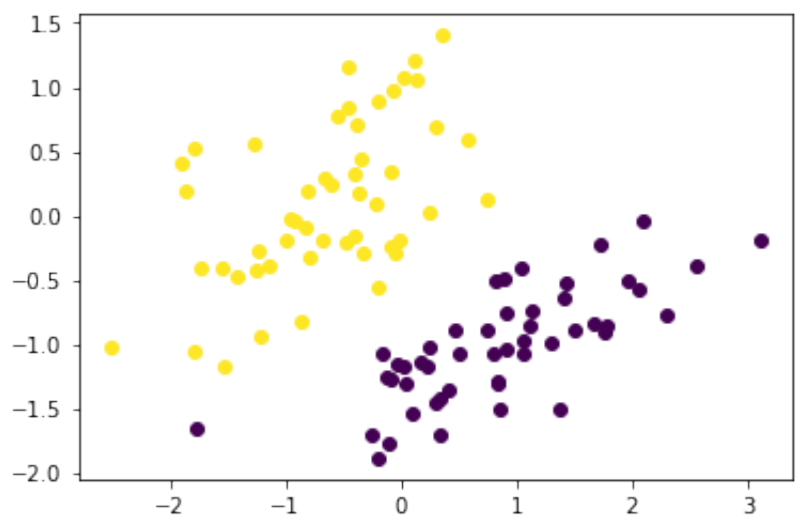
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
from IPython import display
```

Question1

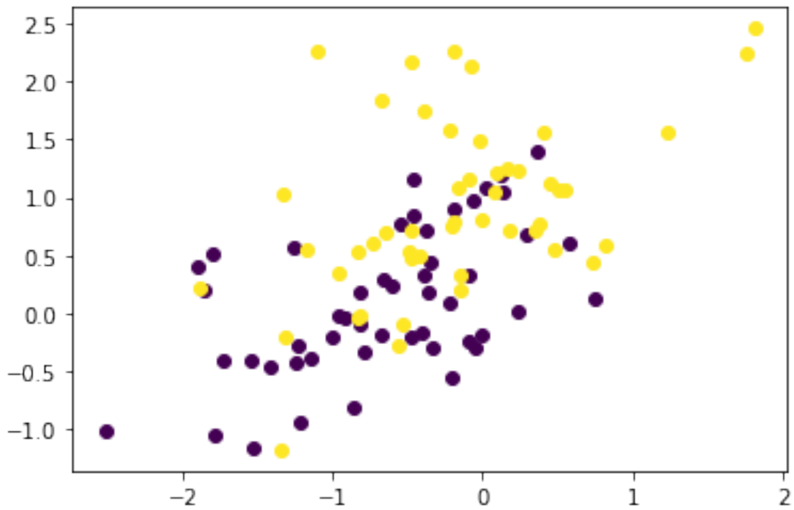
In [2]:
```
iris = np.genfromtxt("data/iris.txt",delimiter=None)
X, Y = iris[:,0:2], iris[:,-1] # get first two features & target
X,Y = ml.shuffleData(X,Y) # order randomly rather than by class label
X,_ = ml.transforms.rescale(X) # rescale to improve numerical stabilit
y, speed convergence
XA, YA = X[Y<2,:], Y[Y<2] # Dataset A: class 0 vs class 1
XB, YB = X[Y>0,:], Y[Y>0] # Dataset B: class 1 vs class 2
```

1.1

In [3]:
```
ml.plotClassify2D(None,XA,YA)
plt.show()
```



In [4]:
```
ml.plotClassify2D(None,XB,YB)
plt.show()
```

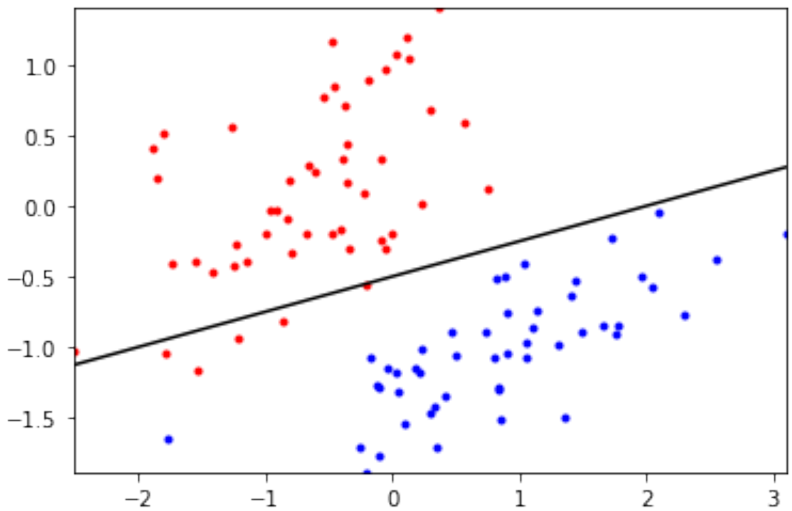


With regarding these two plots, we can see that data set A is linearly separable, and data set B is not.

