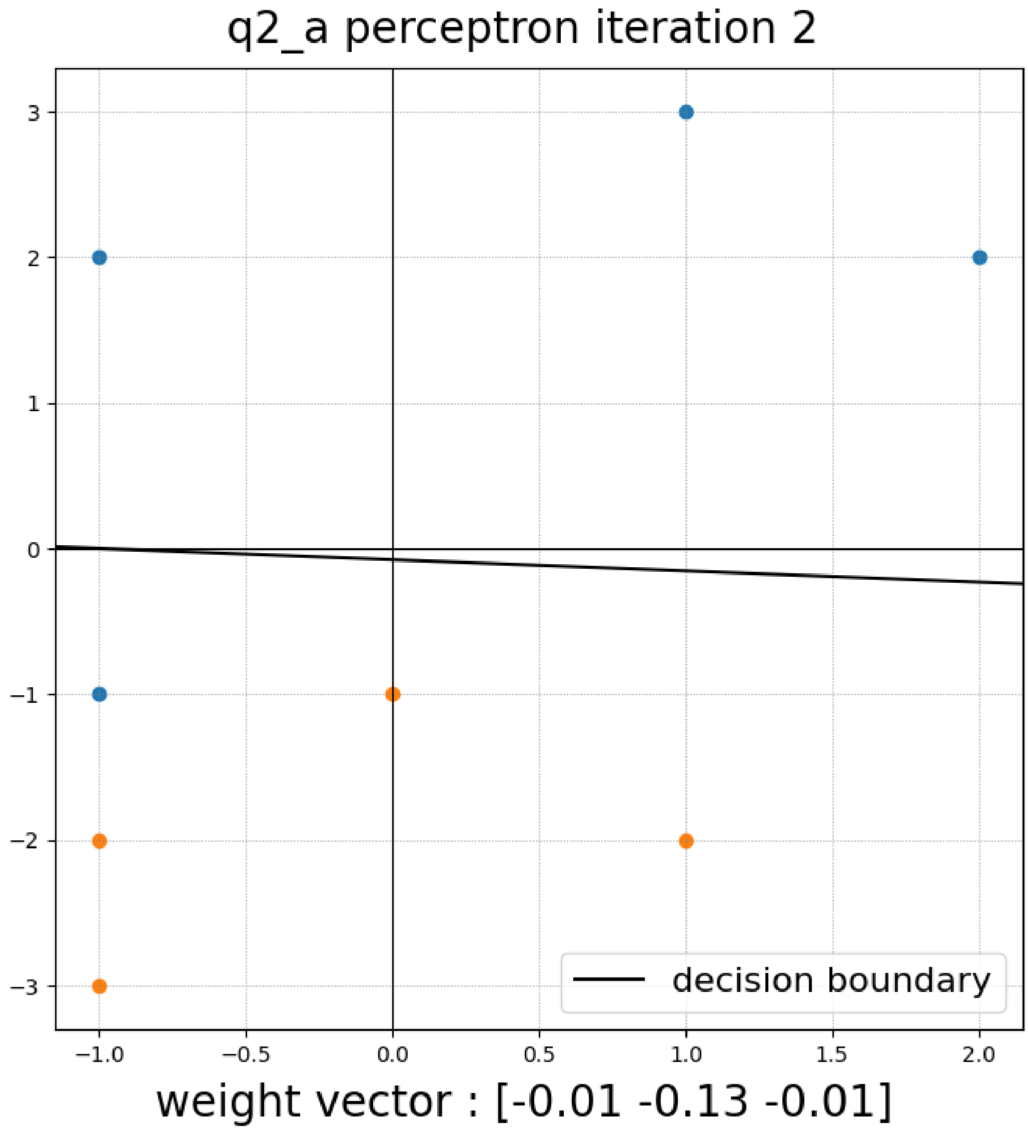
Q2 Demo (We use svm and perceptron classes defined as above)

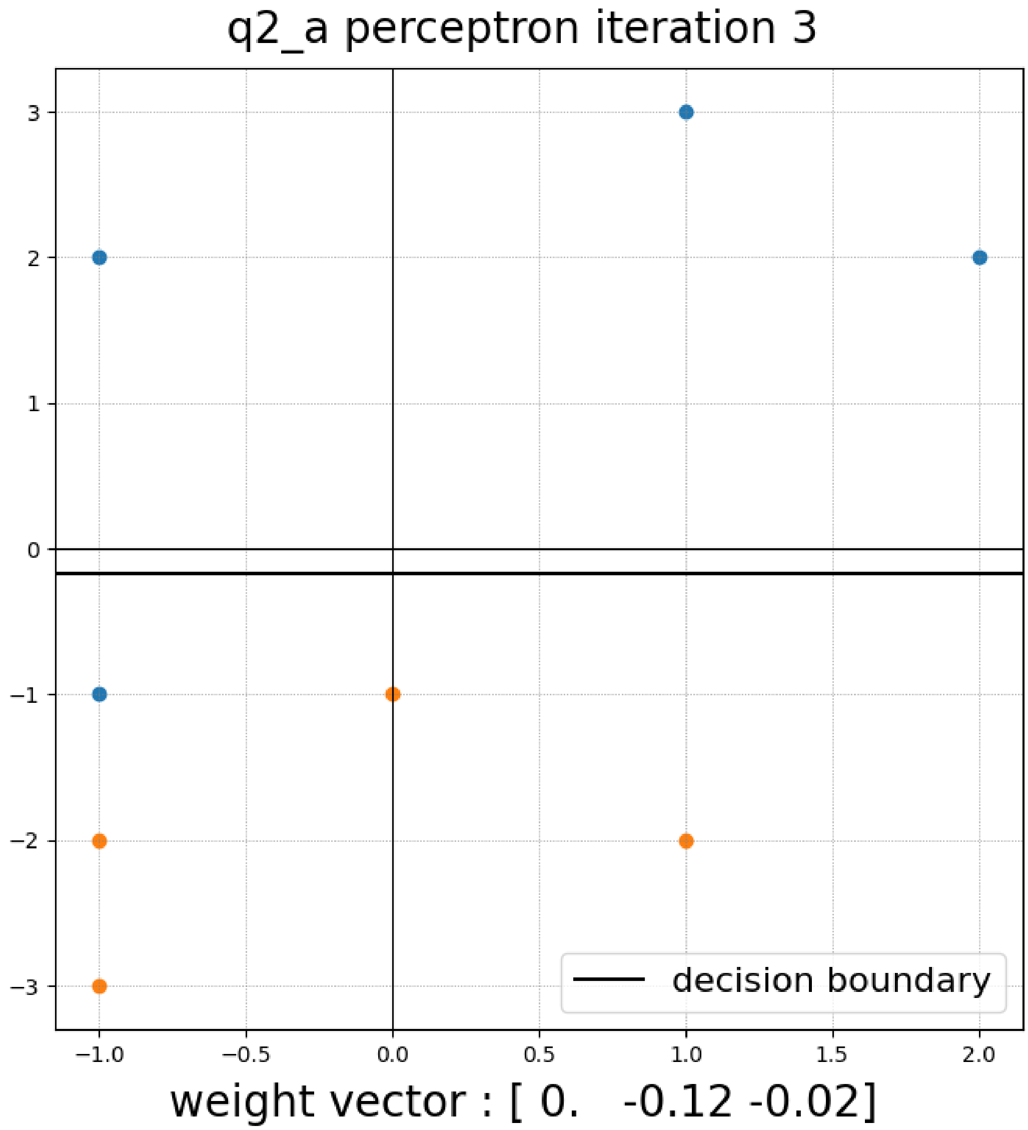
```
import svm,numpy as np,perceptron
```

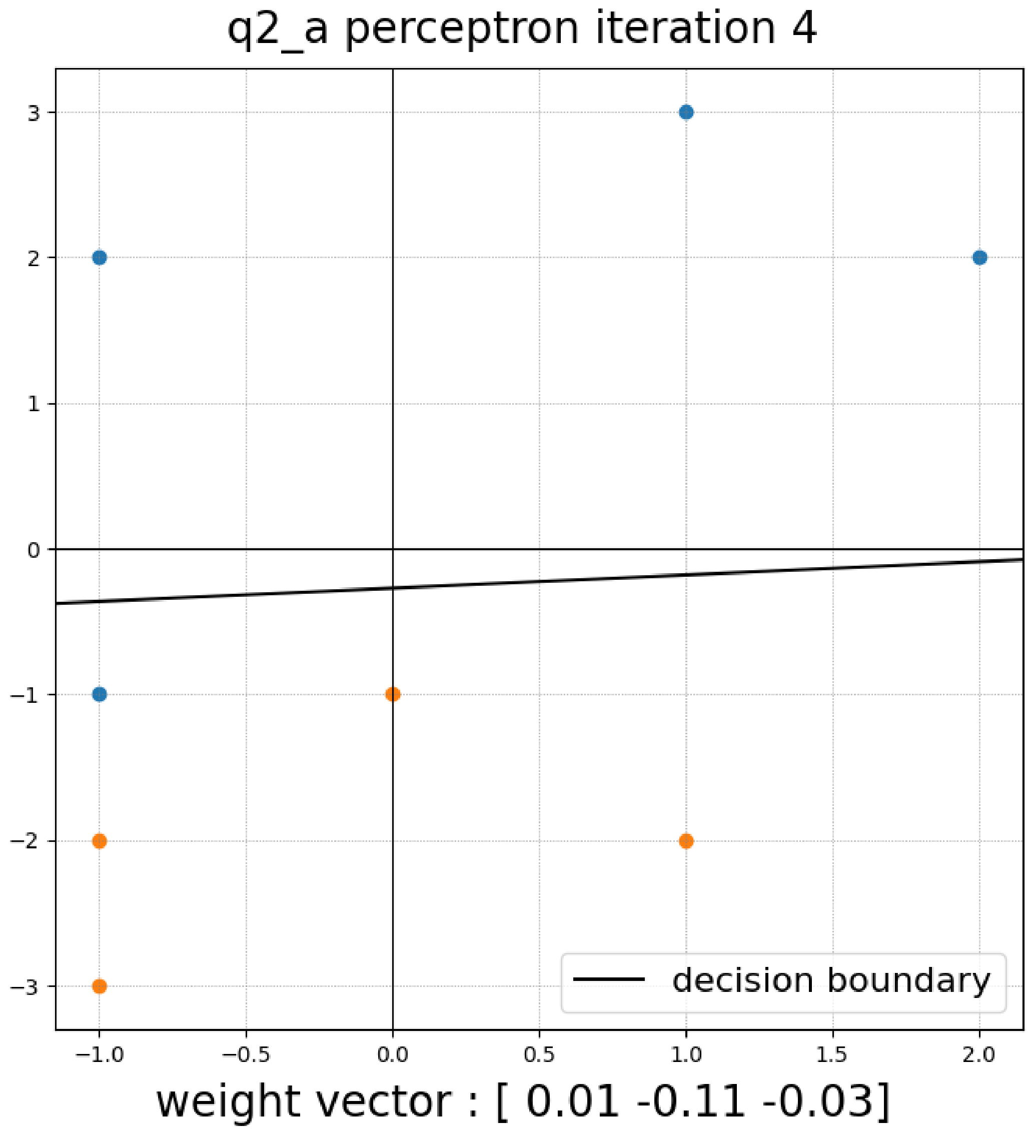
```python
x1=[2,-1,-1,0,1,-1,1,-1]
x2=[2,-3,2,-1,3,-2,-2,-1]

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")
```

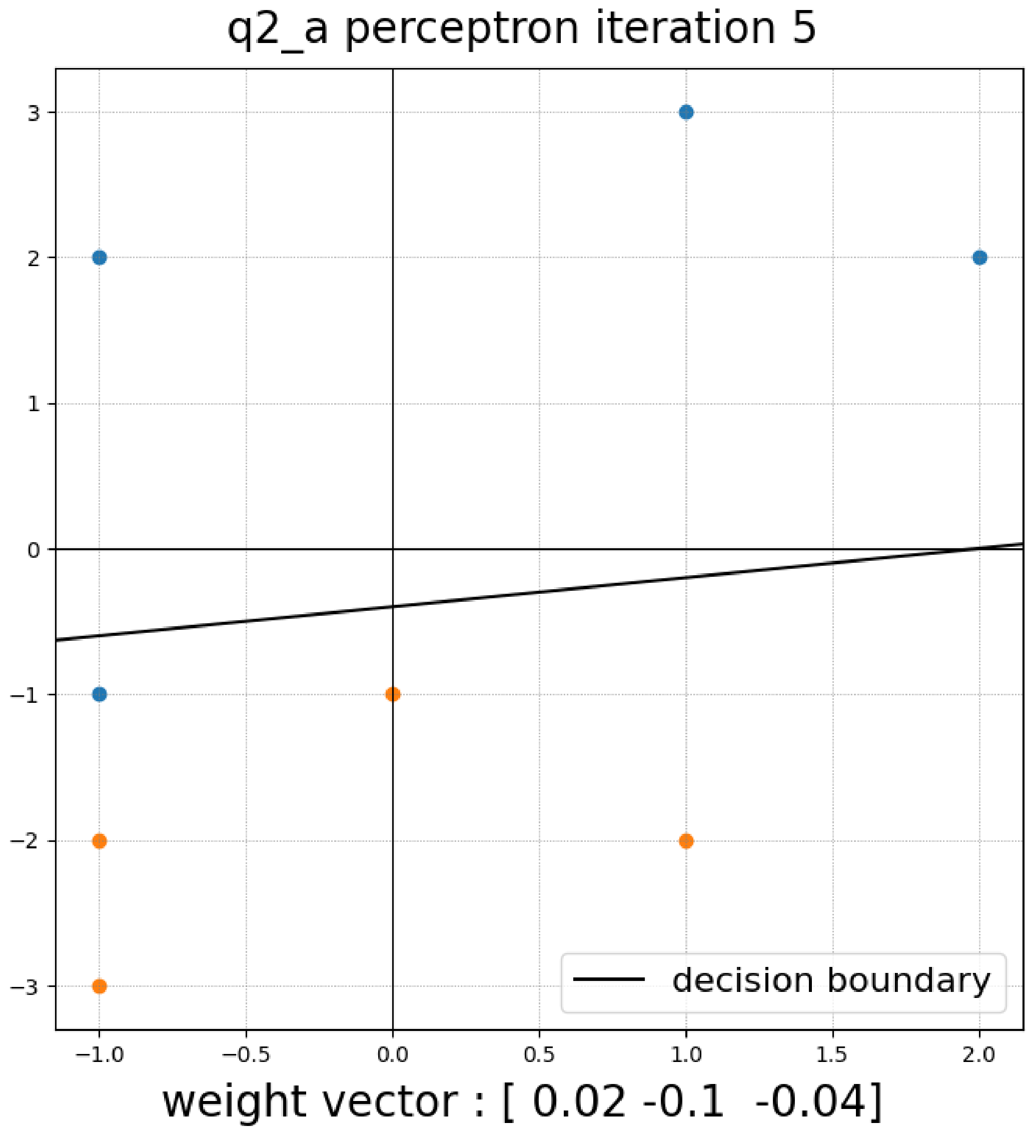

q2_a perceptron before training
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [0 0 0]

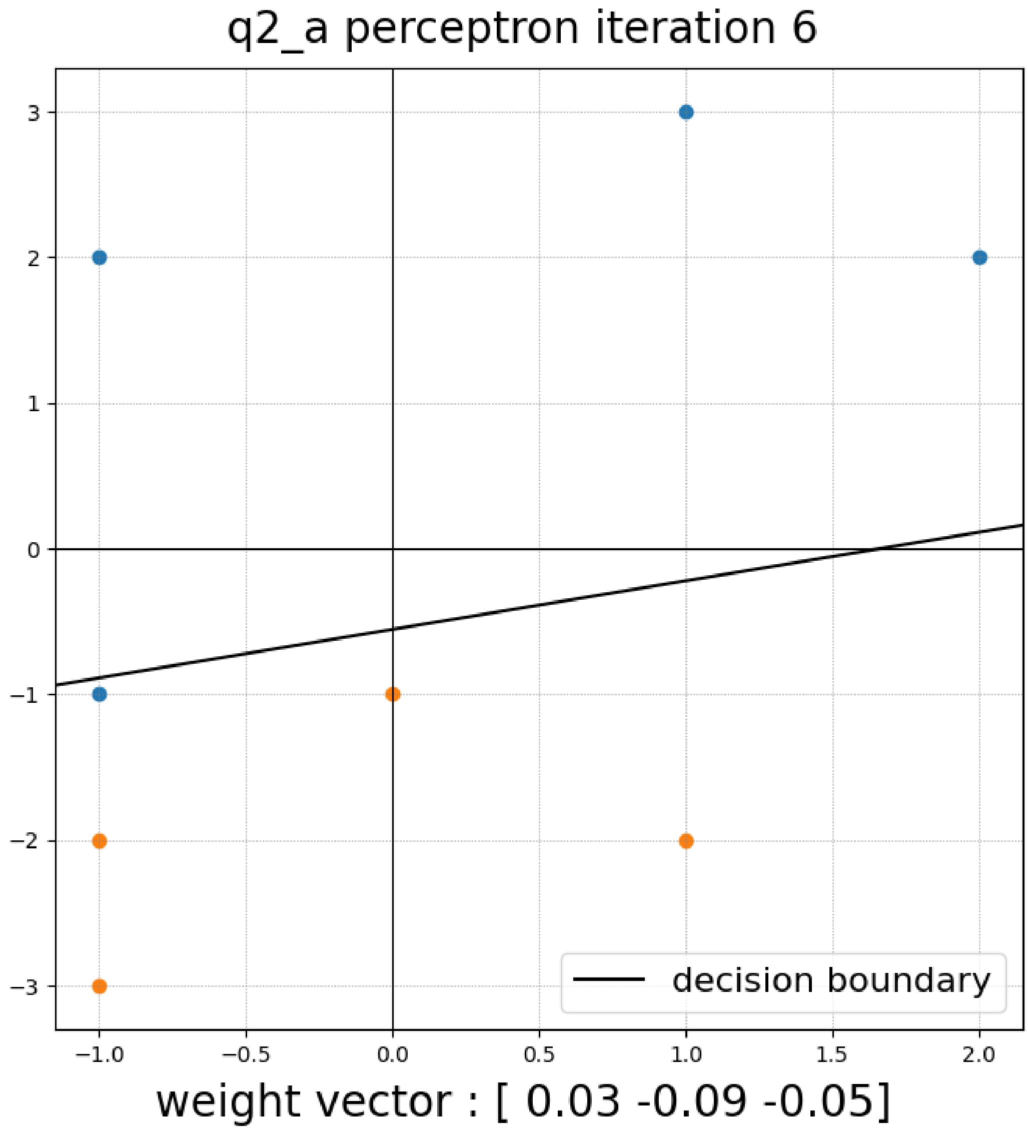

q2_a perceptron iteration 1
decision boundary
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [-0.02 -0.14 0.]

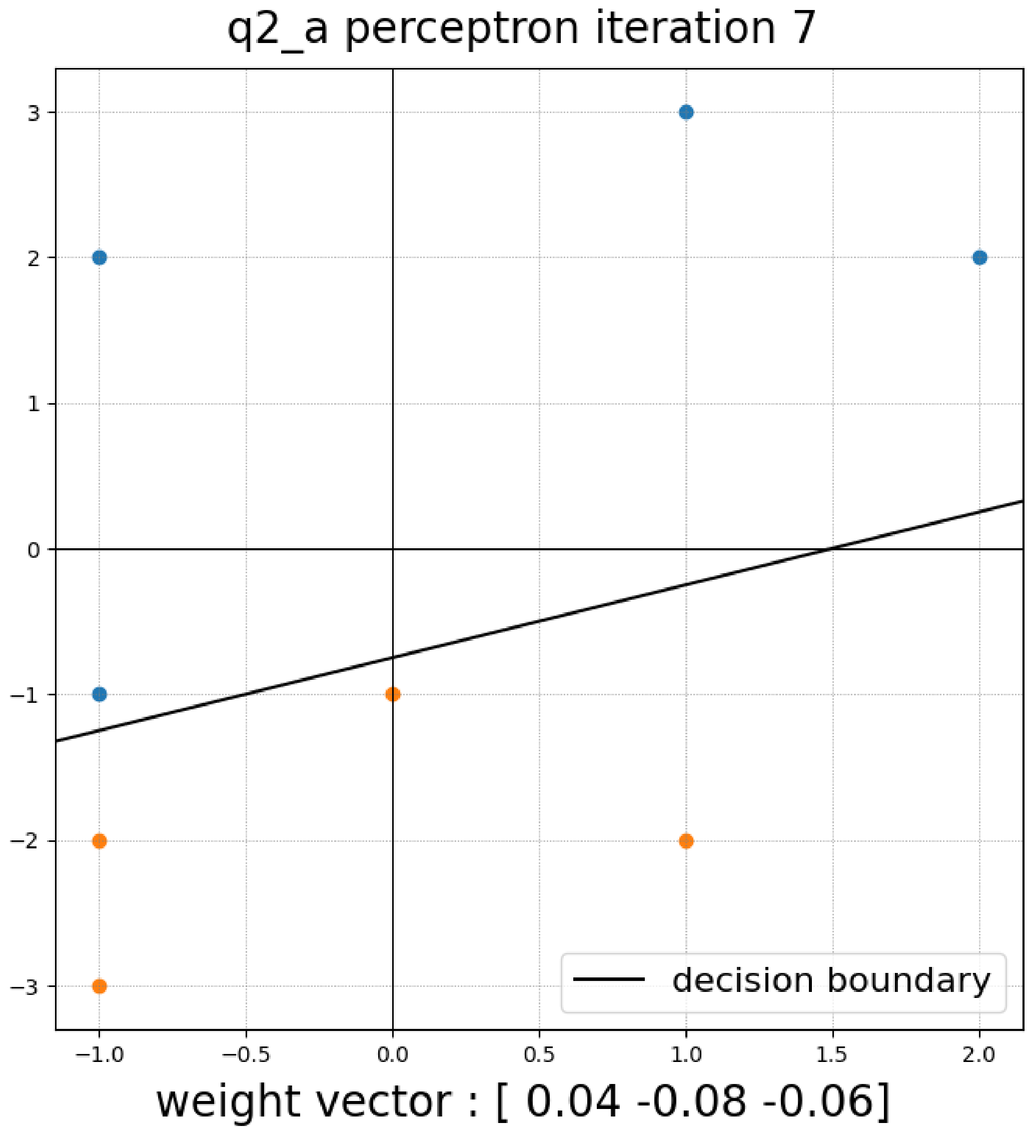

q2_a perceptron iteration 2
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [-0.01 -0.13 -0.01]

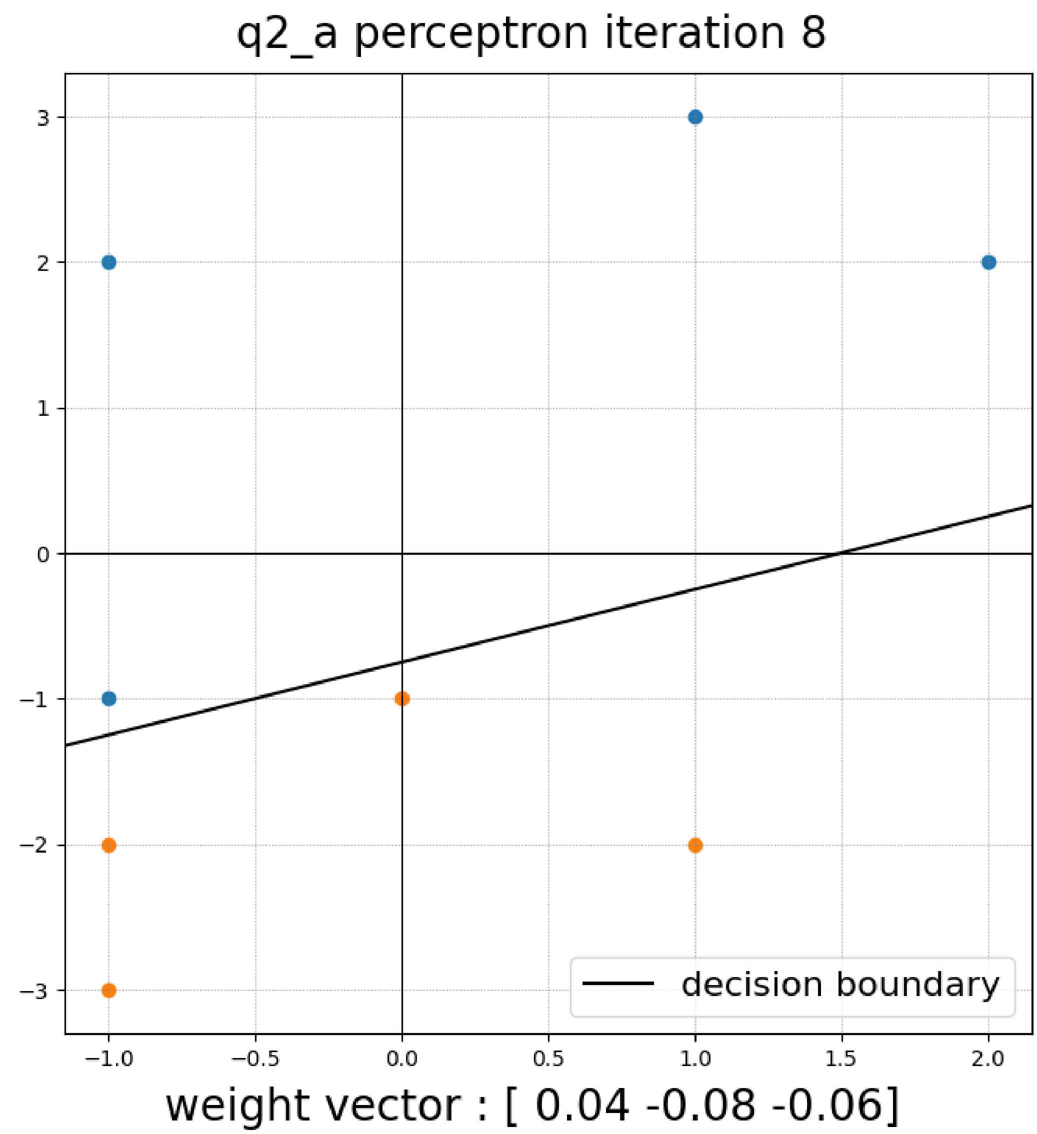

q2_a perceptron iteration 3
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0.   -0.12 -0.02]


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]

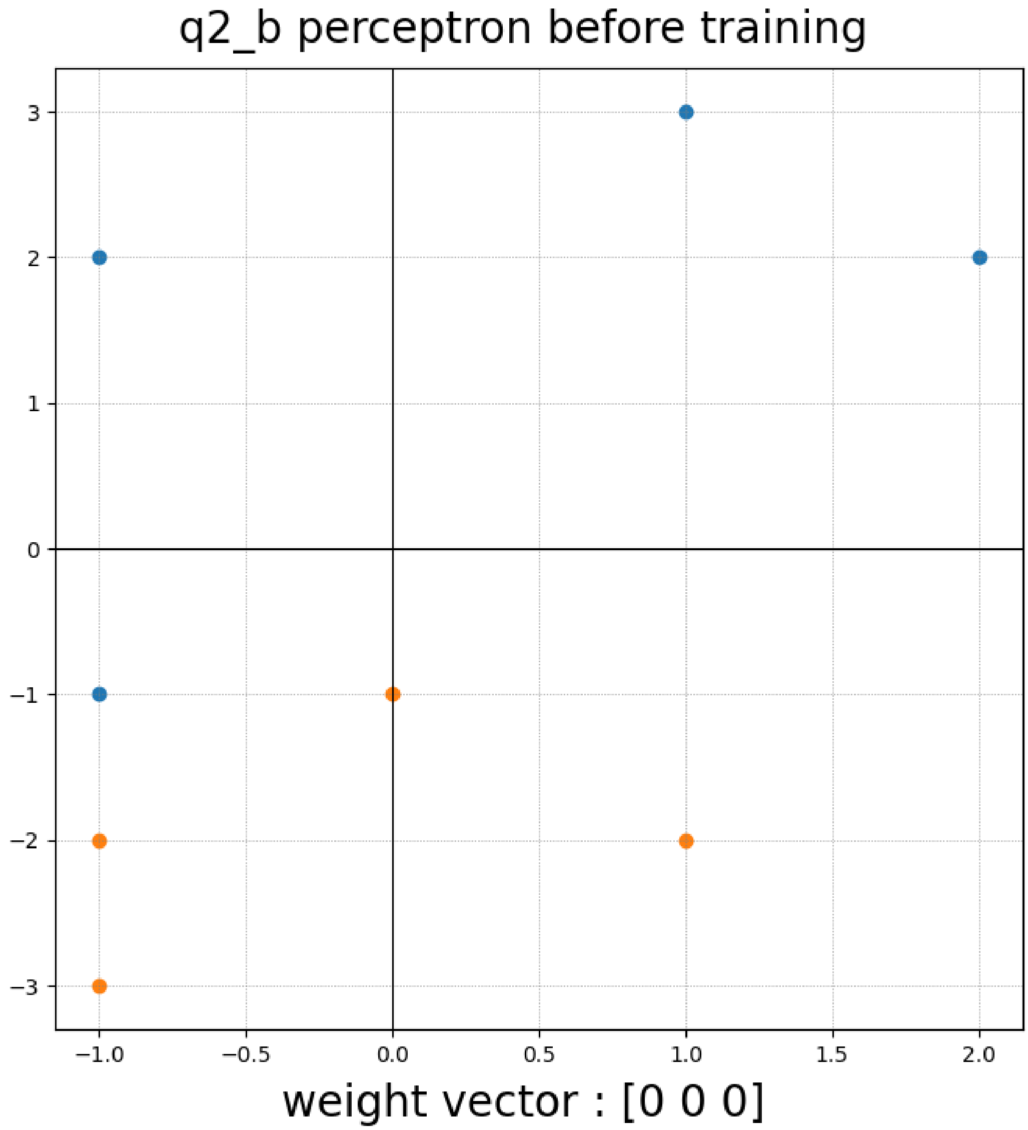

q2_a perceptron iteration 5
decision boundary
weight vector : [0.02 -0.1 -0.04]

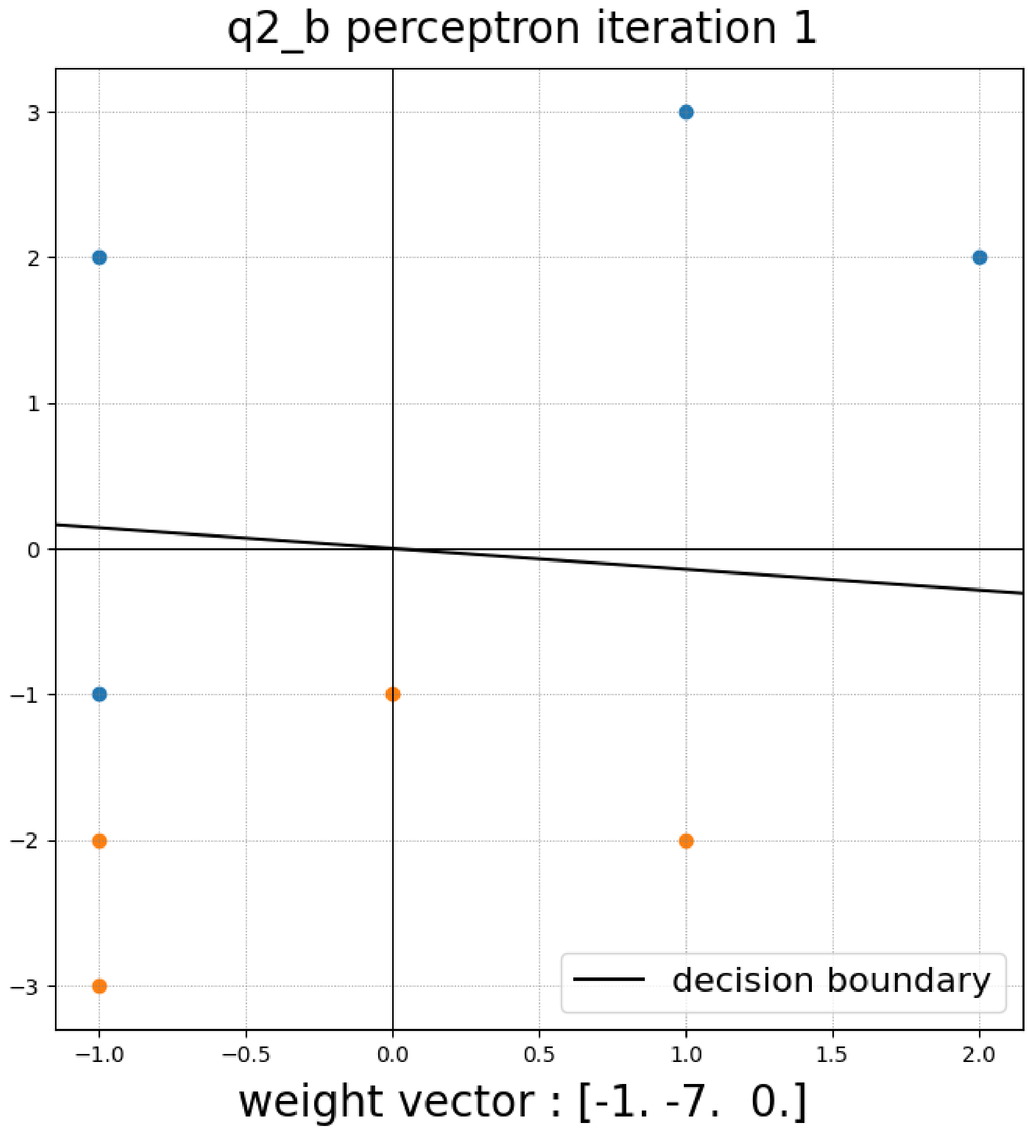

q2_a perceptron iteration 6
decision boundary
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [0.03 -0.09 -0.05]