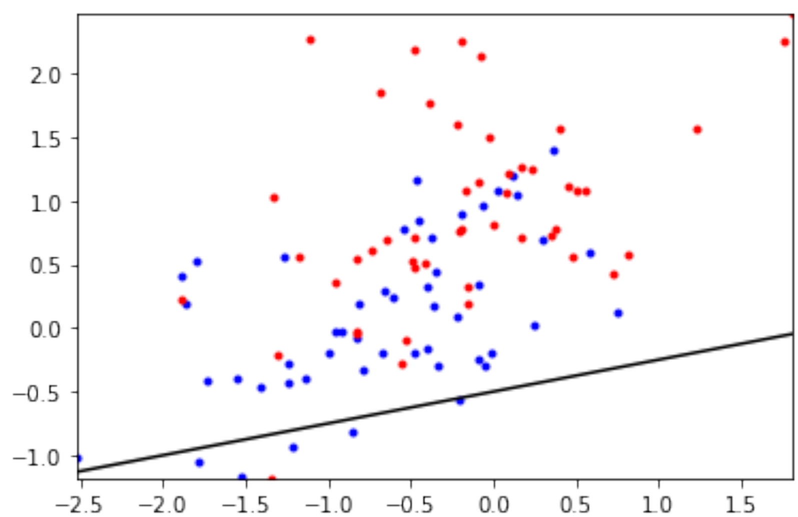
1.2

In [5]:
```
 def myplotBoundary(self,X,Y):
        """ Plot the (linear) decision boundary of the classifier, alo
ng with data """
        if len(self.theta) != 3: raise ValueError('Data & model must b
e 2D');
        ax = X.min(0),X.max(0); ax = (ax[0][0],ax[1][0],ax[0][1],ax[1]
[1]);
        ## TODO: Find two points on decision boundary defined by theta
0 + theta1 X1 + theta2 X2 == 0
        x1b = np.array([ax[0],ax[1]]);  # The X1 coordinates of the tw
o points
        x2b = (-self.theta[0]-self.theta[1]*x1b)/self.theta[2]      #
TODO: Find corresponding X2 coordinates of the two points
        ## Now plot the data and the resulting boundary:
        A = Y==self.classes[0]; # and plot it:
        plt.plot(X[A,0],X[A,1],'b.',X[~A,0],X[~A,1],'r.',x1b,x2b,'k-')
; plt.axis(ax); plt.draw();

import logisticClassify2 as lc2

class logisticClassify2(ml.classifier):
    plotBoundary = myplotBoundary
```

In [6]:
```
learnerA = logisticClassify2()
learnerA.classes = np.unique(YA)
learnerA.theta = np.array( [0.5,-0.25,1] )
myplotBoundary(learnerA,XA,YA)
plt.show()
```



In [7]:
```
learnerB = logisticClassify2()
learnerB.classes = np.unique(YB)
learnerB.theta = np.array( [0.5,-0.25,1] )
myplotBoundary(learnerB,XB,YB)
plt.show()
```



1.3

In [8]:
```
def mypredict(self, X):
    """ Return the predicted class of each data point in X"""
    ## TODO: compute linear response r[i] = theta0 + theta1 X[i,1] + t
heta2 X[i,2] + ... for each i
    ## TODO: if r[i] > 0, predict class 1:  Yhat[i] = self.classes[1]
    ##       else predict class 0:  Yhat[i] = self.classes[0]
    r = self.theta[0] + X.dot(self.theta[1:])
    Yhat = np.zeros(r.shape)
    Yhat[np.where(r > 0)] = self.classes[1]
    Yhat[np.where(r <= 0)] = self.classes[0]
    return Yhat


import logisticClassify2 as lc2
class logistic2(lc2.logisticClassify2): # override methods here for so
lu tion doc
    plotBoundary = myplotBoundary
    predict = mypredict
```

In [9]:
```
learnerA = logistic2()
learnerA.classes = np.unique(YA)
learnerA.theta = np.array( [0.5,-0.25,1] )
learnerA.err(XA, YA)
```

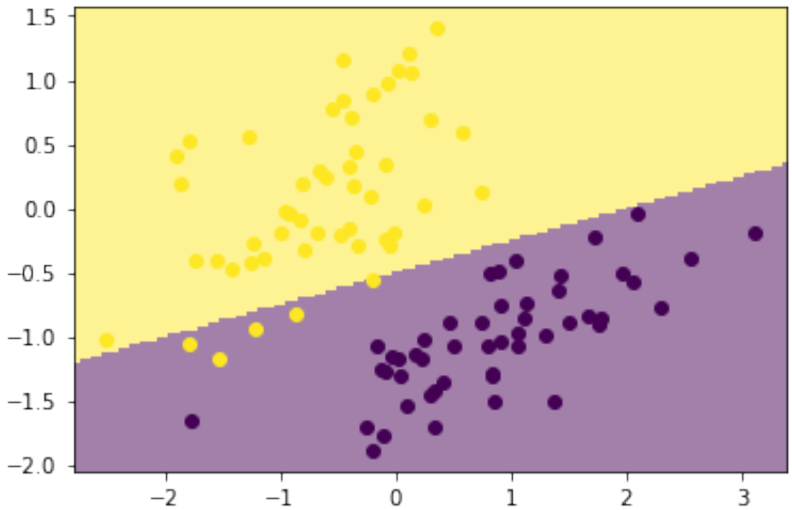
Out[9]: 0.050505050505050504

In [10]:
```
learnerB = logistic2()
learnerB.classes = np.unique(YB)
learnerB.theta = np.array( [0.5,-0.25,1] )
learnerB.err(XB, YB)
```

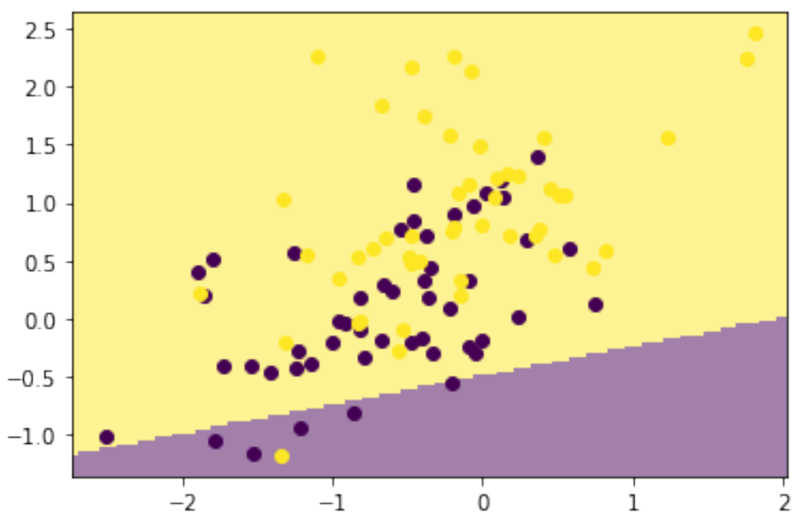
Out[10]: 0.46464646464646464

1.4

In [11]:
```
ml.plotClassify2D(learnerA,XA,YA)
plt.show()
```



In [12]:
```
ml.plotClassify2D(learnerB,XB,YB)
plt.show()
```



1.5

In [13]:
```
from IPython.display import Image
Image(filename = '/Users/wangyiting/Desktop/CS178hw3_code/1.5.jpeg', w
idth=800, height=600)
```

Out[13]:

1.5 $J_j(\theta) = -y^j \log \sigma(x^j \cdot \theta) - (1-y^j) \log(1-\sigma(x^j \cdot \theta))$

$$y^j = \begin{cases} 0 \\ 1 \end{cases} = \begin{cases} -\log(1-\sigma(x^j \cdot \theta)) & y=0 \\ -\log \sigma(x^j \cdot \theta) & y=1 \end{cases}$$

$(y=0)$ $\frac{\partial}{\partial\theta}(-\log(1-\sigma(x^j \cdot \theta)) = \frac{-1}{1-\sigma(x^j \cdot \theta)} \cdot \frac{\partial}{\partial\theta}(\sigma(x^j \cdot \theta))$

$= \frac{-1}{1-\sigma(x^j \cdot \theta)} \cdot \sigma(x^j \cdot \theta) \cdot (1-\sigma(x^j \cdot \theta)) \cdot x^j$

$= -\sigma(x^j \cdot \theta) \cdot x^j$

$(y=1)$ $\frac{\partial}{\partial\theta}(-\log \sigma(x^j \cdot \theta) = \frac{1}{\sigma(x^j \cdot \theta)} \cdot \frac{\partial}{\partial\theta}(\sigma(x^j \cdot \theta))$

$= \frac{1}{\cancel{\sigma(x^j \cdot \theta)}} \cdot \cancel{\sigma(x^j \cdot \theta)}(1-\sigma(x^j \cdot \theta)) \cdot x^j$

$= (1-\sigma(x^j \cdot \theta)) \cdot x^j$

$$\therefore \nabla J_j \theta = \begin{cases} -\sigma(x^j \cdot \theta) \cdot x^j & (y=0) \\ (1-\sigma(x^j \cdot \theta)) \cdot x^j & (y=1) \end{cases}$$

1.6

In [14]:

```
def mytrain(self, X, Y, initStep=1.0, stopTol=1e-4, stopEpochs=5000,
plot=None):
        """ Train the logistic regression using stochastic gradient de
scent """
```

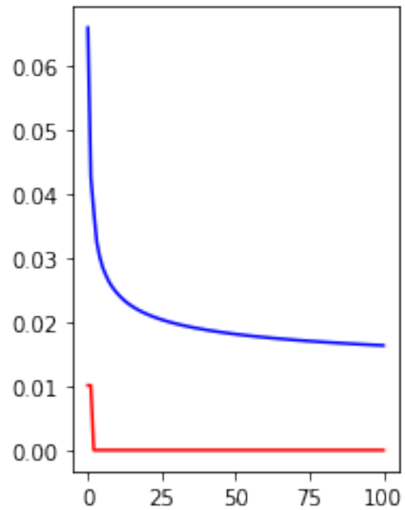
```
        M,N = X.shape;                       # initialize the model if n
ecessary:
        self.classes = np.unique(Y);         # Y may have two classes, a
ny values
        XX = np.hstack((np.ones((M,1)),X)) # XX is X, but with an extr
a column of ones
        YY = ml.toIndex(Y,self.classes);    # YY is Y, but with canonic
al values 0 or 1
        if len(self.theta)!=N+1: self.theta=np.random.rand(N+1);
        # init loop variables:
        epoch=0; done=False; Jnll=[]; J01=[];
        while not done:
            stepsize, epoch = initStep*2.0/(2.0+epoch), epoch+1; # upd
ate stepsize
            # Do an SGD pass through the entire data set:
            for i in np.random.permutation(M):
                ri    =XX[i].dot(self.theta);     # TODO: compute linea
r response r(x)
                si = (1+np.exp(-ri))**(-1)
                gradi = -(1-si) * XX[i,:] if YY[i] else si * XX[i,:];
# TODO: compute gradient of NLL loss
                self.theta -= stepsize * gradi;  # take a gradient ste
p

            J01.append( self.err(X,Y) )  # evaluate the current error
rate

            ## TODO: compute surrogate loss (logistic negative log-lik
elihood)
            ##  Jsur = sum_i [ (log si) if yi==1 else (log(1-si)) ]
            r = XX.dot(self.theta)
            s = (1+np.exp(-r))**(-1)
            Jsur =Jsur = -np.mean(YY*np.log(s)+(1-YY)*np.log(1-s))
            Jnll.append( Jsur ) # evaluate the current NLL loss
            display.clear_output(wait=True);
            plt.subplot(1,2,1); plt.cla(); plt.plot(Jnll,'b-',J01,'r-'
)
#             plt.figure(1); plt.plot(Jnll,'b-',J01,'r-'); plt.draw();
# plot losses
            if N==2: plt.figure(2); self.plotBoundary(X,Y); plt.draw()
; # & predictor if 2D
            plt.pause(.01);                      # let OS draw the plot

            ## For debugging: you may want to print current parameters
& losses
            # print self.theta, ' => ', Jnll[-1], ' / ', J01[-1]
            # raw_input()   # pause for keystroke