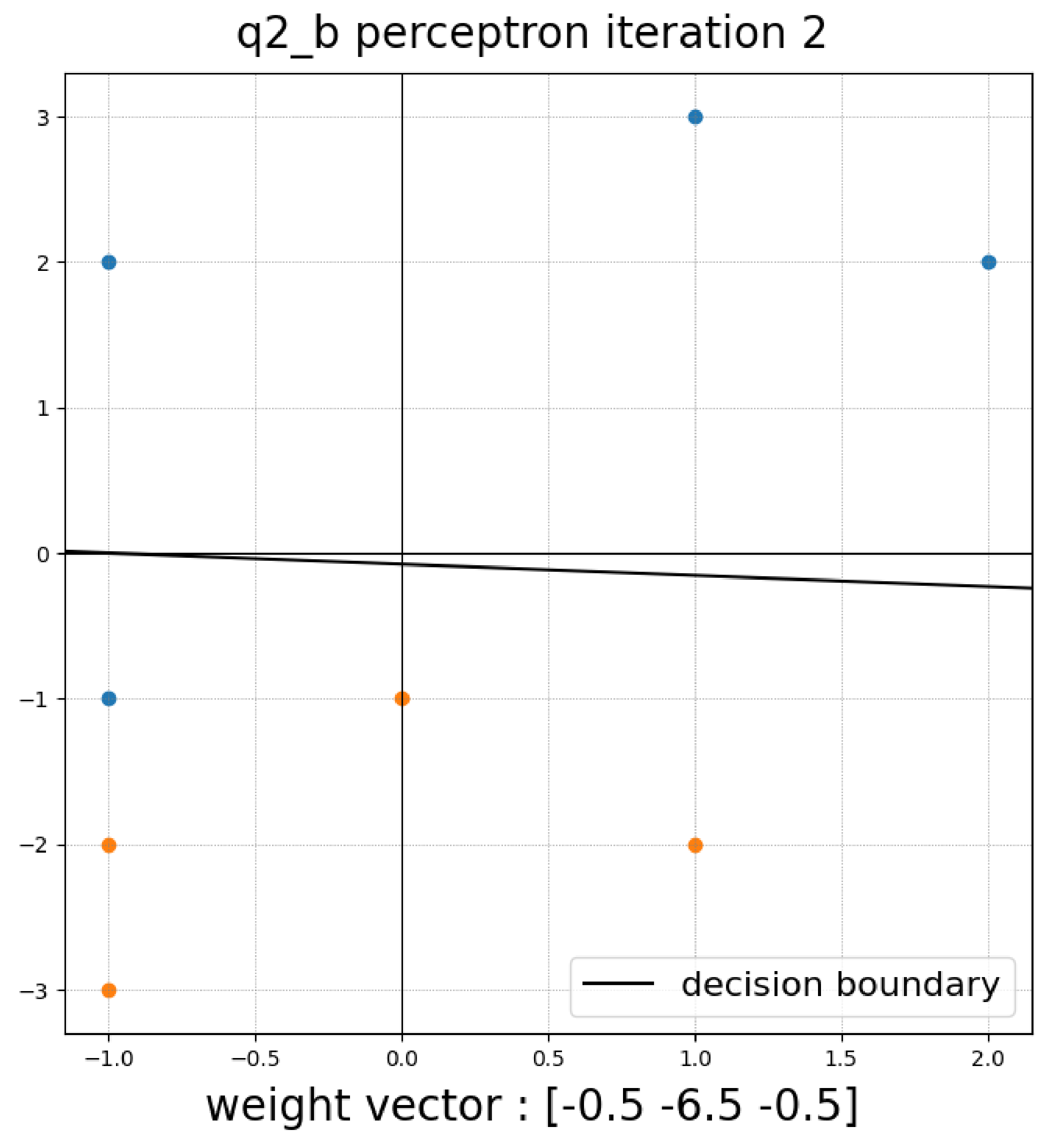

q2_a perceptron iteration 7
decision boundary
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [0.04 -0.08 -0.06]

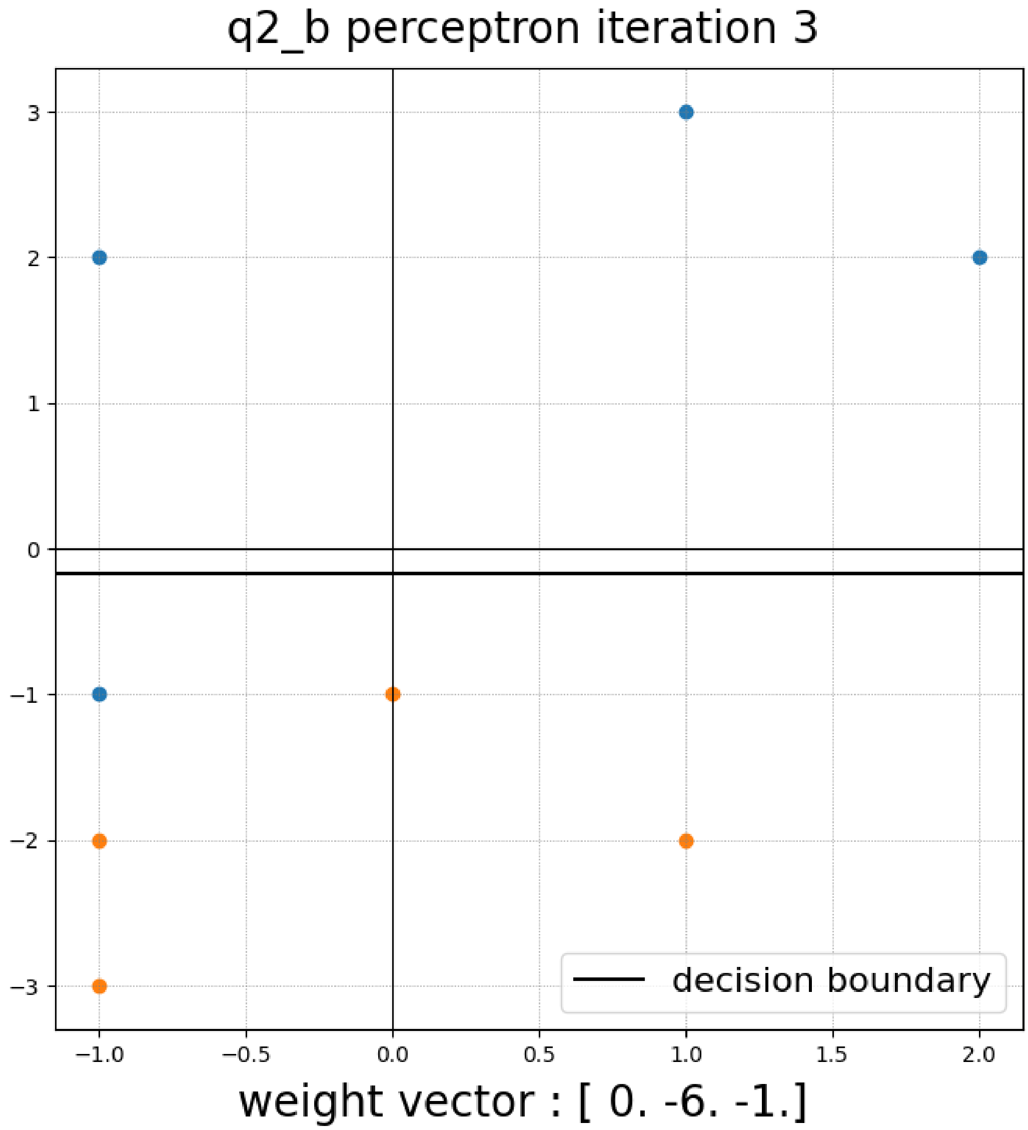


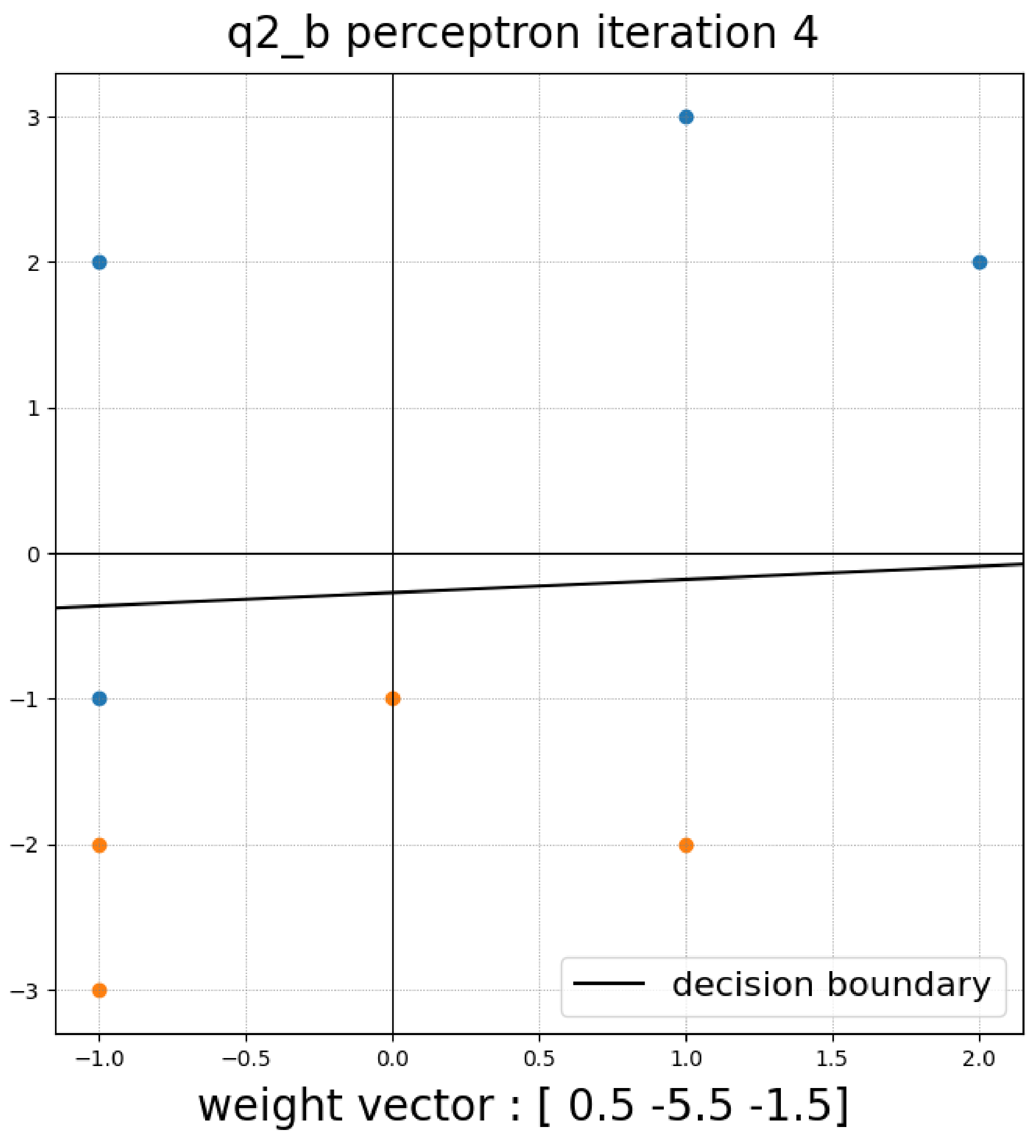
8 iterations for learning rate 0.01

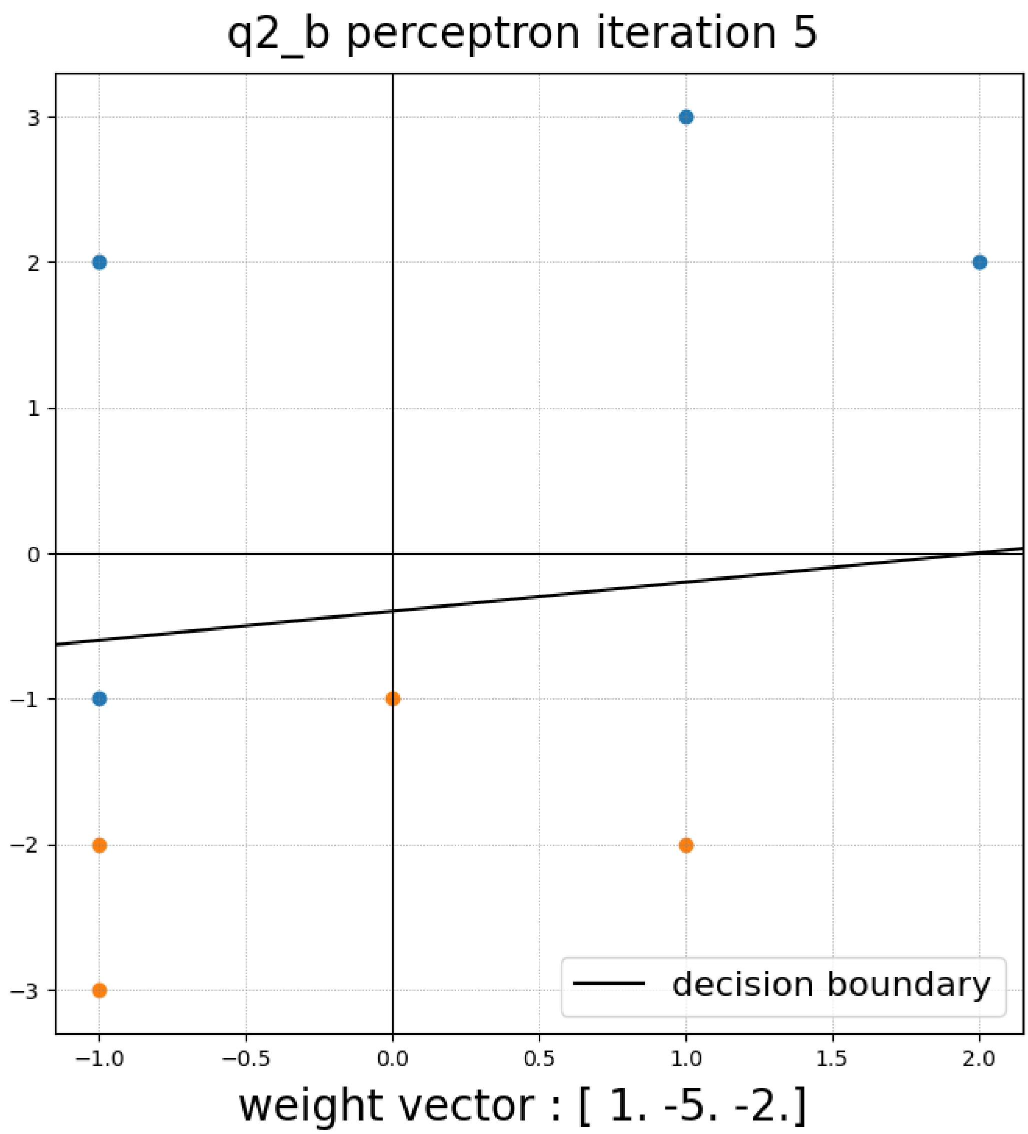

q2_b perceptron before training
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
weight vector : [0 0 0]

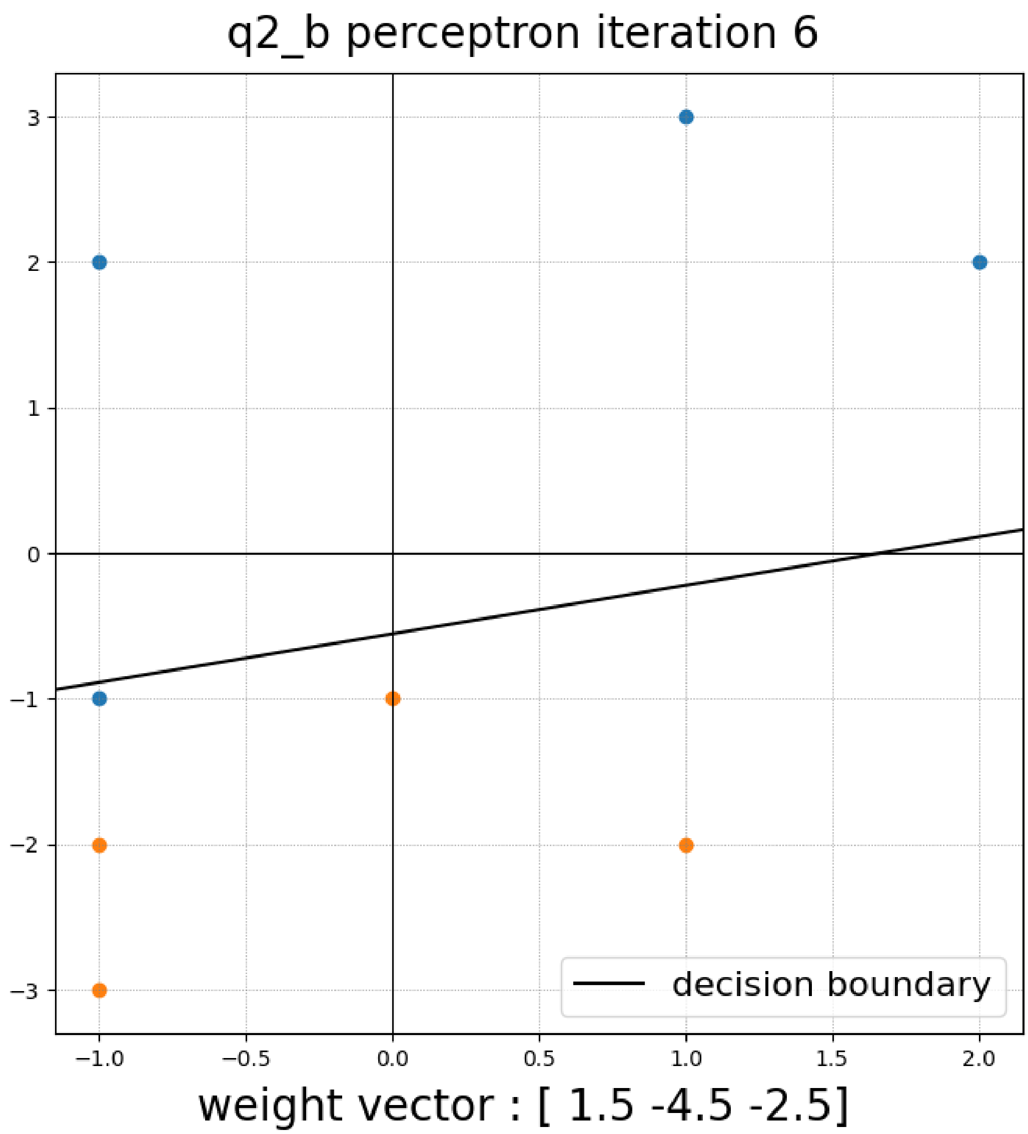

q2_b perceptron iteration 1
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [-1. -7. 0.]