            # TODO check stopping criteria: exit if exceeded # of epoc
hs ( > stopEpochs)
```

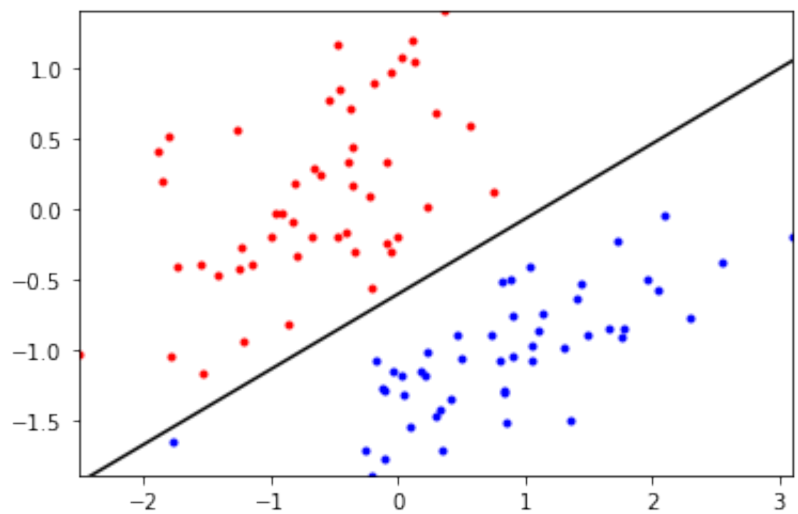
```
            done = epoch > stopEpochs or (epoch > 1 and abs(Jnll[-1] - Jnll[-2]) < stopTol);
            # or if Jnll not changing between epochs ( < stopTol )
```

In [15]:
```
import logisticClassify2 as lc2

class logisticClassify2(ml.classifier):
    plotBoundary = myplotBoundary
    predict = mypredict
    train = mytrain
```

For learnerA,since these points already seperable, I only reduce the stopEpochs to reduce the number of iterations (it is not that difficult enough to need large number of iterations), which can help to avoid overfitting.

In [16]:
```
learnerA = logisticClassify2()
learnerA.theta = np.array([0.,0.,0.]);
learnerA.train(XA,YA,initStep=1,stopEpochs=100,stopTol=1e-5);
```



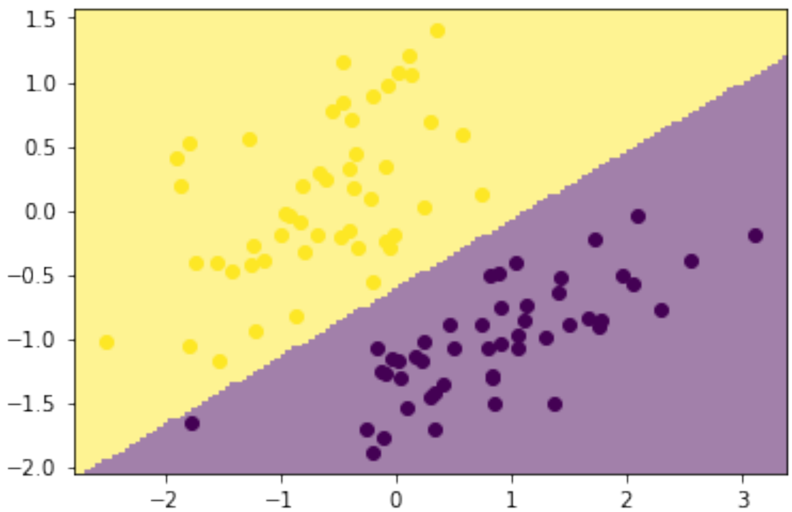


<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

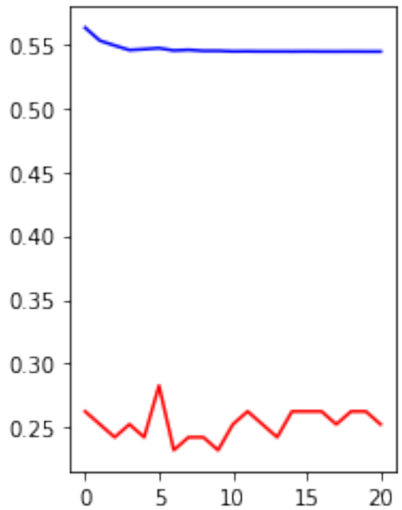
In [17]:
```
ml.plotClassify2D(learnerA,XA,YA)
print(learnerA.err(XA,YA))
plt.show()
```

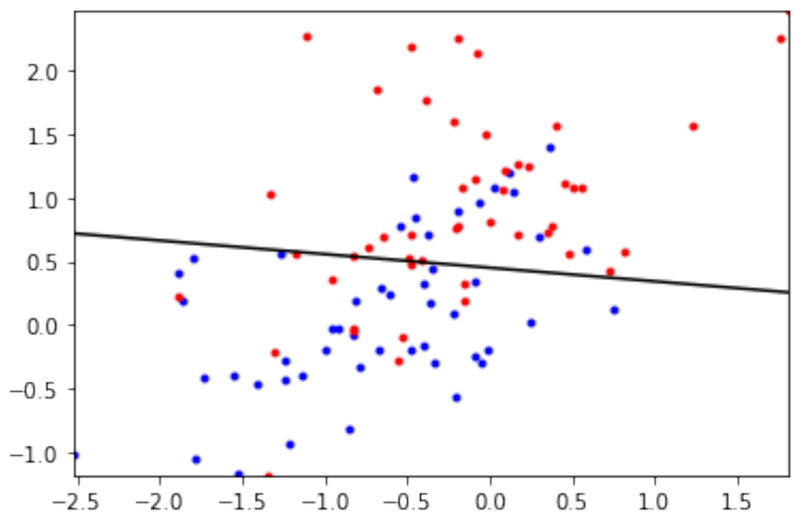
0.0



For learnerB ,I lower the initStep to 0.2 since the points are not seperated well and are mixed together. With the lower initStep, we can get more accurate seperation. Also I increase the stopEpochs to increase the number of iterations, so that it can seperate better for these mixed points.

In [18]:
```
learnerB = logisticClassify2()
learnerB.theta = np.array([0.,0.,0.]);
learnerB.train(XB,YB,initStep=0.2,stopEpochs=6000,stopTol=1e-5);
```

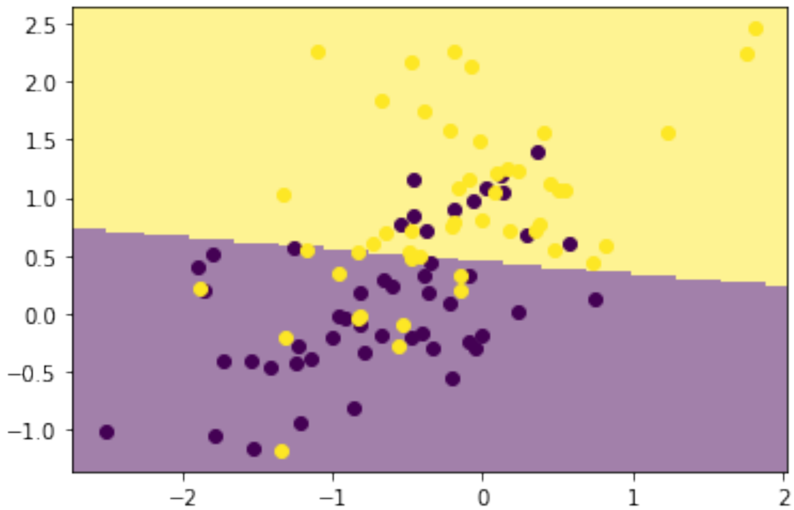