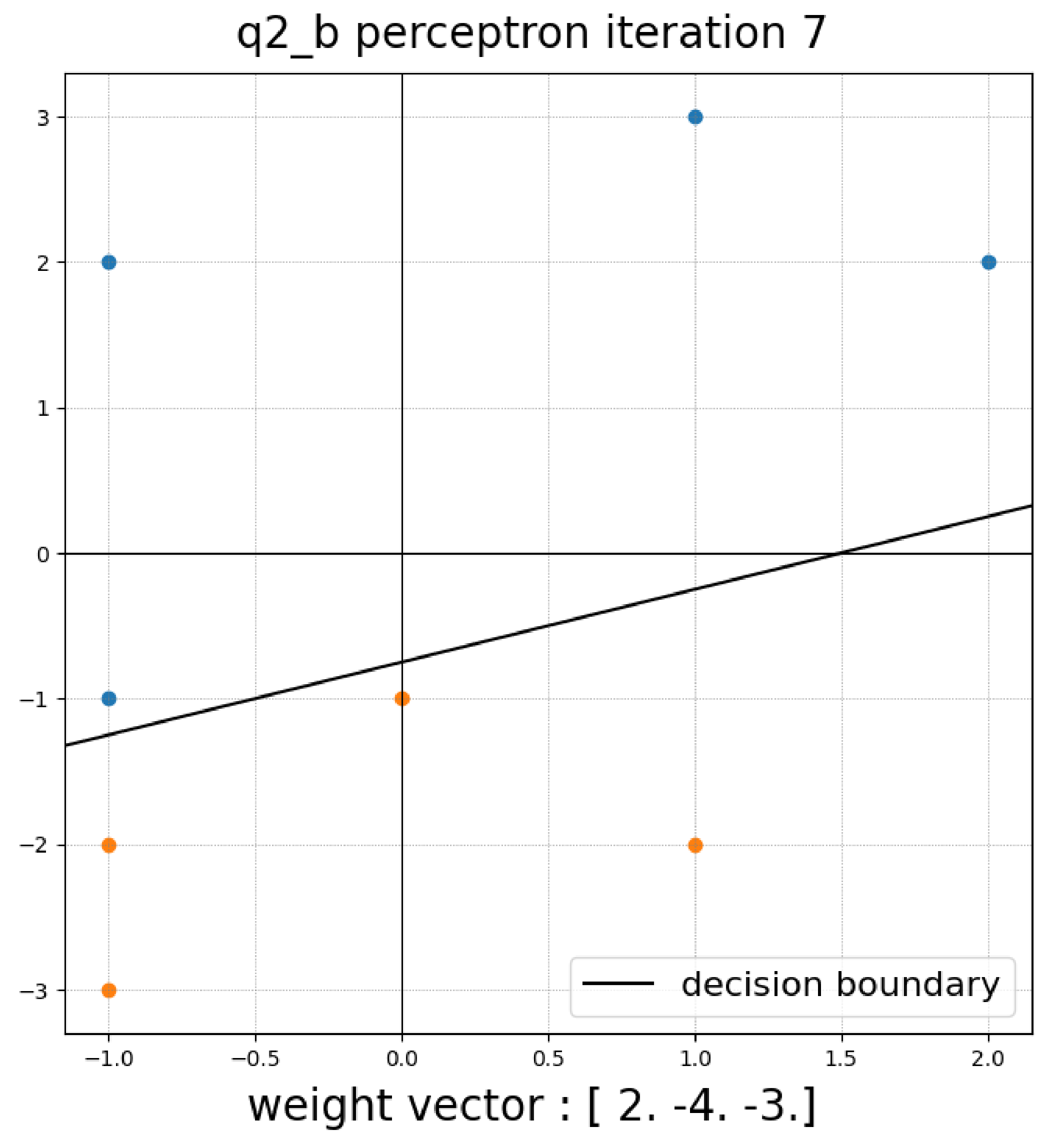

q2_b perceptron iteration 2
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [-0.5 -6.5 -0.5]

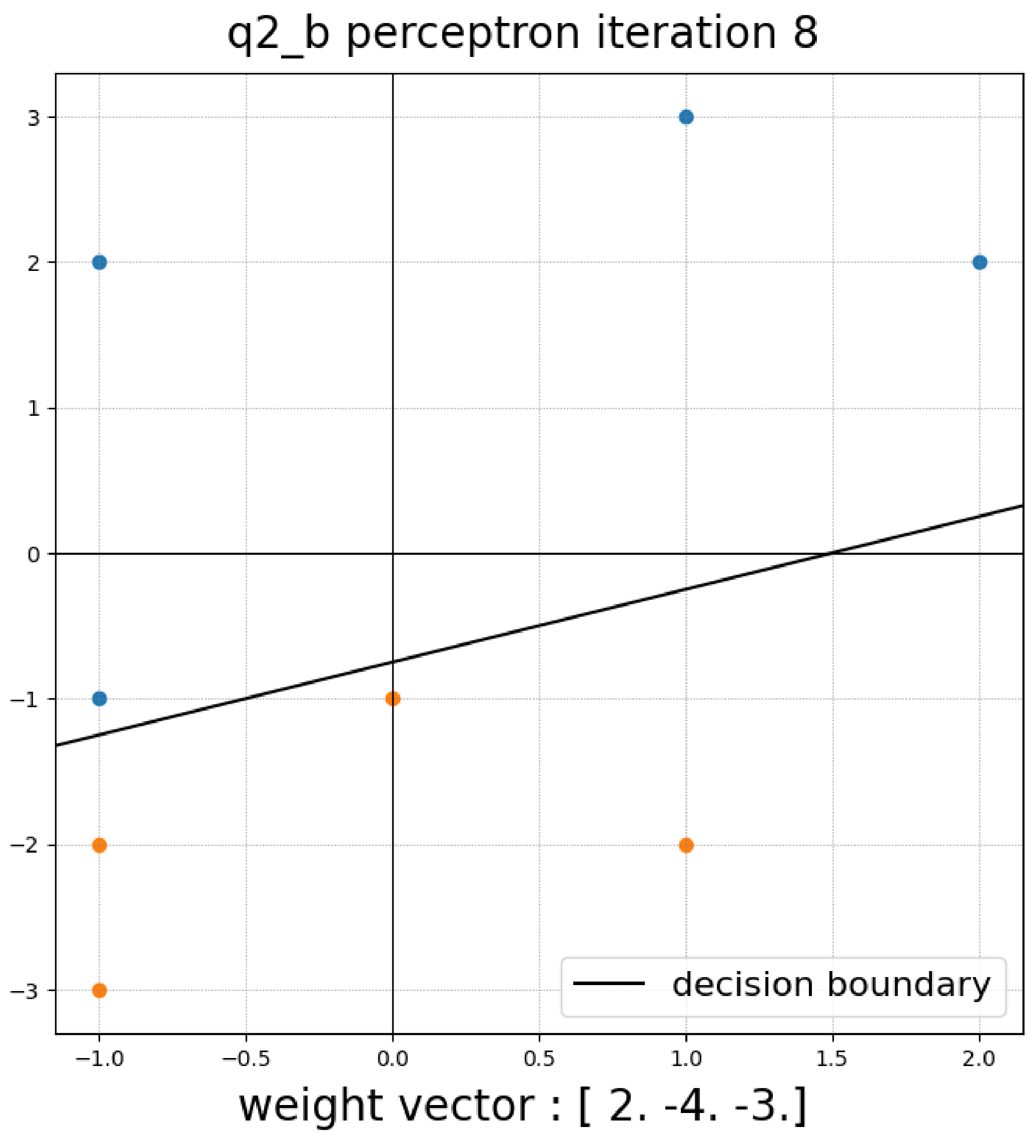

q2_b perceptron iteration 3
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [0. -6. -1.]


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
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. -2.]

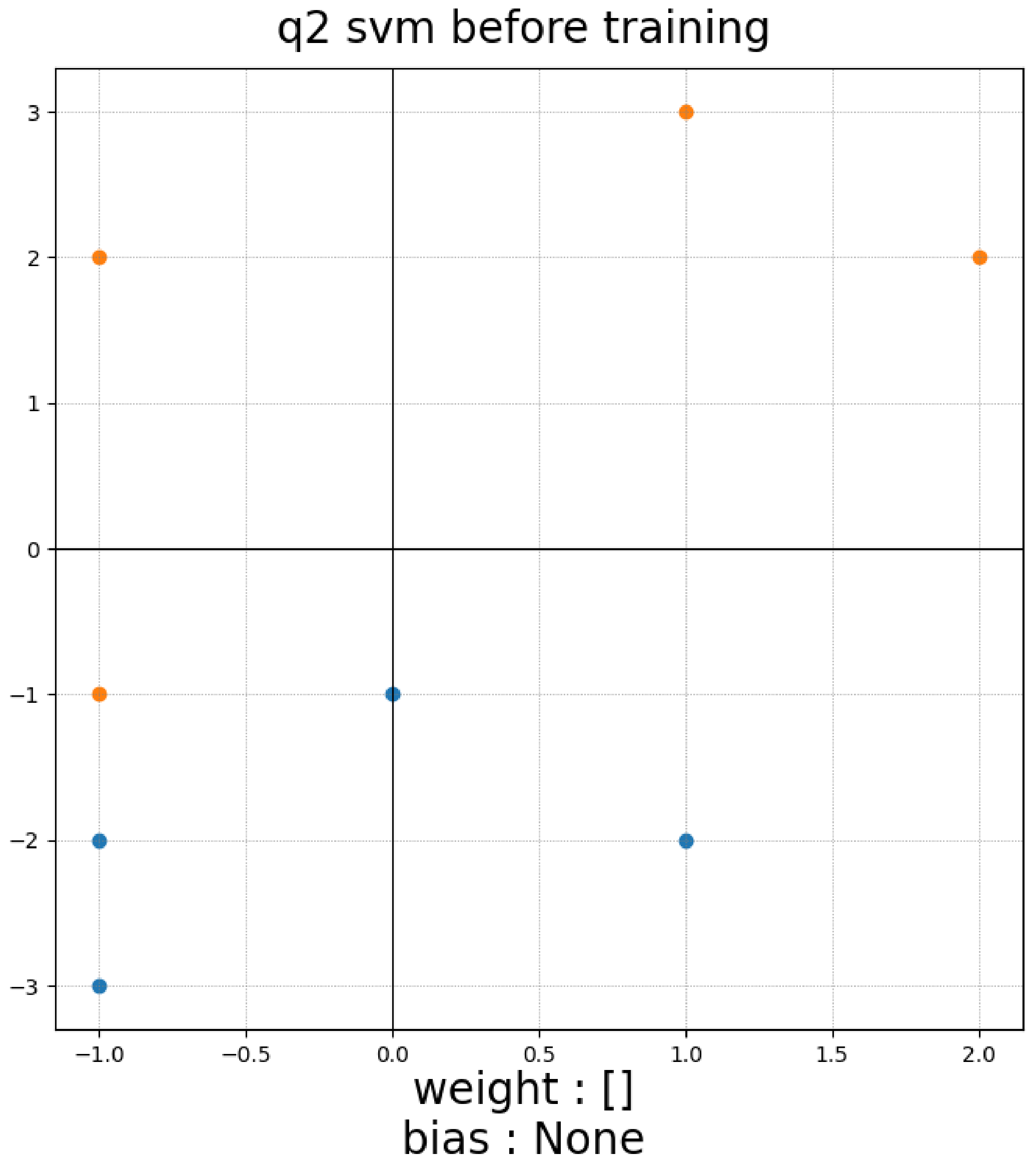

q2_b perceptron iteration 6
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
decision boundary
weight vector : [1.5 -4.5 -2.5]

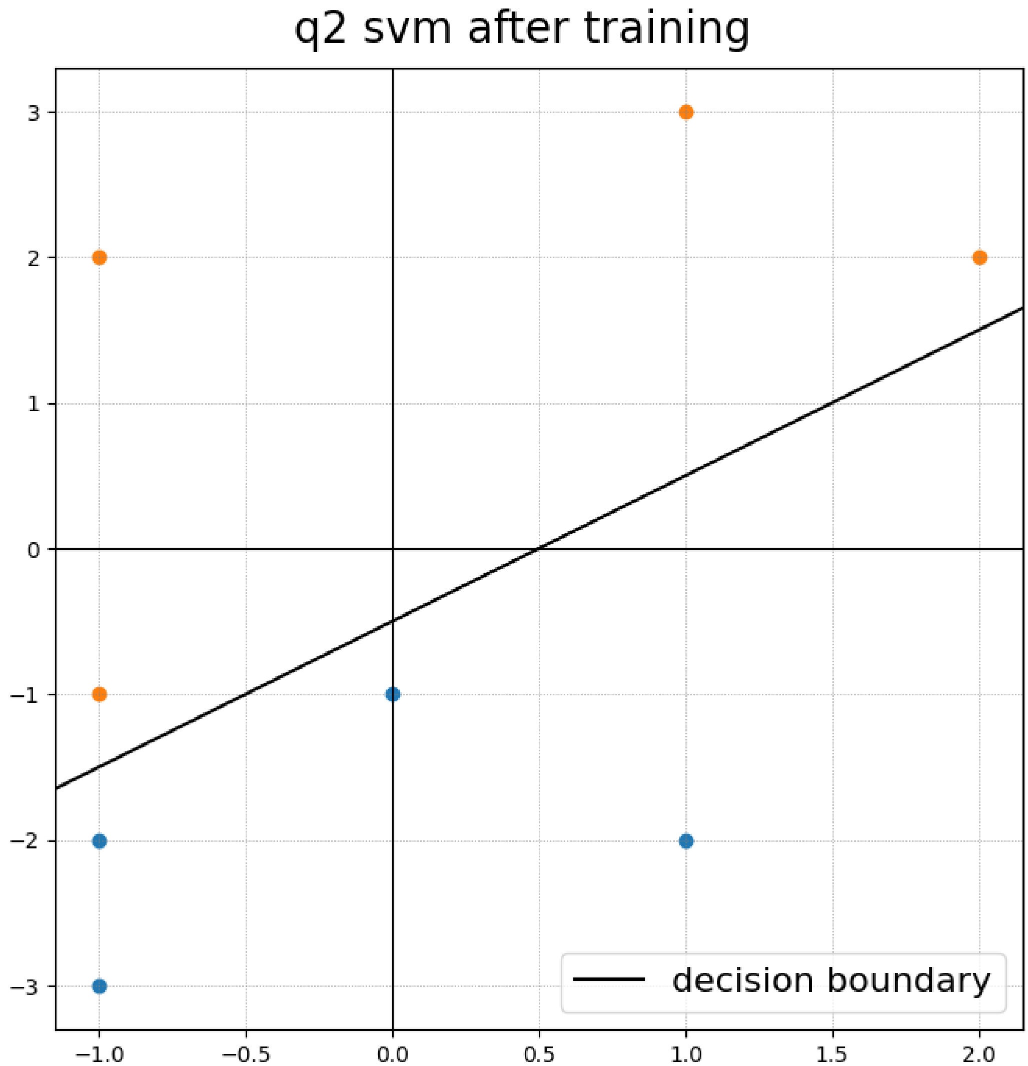

q2_b perceptron iteration 7
decision boundary
weight vector : [2. -4. -3.]