```
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```

In [19]:
```
ml.plotClassify2D(learnerB,XB,YB)
print(learnerB.err(XB,YB))
plt.show()
```

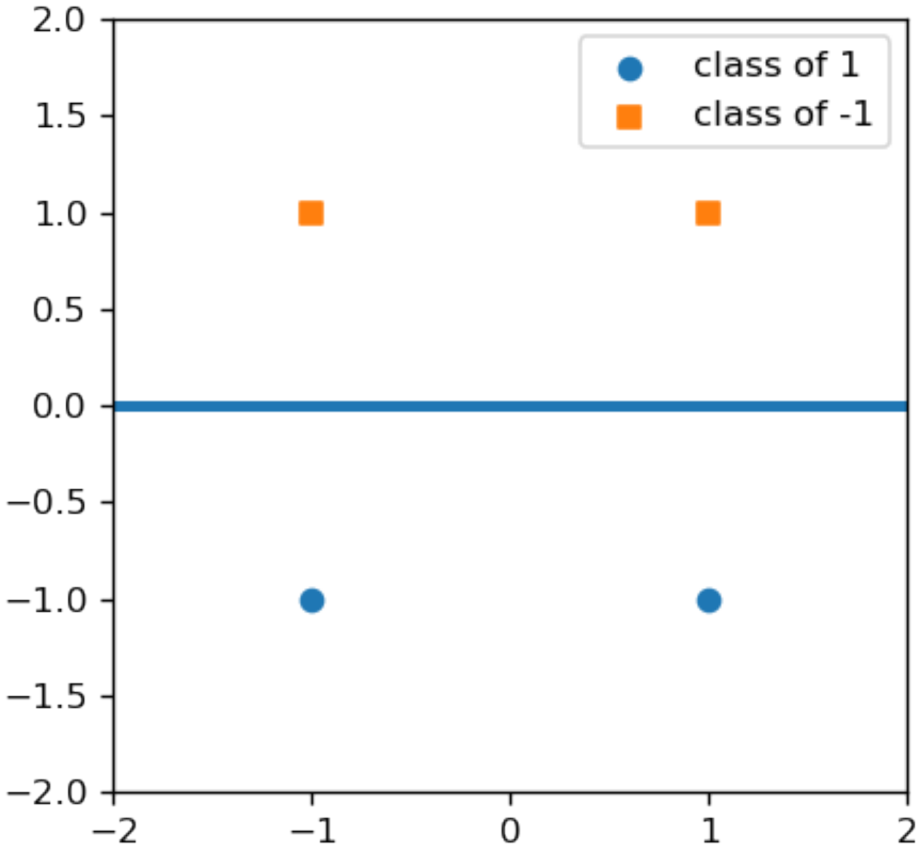
```
0.25252525252525254
```



Question2

2.1

In [20]:
```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(4, 4), dpi=120)
plt.xlim(-2, 2)
plt.ylim(-2, 2)

class1 = np.array([[-1, -1], [1, -1]])
class2 = np.array([[-1, 1], [1, 1]])
plt.scatter(class1[:, 0], class1[:, 1], marker='o')
plt.scatter(class2[:, 0], class2[:, 1], marker='s')
plt.legend(["class of 1","class of -1"])

plt.plot([-2,2],[0,0],linewidth=3)
plt.show()
```

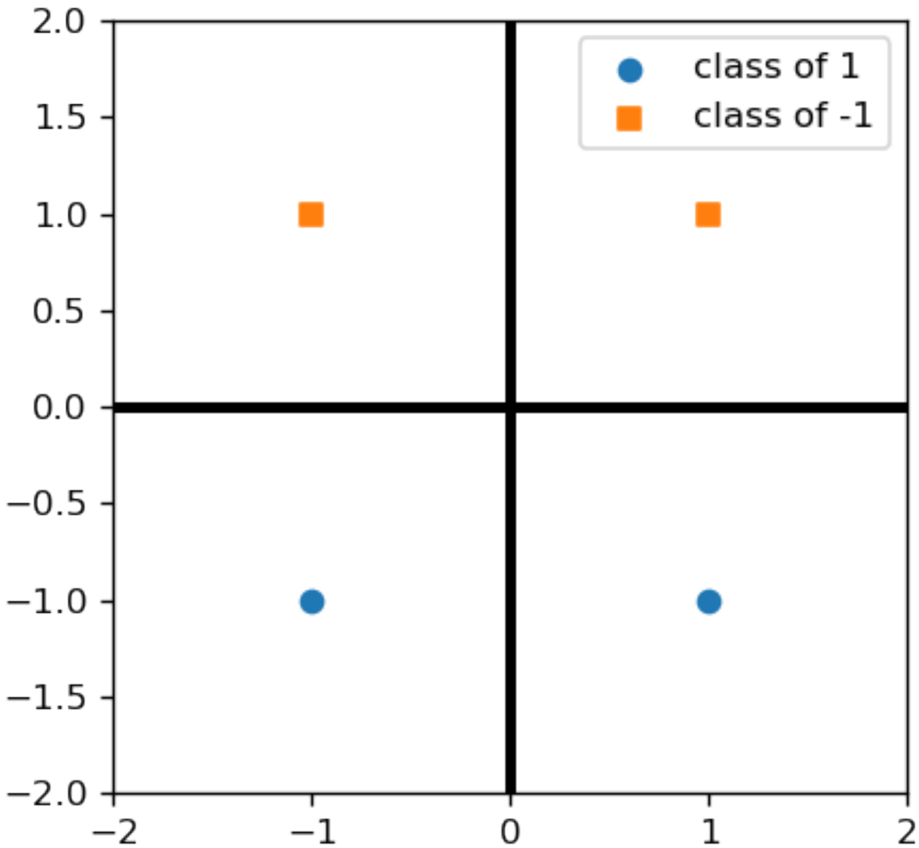


according to the formula that w1*x1+w2*x1*x2=0, and then with the plot I graphed, the line is (x,0), which is x1x2=0. So the w=[0 1].