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

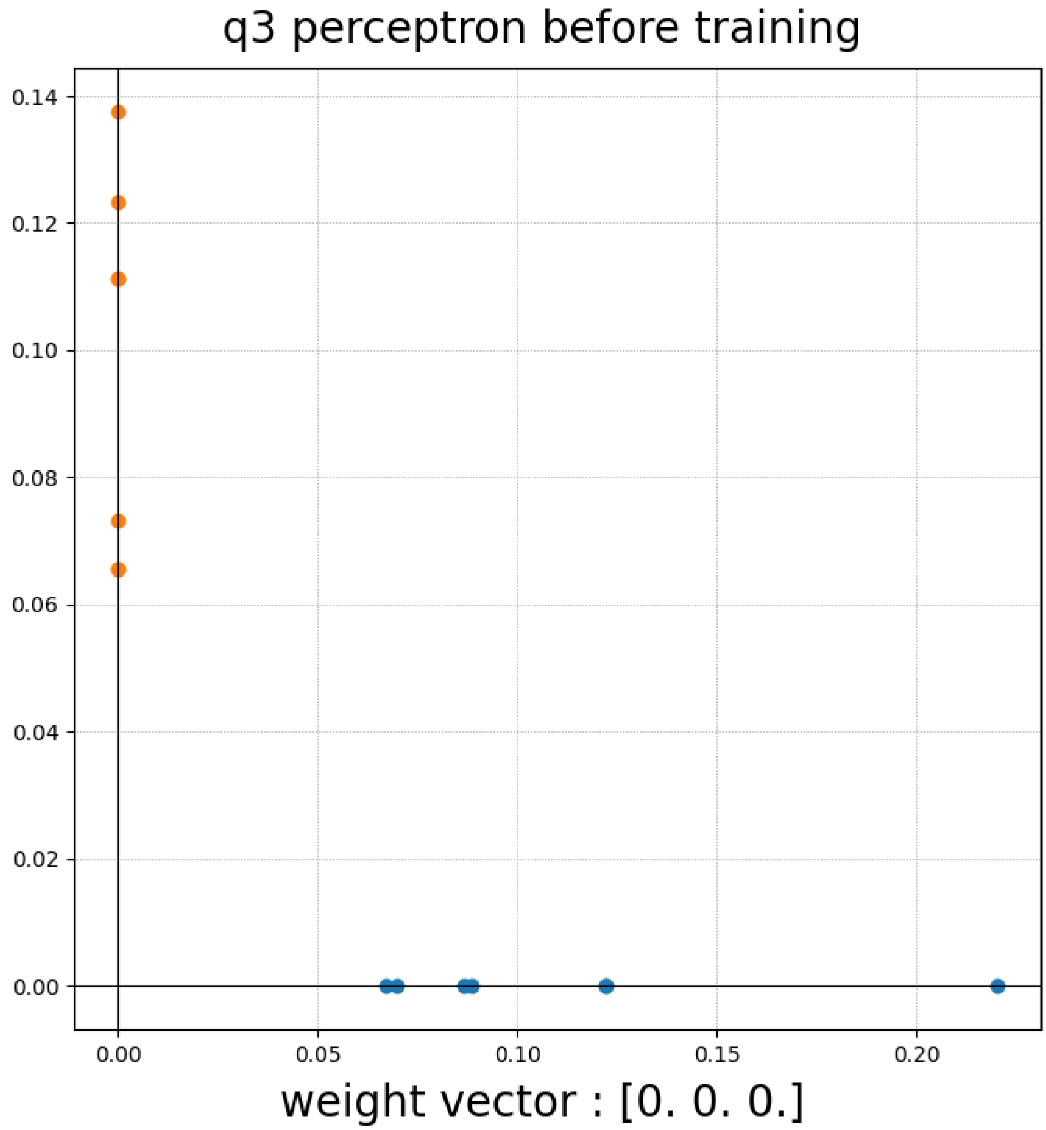


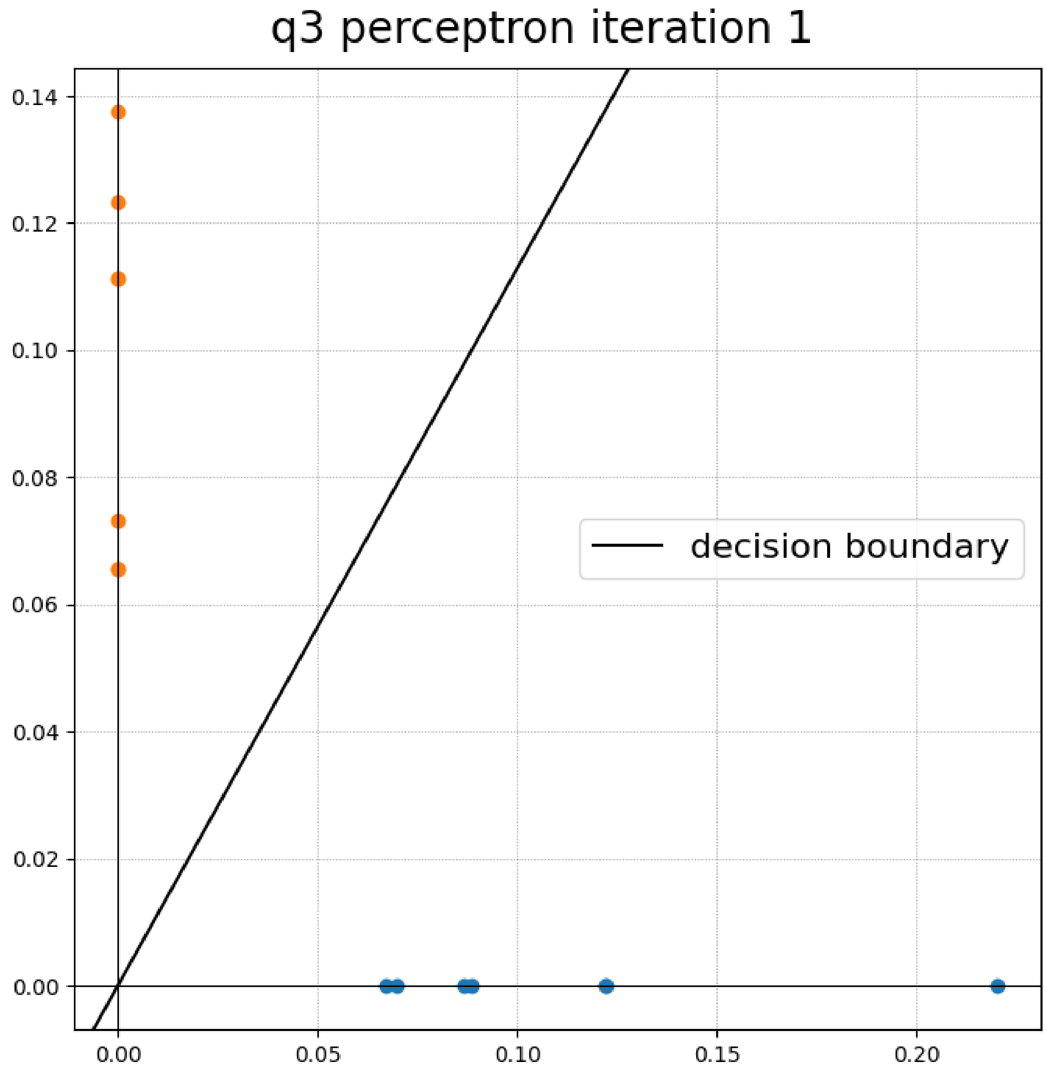
weight : [-2.00000023 2.00000031]
bias : 1.0000000792363046

Q3

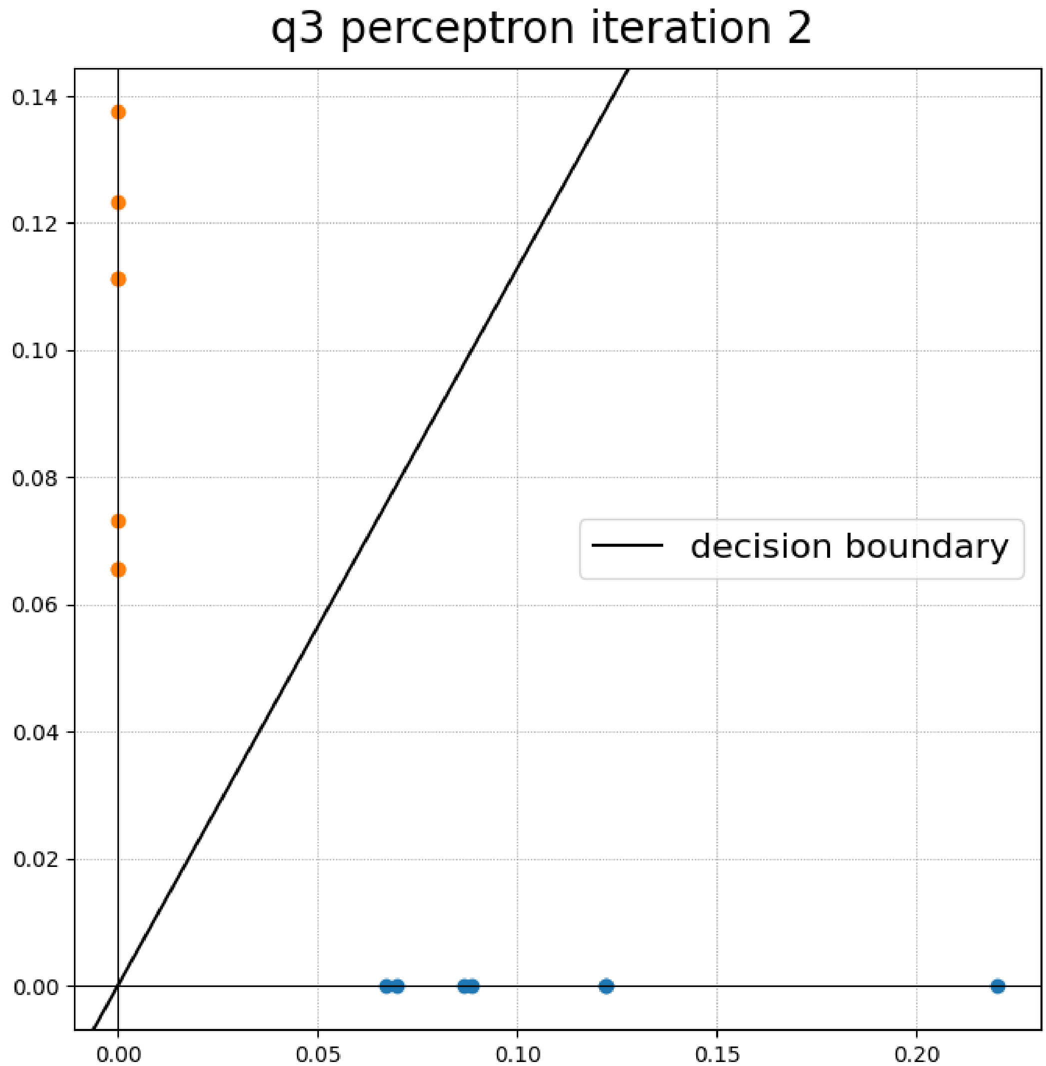
```
import numpy as np, cv2, svm, perceptron

# 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=[]
```

```python
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")
```

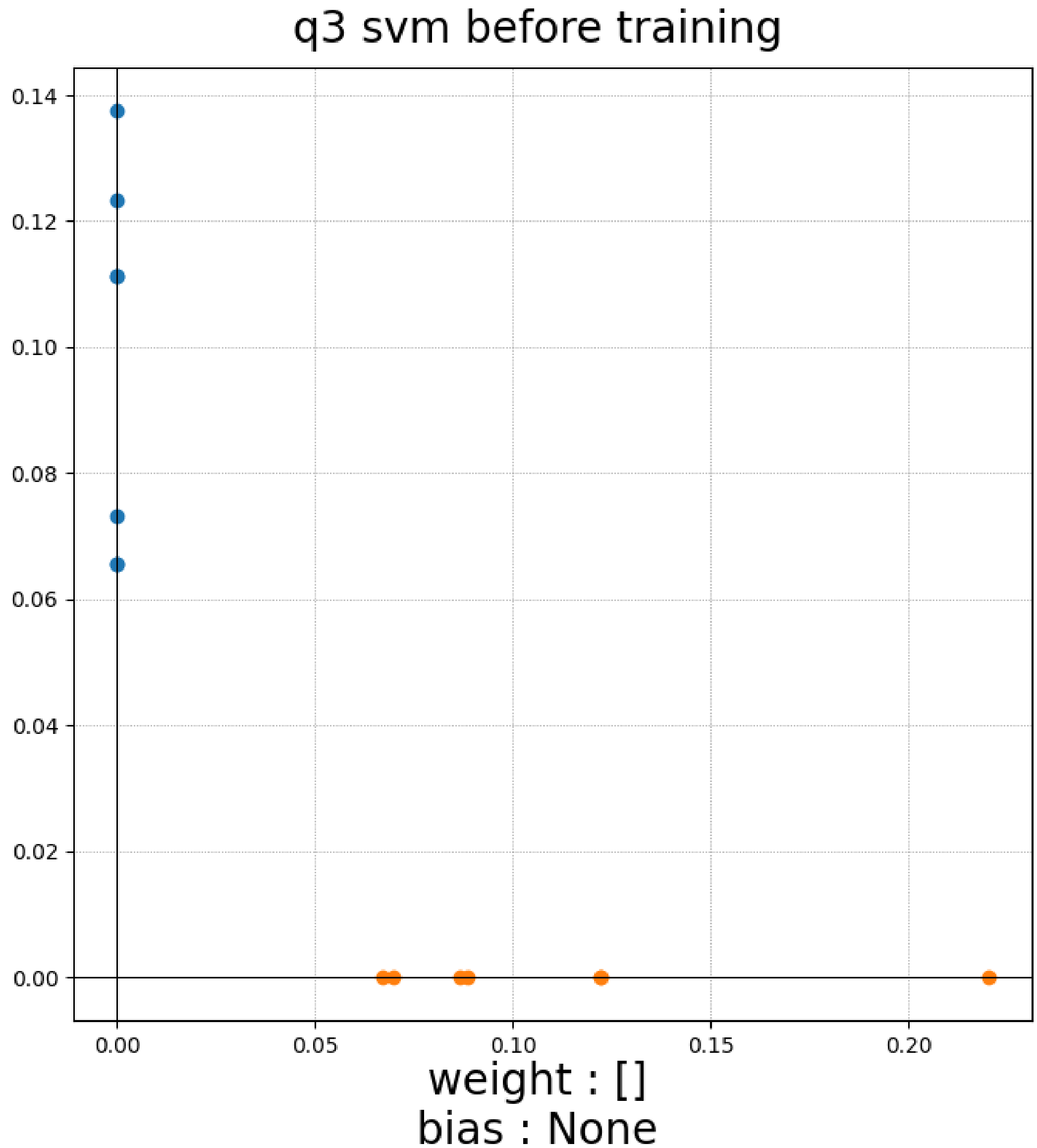

q3 perceptron before training
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0.00
0.05
0.10
0.15
0.20
weight vector : [0. 0. 0.]

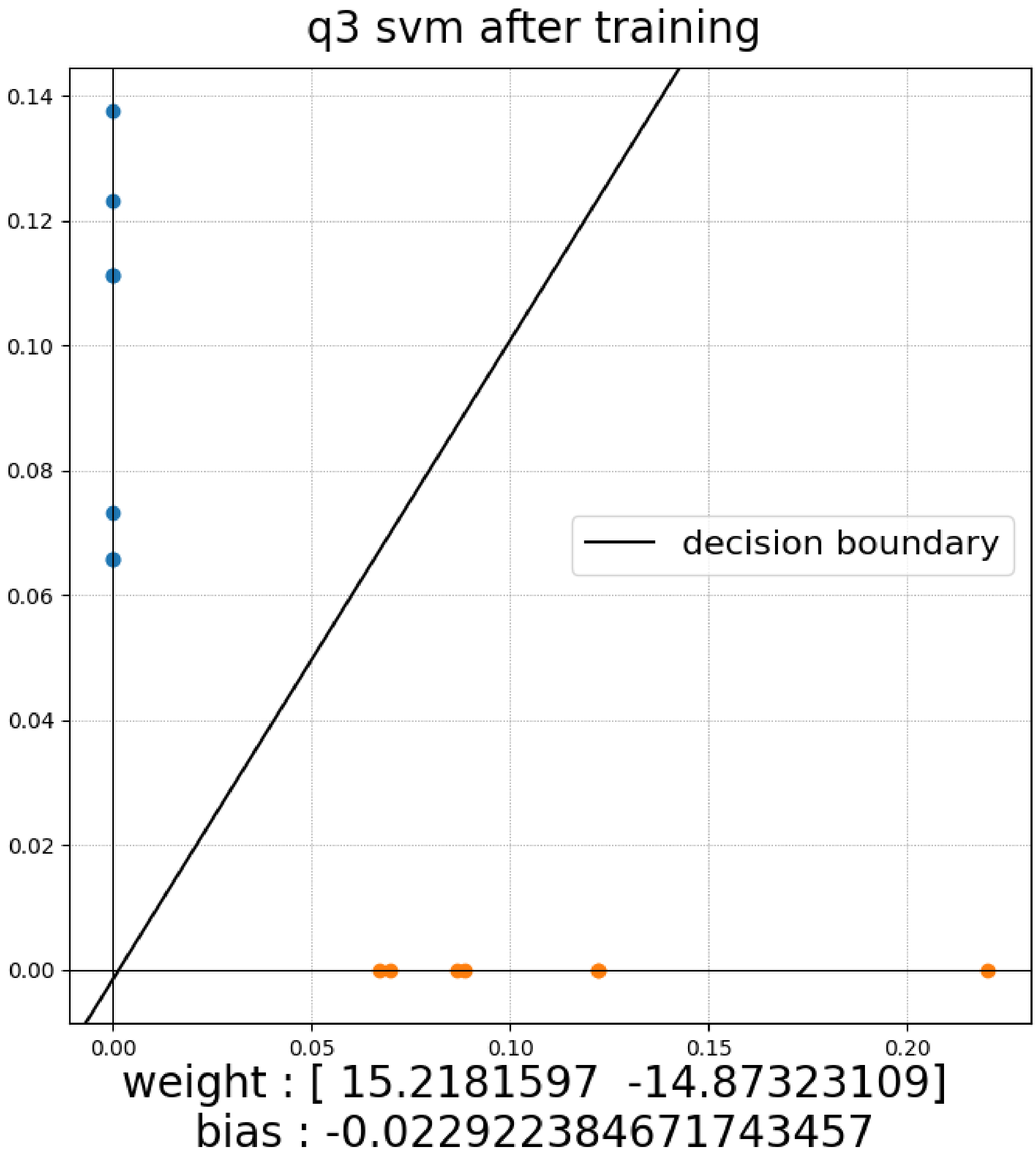


weight vector : [-0.00776683 0.00688116 0.   ]



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

Question 4

In [49]:

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

In [50]:

```
from cvxopt import matrix, solvers             #For Quadratic Programming
```

In [51]:

```
from sympy import *
```

Reading Iris Dataset

In [52]:

```
 iris_ds= pd.read_csv("Iris_dataset.csv",index_col=False)
print(iris_ds.head())
```

```
   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
```

Taking the two features only

In [53]:

```
iris_data = iris_ds.drop(['sepal.length','petal.width'], axis=1)
print(iris_data)
```

```
     sepal.width  petal.length    variety
0            3.5           1.4     Setosa
1            3.0           1.4     Setosa
2            3.2           1.3     Setosa
3            3.1           1.5     Setosa
4            3.6           1.4     Setosa
..           ...           ...        ...
145          3.0           5.2  Virginica
146          2.5           5.0  Virginica
147          3.0           5.2  Virginica
148          3.4           5.4  Virginica
149          3.0           5.1  Virginica

[150 rows x 3 columns]
```

Dividing the Data into 3 sets (Setosa vs Versicolor, Setosa vs Virginica, Versicolor vs Virginica
Since Single Perceptron and SVM can handle two classes

In [54]:

```
iris_data_12 = iris_data[iris_data['variety']!='Virginica']
print(iris_data_12)
```

```
    sepal.width  petal.length     variety
0           3.5           1.4      Setosa
1           3.0           1.4      Setosa
2           3.2           1.3      Setosa
3           3.1           1.5      Setosa
4           3.6           1.4      Setosa
..          ...           ...         ...
95          3.0           4.2  Versicolor
96          2.9           4.2  Versicolor
97          2.9           4.3  Versicolor
98          2.5           3.0  Versicolor
99          2.8           4.1  Versicolor

[100 rows x 3 columns]
```

In [55]:

```
var=[]
for i in range(len(iris_data_12)):
    if iris_data_12['variety'].iloc[i]=='Setosa':
        var.append(1)
    else:
        var.append(-1)
iris_data_12['var'] = var
iris_data_12 = iris_data_12.drop(['variety'],axis=1)
print(iris_data_12)
```

```
    sepal.width  petal.length  var
0           3.5           1.4    1
1           3.0           1.4    1
2           3.2           1.3    1
3           3.1           1.5    1
4           3.6           1.4    1
..          ...           ...  ...
95          3.0           4.2   -1
96          2.9           4.2   -1
97          2.9           4.3   -1
98          2.5           3.0   -1
99          2.8           4.1   -1

[100 rows x 3 columns]
/home/shreja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:7: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

In [56]:

```
iris_data_13 = iris_data[iris_data['variety']!='Versicolor']
print(iris_data_13)
```

```
     sepal.width  petal.length    variety
0            3.5           1.4     Setosa
1            3.0           1.4     Setosa
2            3.2           1.3     Setosa
3            3.1           1.5     Setosa
4            3.6           1.4     Setosa
..           ...           ...        ...
145          3.0           5.2  Virginica
146          2.5           5.0  Virginica
147          3.0           5.2  Virginica
148          3.4           5.4  Virginica
149          3.0           5.1  Virginica

[100 rows x 3 columns]
```

In [57]:

```
var=[]
for i in range(len(iris_data_13)):
    if iris_data_13['variety'].iloc[i]=='Setosa':
        var.append(1)
    else:
        var.append(-1)
iris_data_13['var'] = var
iris_data_13 = iris_data_13.drop(['variety'],axis=1)
print(iris_data_13)
```

```
     sepal.width  petal.length  var
0            3.5           1.4    1
1            3.0           1.4    1
2            3.2           1.3    1
3            3.1           1.5    1
4            3.6           1.4    1
..           ...           ...  ...
145          3.0           5.2   -1
146          2.5           5.0   -1
147          3.0           5.2   -1
148          3.4           5.4   -1
149          3.0           5.1   -1

[100 rows x 3 columns]
/home/shreja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:7: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

In [58]:

```
iris_data_23 = iris_data[iris_data['variety']!='Setosa']
print(iris_data_23)
```

```
     sepal.width  petal.length     variety
50           3.2           4.7  Versicolor
51           3.2           4.5  Versicolor
52           3.1           4.9  Versicolor
53           2.3           4.0  Versicolor
54           2.8           4.6  Versicolor
..           ...           ...         ...
145          3.0           5.2   Virginica
146          2.5           5.0   Virginica
147          3.0           5.2   Virginica
148          3.4           5.4   Virginica
149          3.0           5.1   Virginica

[100 rows x 3 columns]
```

In [59]:

```
var=[]
for i in range(len(iris_data_23)):
    if iris_data_23['variety'].iloc[i]=='Versicolor':
        var.append(1)
    else:
        var.append(-1)
iris_data_23['var'] = var
iris_data_23 = iris_data_23.drop(['variety'],axis=1)
print(iris_data_23)
```

```
     sepal.width  petal.length  var
50           3.2           4.7    1
51           3.2           4.5    1
52           3.1           4.9    1
53           2.3           4.0    1
54           2.8           4.6    1
..           ...           ...  ...
145          3.0           5.2   -1
146          2.5           5.0   -1
147          3.0           5.2   -1
148          3.4           5.4   -1
149          3.0           5.1   -1

[100 rows x 3 columns]
/home/shreja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:7: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

Training For Perceptron

In [60]:

```
data_aug_12 = pd.DataFrame(list(zip(iris_data_12['petal.length'],iris_data_12['sepal.wi
dth'], [1 for i in range(len(iris_data_12))],iris_data_12['var'])), columns=['petal.len
gth','sepal.width','augment','var'])
print(data_aug_12)
```

```
    petal.length  sepal.width  augment  var
0            1.4          3.5        1    1
1            1.4          3.0        1    1
2            1.3          3.2        1    1
3            1.5          3.1        1    1
4            1.4          3.6        1    1
..           ...          ...      ...  ...
95           4.2          3.0        1   -1
96           4.2          2.9        1   -1
97           4.3          2.9        1   -1
98           3.0          2.5        1   -1
99           4.1          2.8        1   -1

[100 rows x 4 columns]
```

In [61]:

```
data_aug_13 = pd.DataFrame(list(zip(iris_data_13['petal.length'],iris_data_13['sepal.wi
dth'], [1 for i in range(len(iris_data_13))],iris_data_13['var'])), columns=['petal.len
gth','sepal.width','augment','var'])
print(data_aug_13)
```

```
    petal.length  sepal.width  augment  var
0            1.4          3.5        1    1
1            1.4          3.0        1    1
2            1.3          3.2        1    1
3            1.5          3.1        1    1
4            1.4          3.6        1    1
..           ...          ...      ...  ...
95           5.2          3.0        1   -1
96           5.0          2.5        1   -1
97           5.2          3.0        1   -1
98           5.4          3.4        1   -1
99           5.1          3.0        1   -1

[100 rows x 4 columns]
```

In [62]:

```
data_aug_23 = pd.DataFrame(list(zip(iris_data_23['petal.length'],iris_data_23['sepal.wi
dth'], [1 for i in range(len(iris_data_23))],iris_data_23['var'])), columns=['petal.len
gth','sepal.width','augment','var'])
print(data_aug_23)
```

```
    petal.length  sepal.width  augment  var
0            4.7          3.2        1    1
1            4.5          3.2        1    1
2            4.9          3.1        1    1
3            4.0          2.3        1    1
4            4.6          2.8        1    1
..           ...          ...      ...  ...
95           5.2          3.0        1   -1
96           5.0          2.5        1   -1
97           5.2          3.0        1   -1
98           5.4          3.4        1   -1
99           5.1          3.0        1   -1

[100 rows x 4 columns]
```

Splitting Features and Labels

In [63]:

```
def split(data_aug):
    data_up = data_aug.copy()
    for i in range(len(data_up)):
        if(data_up['var'][i] == 1):
            data_up['petal.length'][i] = - data_up['petal.length'][i]
            data_up['sepal.width'][i] = -data_up['sepal.width'][i]
            data_up['augment'][i] = -data_up['augment'][i]
    # print(data_up)
    data_up = data_up.drop(['var'], axis=1)
    data_up = np.array(data_up)
    # print(data_up)

    labels = data_aug['var']
    labels = np.array(labels)
    # print(labels)

    return (data_up, labels)
```

In [64]:

```
data_up_12, labels_12 = split(data_aug_12)
data_up_13, labels_13 = split(data_aug_13)
data_up_23, labels_23 = split(data_aug_23)
```

```
/home/shreja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:5: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """
/home/shreja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:6: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/home/shreja/.local/lib/python3.6/site-packages/ipykernel_launcher.py:7: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  import sys
```

In [65]:

```
def Perceptron_Train(data_up, labels, eta):

    threshold = 5000
    A = np.zeros(len(data_up[0]))

    # print(A)

    for epoch in range(threshold):
        ctr = 0

        for inp, label in zip(data_up, labels):
            summation = np.dot(inp, A)
            if summation <= 0:
                A += eta*inp
                ctr = 0

            else:
                ctr+=1

        if ctr == len(data_up):     #Until all input is correctly classified
            break
    print(epoch)
    return A
```

In [66]:

```
x1, x2, b = symbols('x1 x2 1')
P = np.array([x1,x2, b])
print(P)
```

```
[x1 x2 1]
```

In [67]:

```
eta = 0.01
A_12 = Perceptron_Train(data_up_12, labels_12, eta)
print(A_12)
```

```
3
[ 0.052 -0.041 -0.01 ]
```

In [68]:

```
Percep_equation_12 = np.dot(A_12.T, P)
print("Decision Boundary between Setosa and Versicolor: ", Percep_equation_12, ' = 0')
```

```
Decision Boundary between Setosa and Versicolor:  -0.01*1 + 0.052*x1 - 0.0
41*x2  = 0
```

In [69]:

```
eta = 0.01
A_13 = Perceptron_Train(data_up_13, labels_13, eta)
print(A_13)
```

```
2
[ 0.032 -0.037 -0.01 ]
```

In [70]:

```
Percep_equation_13 = np.dot(A_13.T, P)
print("Decision Boundary between Setosa and Virginica: ", Percep_equation_13, ' = 0')
```

```
Decision Boundary between Setosa and Virginica:  -0.01*1 + 0.032*x1 - 0.03
7*x2  = 0
```

In [71]:

```
eta = 0.01
A_23 = Perceptron_Train(data_up_23, labels_23, eta)
print(A_23)
```

```
4999
[ 1.296 -0.837 -3.43 ]
```

In [72]:

```
Percep_equation_23 = np.dot(A_23.T, P)
print("Decision Boundary between Versicolor and Virginica: ", Percep_equation_23, ' =
 0')
```

```
Decision Boundary between Versicolor and Virginica:  -3.42999999999997*1 +
1.29600000000016*x1 - 0.837*x2  = 0
```

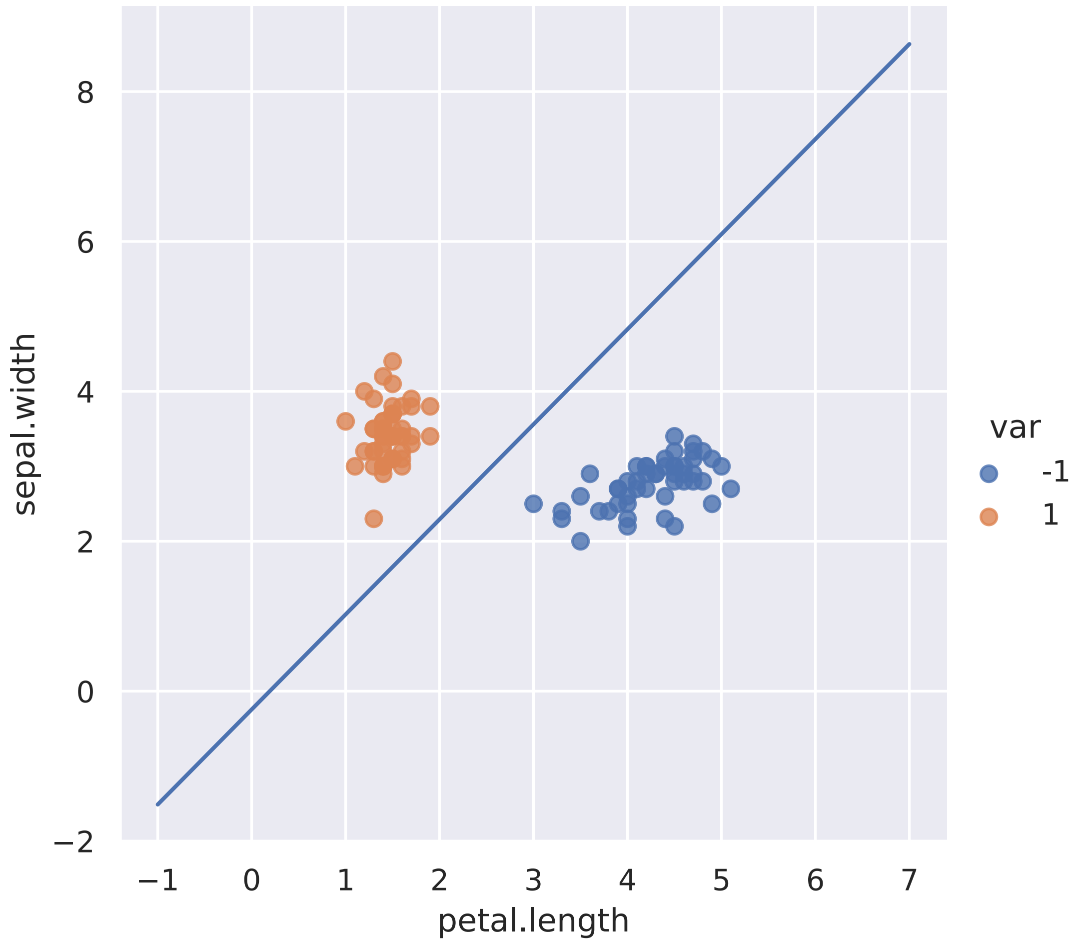
Plotting the Decision Boundary

In [73]:

```
def plot_percep(A, data_aug):
    a = np.linspace(-1,7,100)
    b = -a*A[0]/A[1] - A[2]/A[1]

    sns.set(style="darkgrid")
    sns.lmplot(x='petal.length',y='sepal.width', data=data_aug, fit_reg=False, hue='va
r', legend=True)
    plt.plot(a,b)
    plt.show()
```

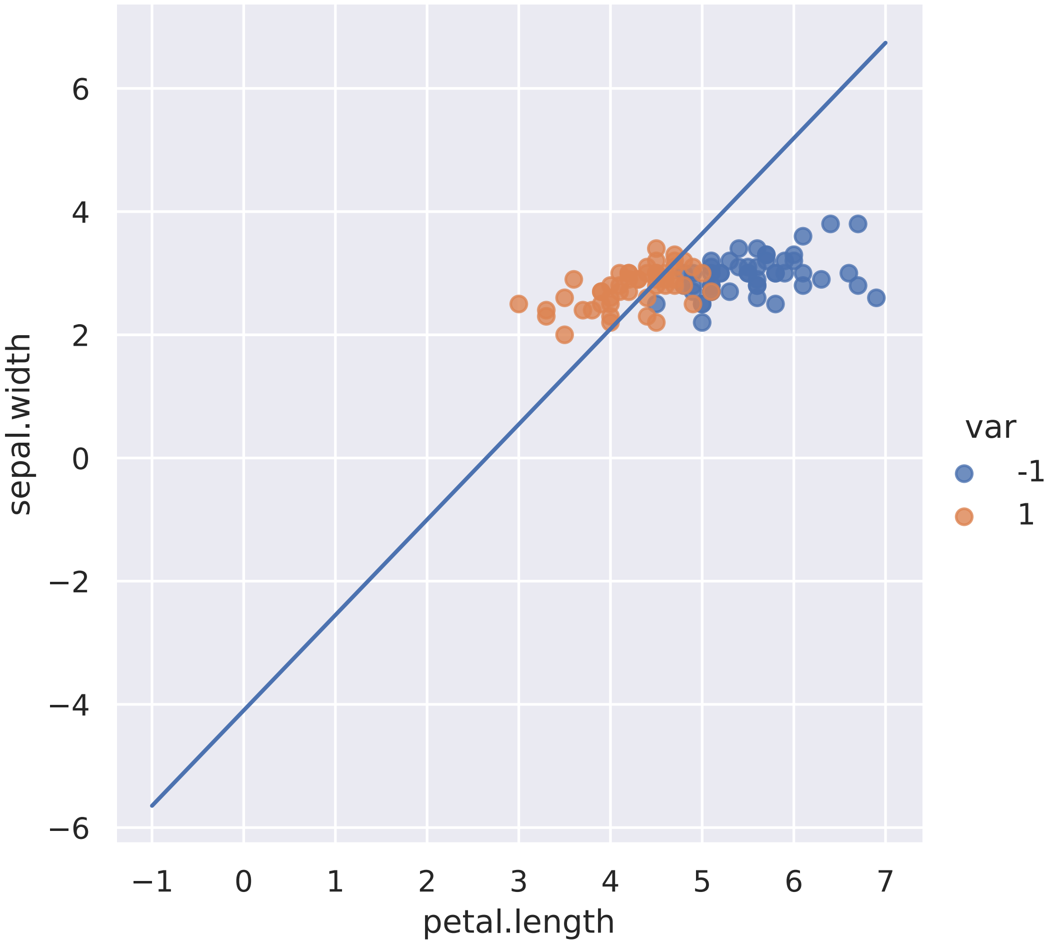
In [74]:

```
plot_percep(A_12,data_aug_12)
```



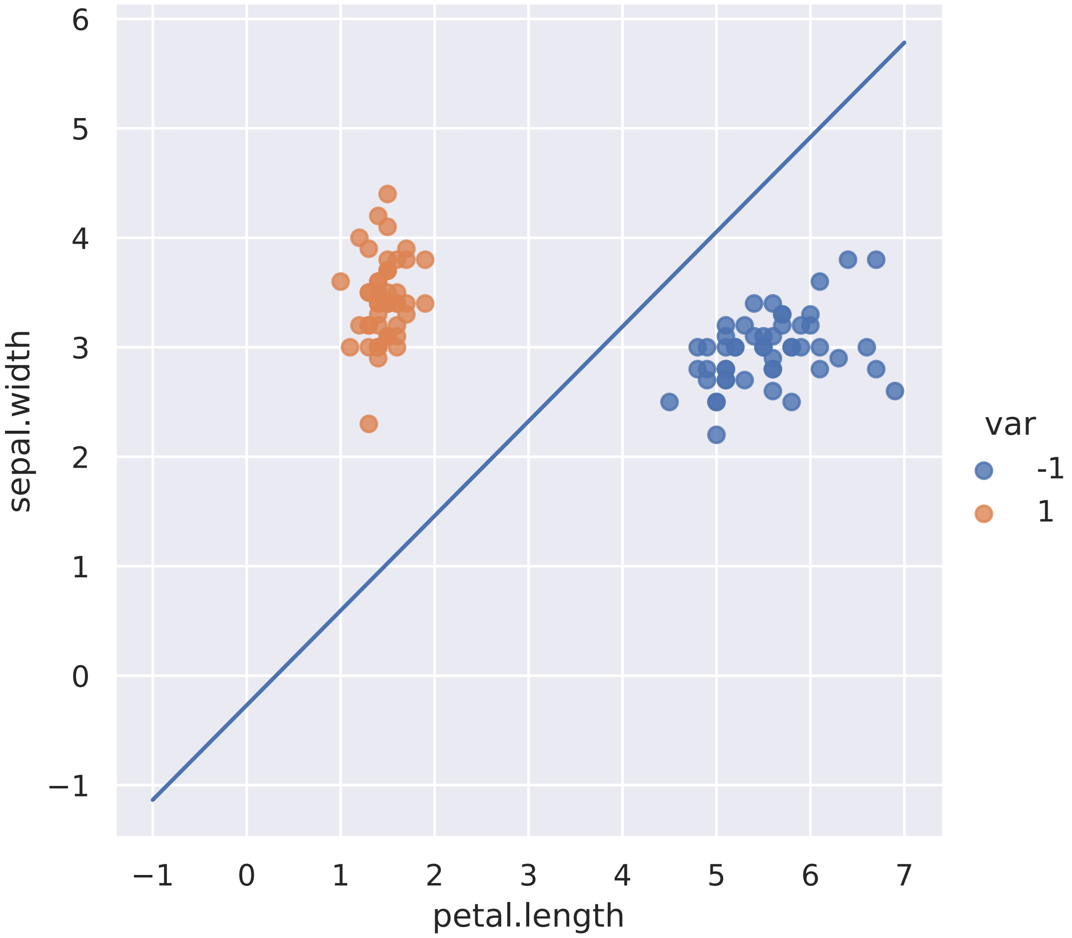
In [75]:

```
plot_percep(A_23,data_aug_23)
```



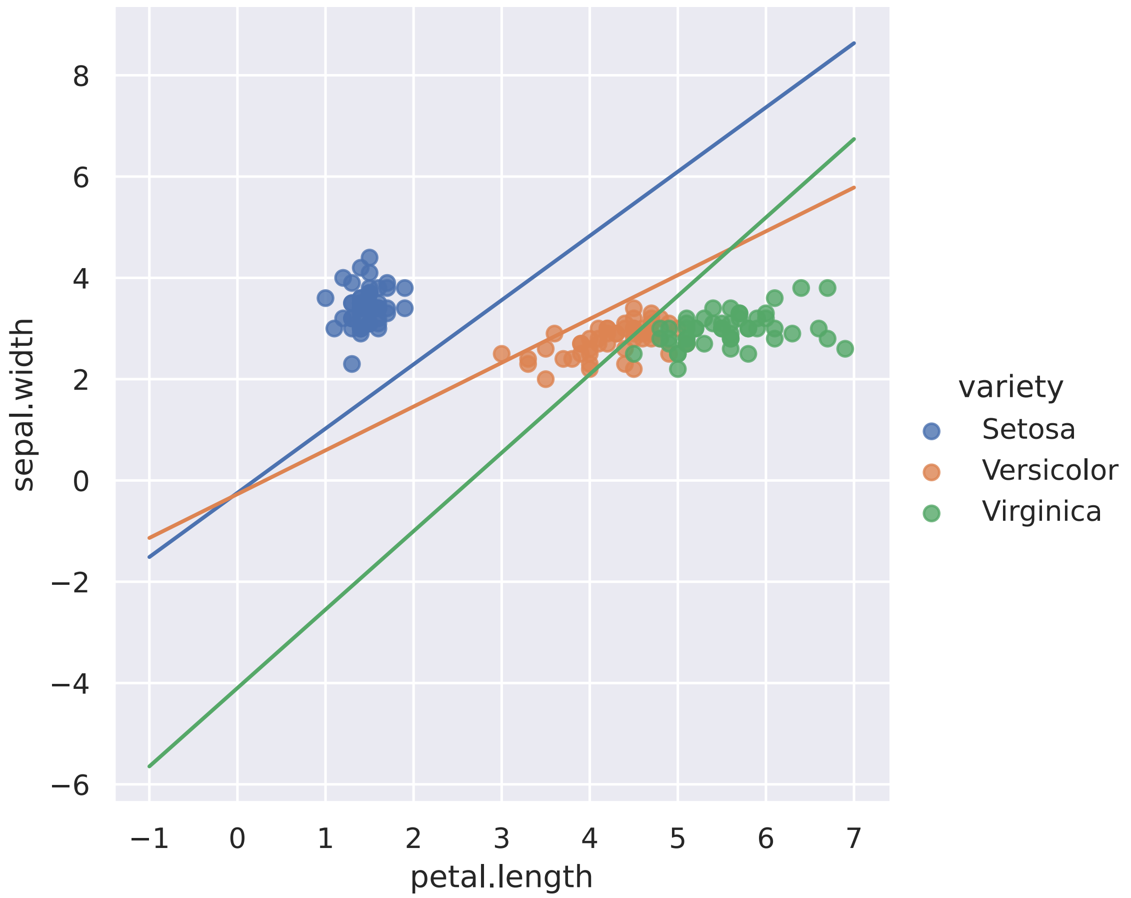
In [76]:

```
plot_percep(A_13, data_aug_13)
```



In [77]:

```
a = np.linspace(-1,7,100)
b_12 = -a*A_12[0]/A_12[1] - A_12[2]/A_12[1]
b_13 = -a*A_13[0]/A_13[1] - A_13[2]/A_13[1]
b_23 = -a*A_23[0]/A_23[1] - A_23[2]/A_23[1]
sns.set(style="darkgrid")
sns.lmplot(x='petal.length',y='sepal.width', data=iris_data, fit_reg=False, hue='variet
y', legend=True)
# plt.scatter(data_up[:50][0], data_up[:50][1])
# plt.scatter(data_up[50:][0],data_up[50:][1])
plt.plot(a,b_12)
plt.plot(a,b_13)
plt.plot(a,b_23)
plt.show()
```



Training with SVM

In [78]:

```
def data_svm_split(iris):
    X = iris[['petal.length','sepal.width']]
    X = np.array(X)
    # print(X)

    Y = iris['var']
    Y = np.array(Y).reshape(len(iris),1)
    # print(Y)

    return (X,Y)
```

In [79]:

```
X_12,Y_12 = data_svm_split(iris_data_12)
X_13,Y_13 = data_svm_split(iris_data_13)
X_23,Y_23 = data_svm_split(iris_data_23)
```

In [80]:

```
def SVM_Train(X,Y):
    m = len(X)

    # print(np.dot(Y,Y.T))
    # print(np.dot(X,X.T))
    P = matrix(np.multiply(np.dot(Y, Y.T), np.dot(X, X.T)))
    # print(P)
    q = matrix(np.ones(m) * -1)
    g1 = np.asarray(np.diag(np.ones(m) * -1))
    # g2 = np.asarray(np.diag(np.ones(m)))
    # G = matrix(np.append(g1, g2, axis=0))
    print(np.array(g1).shape)
    h = matrix(np.zeros(m))
    A = np.reshape((Y.T), (1,m))
    b=[[0]]
    # b = np.array(b).reshape(m,1)

    P = matrix(P,(m,m),'d') #dense
    A = matrix(A,(1,m),'d')
    g1 = matrix(g1,(m,m),'d')
    b = matrix(b,(1,1),'d')

    sol = solvers.qp(P, q, g1, h, A, b)
    alpha = np.array(sol['x'])
    ind = (alpha > 1e-4).flatten()
    print(ind)

    W = np.dot(np.transpose(alpha*Y),X)
    print(W)

    for i in range(m):
        if ind[i] == True:
            W0 = 1 - np.dot(X[i],W.T)
            print
            break

    print(W0)

    return (W, W0)
```

In [81]:

```
x1, x2 = symbols('x1 x2')
P_SVM = np.array([x1,x2])
print(P_SVM)
```

```
[x1 x2]
```

In [82]:

```
W_12,W0_12 = SVM_Train(X_12,Y_12)
```

```
(100, 100)
     pcost       dcost       gap    pres   dres
 0: -4.3867e+00 -8.1716e+00  3e+02  1e+01  2e+00
 1: -2.3798e+00 -2.5258e+00  2e+01  1e+00  2e-01
 2: -4.3479e-01 -1.7538e+00  2e+00  5e-02  6e-03
 3: -6.7595e-01 -1.0544e+00  5e-01  1e-02  1e-03
 4: -8.1435e-01 -1.1260e+00  4e-01  4e-03  5e-04
 5: -1.0085e+00 -1.0271e+00  2e-02  8e-05  1e-05
 6: -1.0251e+00 -1.0253e+00  2e-04  9e-07  1e-07
 7: -1.0253e+00 -1.0253e+00  2e-06  9e-09  1e-09
 8: -1.0253e+00 -1.0253e+00  2e-08  9e-11  1e-11
Optimal solution found.
[False False False False False False False False False False False False
 False False False False False False False False False False False False
  True False False False False False False False False False False False
 False False False False False  True False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False  True False]
[[-1.25714286  0.68571428]]
[1.05714287]
```

In [83]:

```
SVM_equation_12 = np.dot(W_12[0].T, P_SVM) + W0_12[0]
print("Decision Boundary between Setosa and Versicolor: ", SVM_equation_12, ' = 0')
```

```
Decision Boundary between Setosa and Versicolor:  -1.25714285940527*x1 +
0.685714284290053*x2 + 1.05714286628384  = 0
```

In [84]:

```
W_13,W0_13 = SVM_Train(X_13,Y_13)
```

```
(100, 100)
     pcost       dcost       gap    pres   dres
 0: -3.3137e+00 -4.9765e+00  2e+02  2e+01  1e+00
 1: -1.6126e-01 -8.1531e-01  1e+01  9e-01  8e-02
 2:  1.9106e-02 -6.5205e-01  9e-01  1e-02  1e-03
 3: -1.5487e-01 -2.7974e-01  1e-01  7e-04  6e-05
 4: -2.2572e-01 -2.8972e-01  6e-02  1e-04  1e-05
 5: -2.5983e-01 -2.6459e-01  5e-03  1e-05  8e-07
 6: -2.6415e-01 -2.6421e-01  5e-05  1e-07  9e-09
 7: -2.6420e-01 -2.6420e-01  5e-07  1e-09  9e-11
 8: -2.6420e-01 -2.6420e-01  5e-09  1e-11  9e-13
Optimal solution found.
[False False False False False False False False False False False False
 False False False False False False False False False False False False
  True False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False  True False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False]
[[-0.68692206  0.23778071]]
[1.49669752]
```

In [85]:

```
SVM_equation_13 = np.dot(W_13[0].T, P_SVM) + W0_13[0]
print("Decision Boundary between Setosa and Virginica: ", SVM_equation_13, ' = 0')
```

```
Decision Boundary between Setosa and Virginica:  -0.686922063593103*x1 +
0.237780707125038*x2 + 1.49669751660177  = 0
```

In [86]:

```
W_23,W0_23 = SVM_Train(X_23,Y_23)
```

```
(100, 100)
     pcost       dcost       gap    pres   dres
 0: -3.7394e+01 -8.7261e+01  5e+02  2e+01  2e+00
 1: -1.0739e+02 -1.7754e+02  3e+02  1e+01  2e+00
 2: -5.7593e+02 -8.7585e+02  4e+02  1e+01  2e+00
 3: -1.9989e+03 -2.2838e+03  3e+02  1e+01  1e+00
 4: -6.2946e+03 -6.9640e+03  7e+02  1e+01  1e+00
 5: -3.5722e+04 -3.7828e+04  2e+03  1e+01  1e+00
 6: -1.6095e+05 -1.6859e+05  8e+03  1e+01  1e+00
 7: -2.7867e+06 -2.8211e+06  3e+04  9e+00  1e+00
 8: -1.2047e+08 -1.2120e+08  7e+05  9e+00  1e+00
 9: -1.9026e+10 -1.9034e+10  8e+06  9e+00  1e+00
10: -1.9238e+10 -1.9247e+10  9e+06  9e+00  1e+00
11: -1.9261e+10 -1.9270e+10  9e+06  9e+00  1e+00
12: -3.0216e+10 -3.0229e+10  1e+07  9e+00  1e+00
13: -5.7431e+10 -5.7455e+10  2e+07  9e+00  1e+00
14: -7.1454e+10 -7.1481e+10  3e+07  9e+00  1e+00
15: -7.9138e+10 -7.9165e+10  3e+07  9e+00  1e+00
16: -9.4603e+10 -9.4630e+10  3e+07  9e+00  1e+00
Terminated (singular KKT matrix).
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True]
[[-2.44140625e-04 -7.62939453e-06]]
[1.00117188]
```

In [87]:

```
SVM_equation_23 = np.dot(W_23[0].T, P_SVM) + W0_23[0]
print("Decision Boundary between Versicolor and Virginica: ", SVM_equation_23, ' = 0')
```

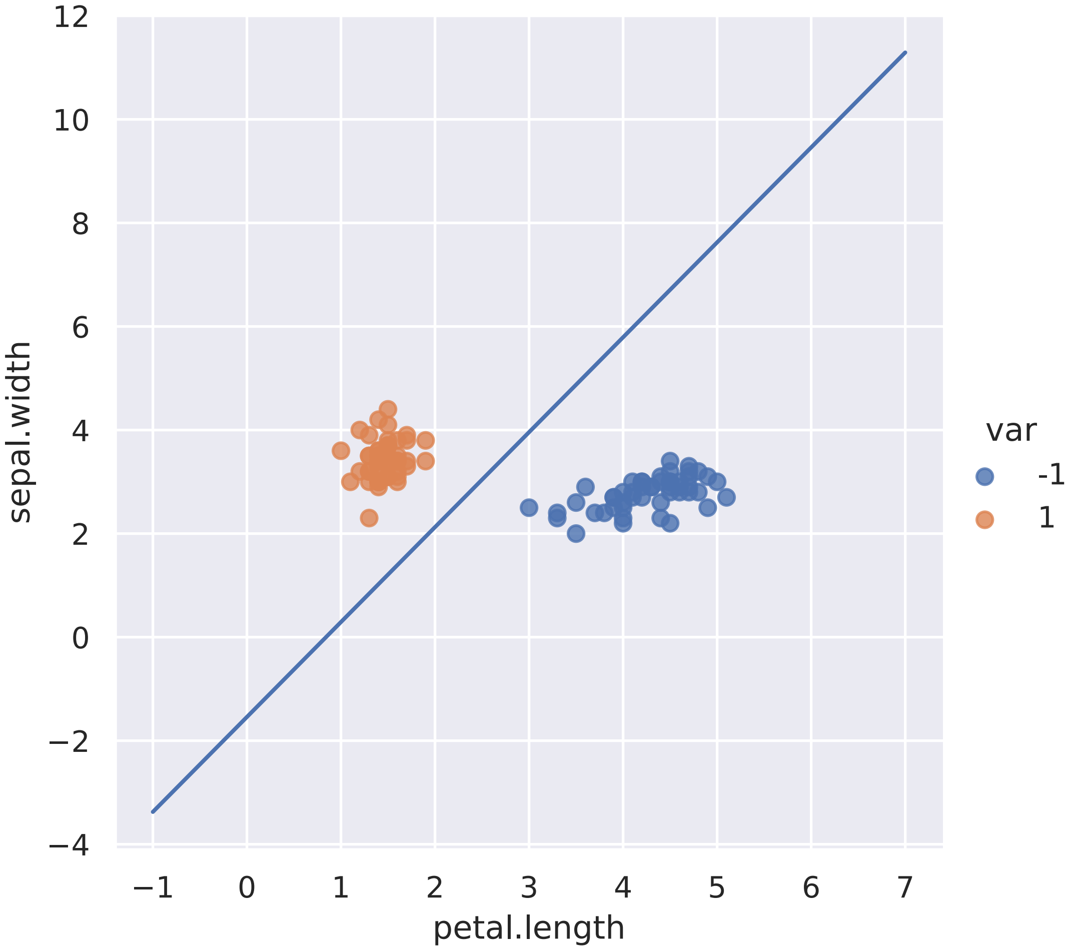
```
Decision Boundary between Versicolor and Virginica:  -0.000244140625*x1 -
7.62939453125e-6*x2 + 1.001171875  = 0
```

In [88]:

```
def plot_SVM(W,W0, data_aug):
    a = np.linspace(-1,7,100)
    b = -a*W[0][0]/W[0][1] - W0[0]/W[0][1]
    sns.set(style="darkgrid")
    sns.lmplot(x='petal.length',y='sepal.width', data=data_aug, fit_reg=False, hue='va
r', legend=True)
    plt.plot(a,b)
    plt.show()
```

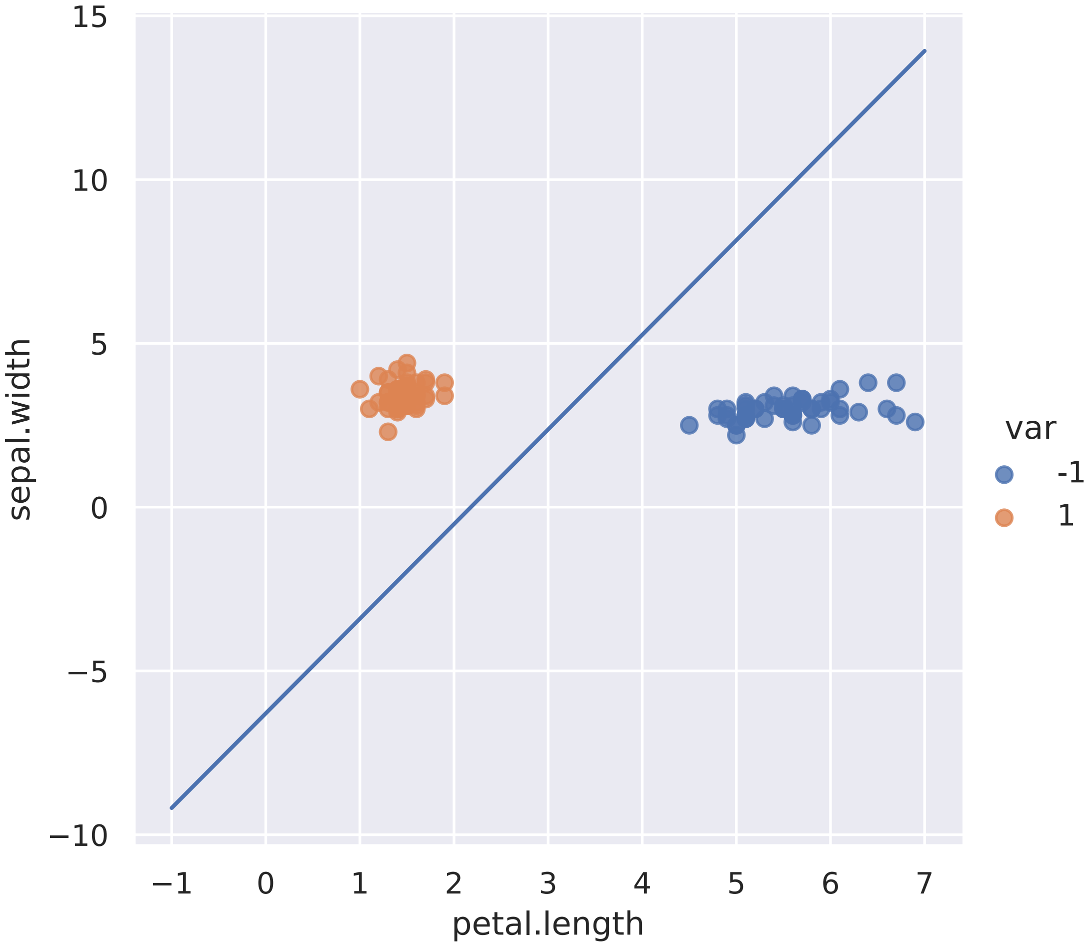
In [89]:

```
plot_SVM(W_12,W0_12,iris_data_12)
```



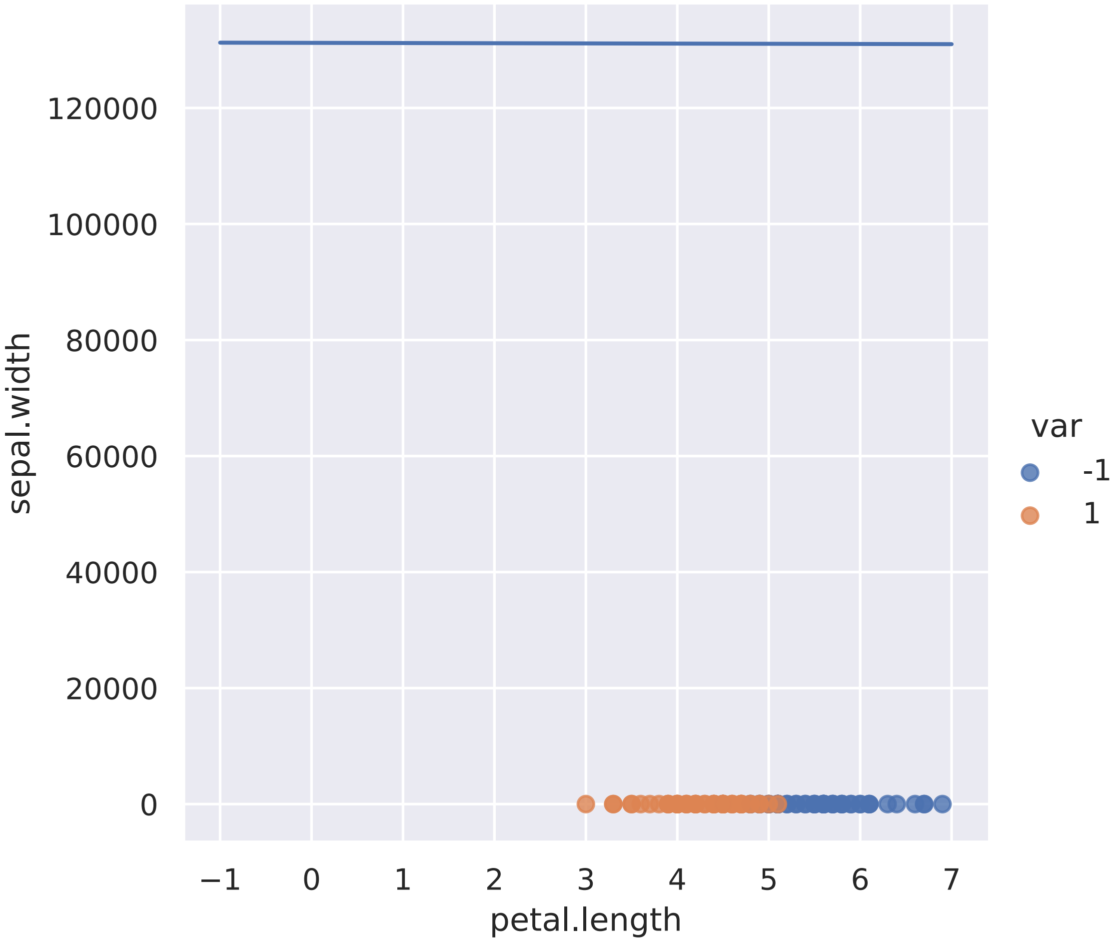
In [90]:

```
plot_SVM(W_13,W0_13,iris_data_13)
```



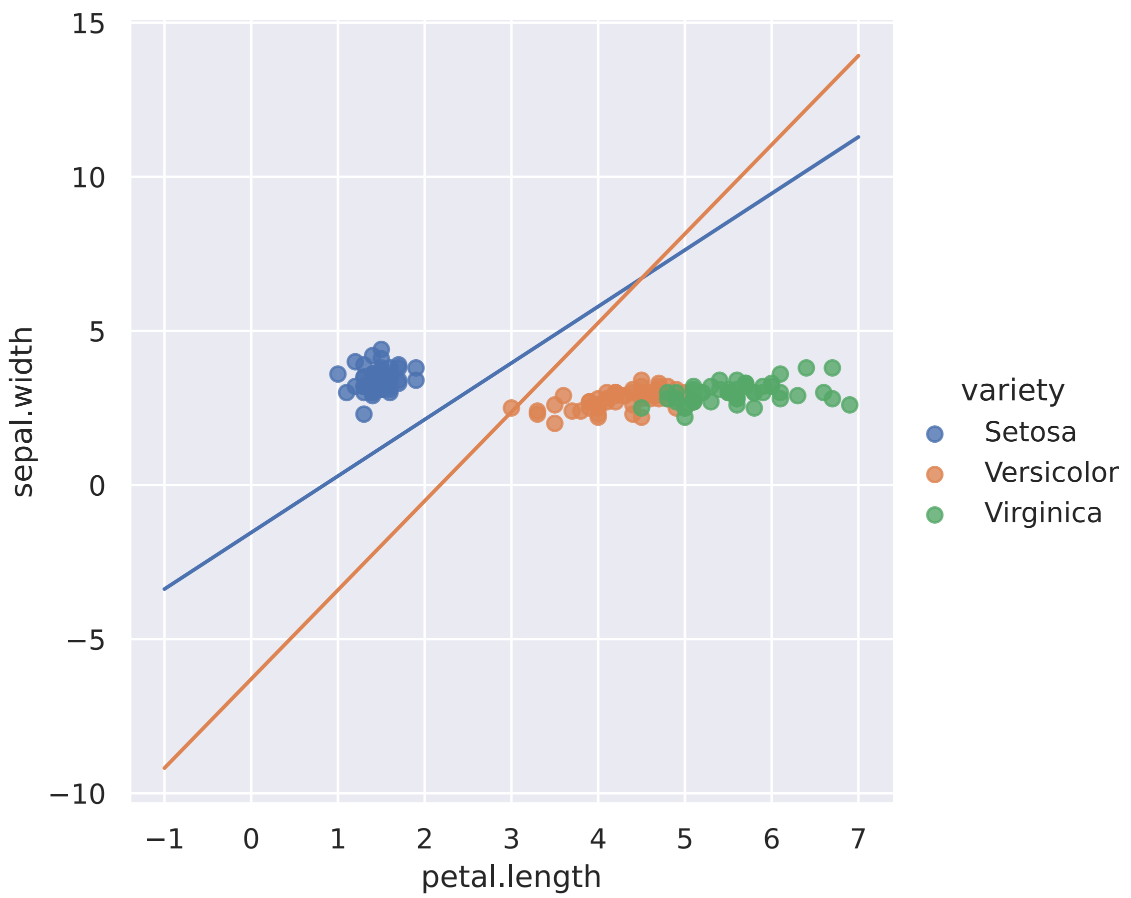
In [91]:

```
plot_SVM(W_23,W0_23,iris_data_23)
```



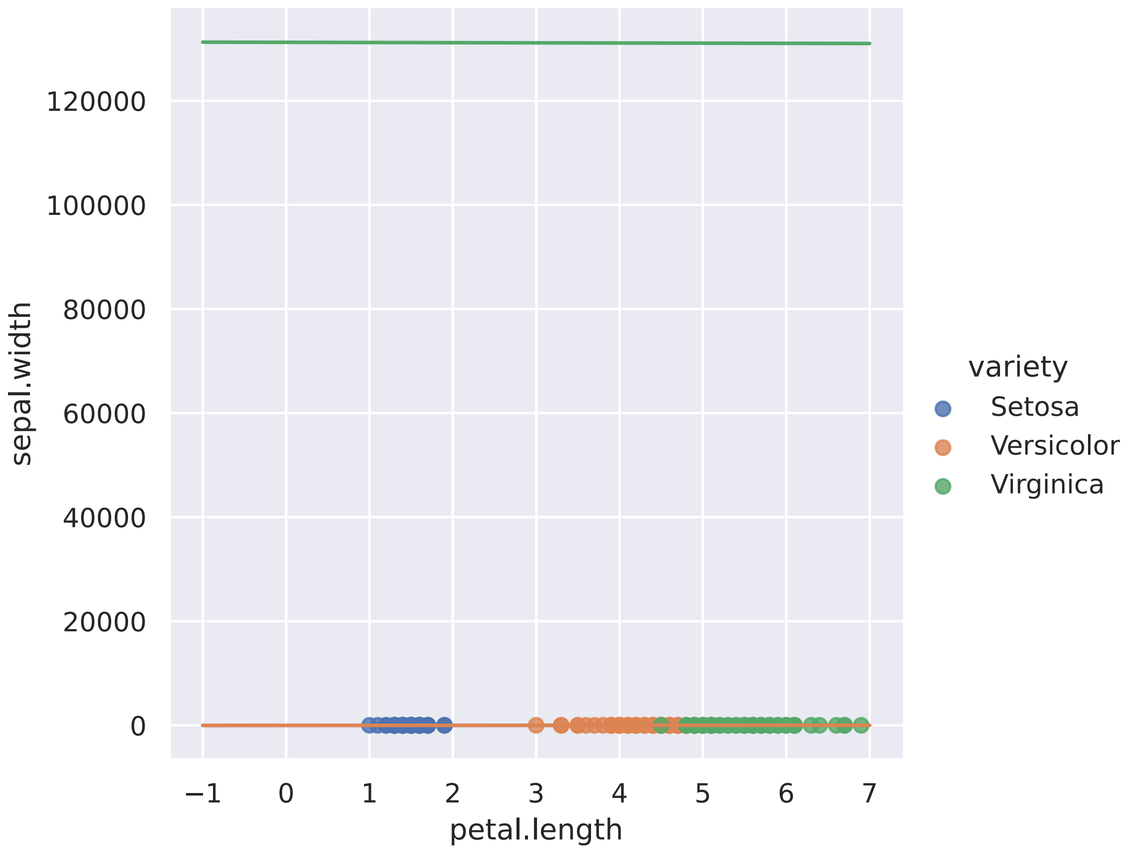
In [92]:

```
a = np.linspace(-1,7,100)
b_12 = -a*W_12[0][0]/W_12[0][1] - W0_12[0]/W_12[0][1]
b_13 = -a*W_13[0][0]/W_13[0][1] - W0_13[0]/W_13[0][1]
# b_23 = -a*W_23[0][0]/W_23[0][1] - W0_23[0]/W_23[0][1]
sns.set(style="darkgrid")
sns.lmplot(x='petal.length',y='sepal.width', data=iris_data, fit_reg=False, hue='variet
y', legend=True)
# plt.scatter(data_up[:50][0], data_up[:50][1])
# plt.scatter(data_up[50:][0],data_up[50:][1])
plt.plot(a,b_12)
plt.plot(a,b_13)
# plt.plot(a,b_23)
plt.show()
```



In [93]:

```
a = np.linspace(-1,7,100)
b_12 = -a*W_12[0][0]/W_12[0][1] - W0_12[0]/W_12[0][1]
b_13 = -a*W_13[0][0]/W_13[0][1] - W0_13[0]/W_13[0][1]
b_23 = -a*W_23[0][0]/W_23[0][1] - W0_23[0]/W_23[0][1]
sns.set(style="darkgrid")
sns.lmplot(x='petal.length',y='sepal.width', data=iris_data, fit_reg=False, hue='variet
y', legend=True)
# plt.scatter(data_up[:50][0], data_up[:50][1])
# plt.scatter(data_up[50:][0],data_up[50:][1])
plt.plot(a,b_12)
plt.plot(a,b_13)
plt.plot(a,b_23)
plt.show()
```



In [ ]:

```

```

In [ ]:

```

```