The coressponding margin between the hyperplanes is |0-1|+|0-(-1)|=2.

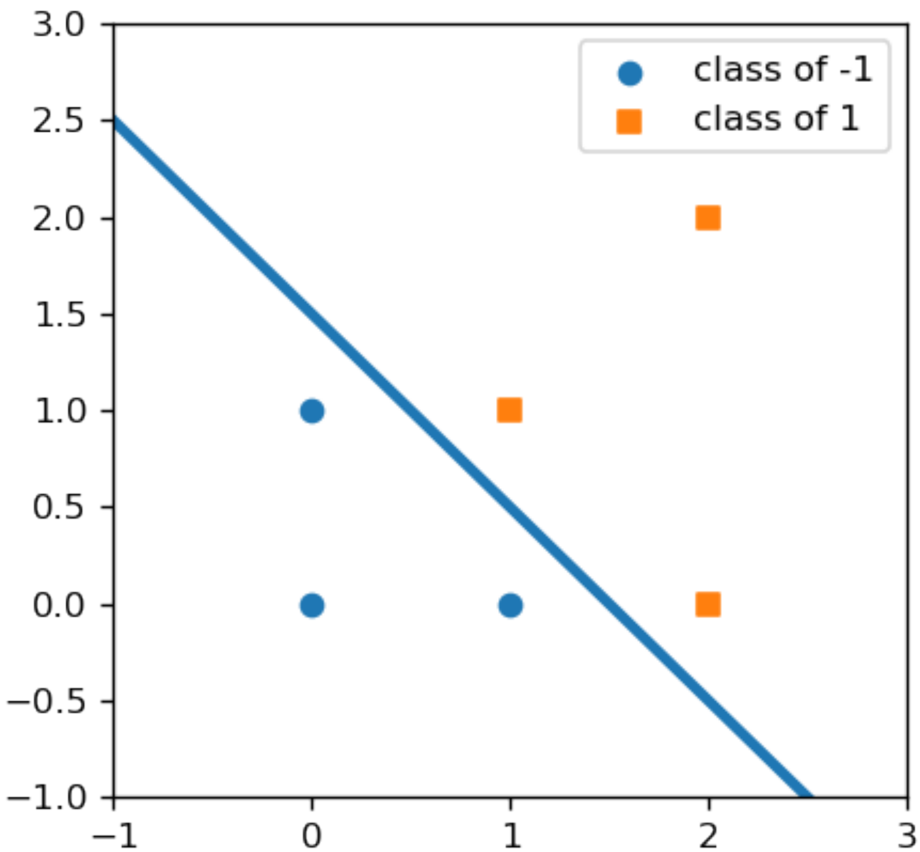
2.2

In [23]:
```
plt.figure(figsize=(4, 4), dpi=120)
plt.xlim(-2, 2)
plt.ylim(-2, 2)

class1 = np.array([[-1, -1], [1, -1]])
class2 = np.array([[-1, 1], [1, 1]])
plt.scatter(class1[:, 0], class1[:, 1], marker='o')
plt.scatter(class2[:, 0], class2[:, 1], marker='s')
plt.legend(["class of 1","class of -1"])

plt.plot([-2,2],[0,0],linewidth=3,color="black")
plt.plot([0,0],[-2,2],linewidth=3,color="black")
plt.show()
```



2.3

In [24]:
```
plt.figure(figsize=(4, 4), dpi=120)
plt.xlim(-1, 3)
plt.ylim(-1, 3)

class1 = np.array([[0, 1], [1, 0],[0, 0]])
class2 = np.array([[2, 0], [1, 1],[2, 2]])
plt.scatter(class1[:, 0], class1[:, 1], marker='o')
plt.scatter(class2[:, 0], class2[:, 1], marker='s')
plt.legend(["class of -1","class of 1"])

plt.plot([-1,2.5],[2.5,-1],linewidth=3)
plt.show()
```



according to the plot, the line x1+x2-1.5=0. By the equation that w1+w2*x1*+*w3*x3=0, we can get w=[-1.5 1 1]

The coressponding margin between the hyperplanes is root 2/4*2=root 2/2=0.707 by calculating the distance of the two cloest points to the line, and the point I choose to use is (0,1) and (1,1).

2.4

(0,0),(2,2) These two points are too far away from the hyper-plane, which can not help to seperate two classes, so they are not support vectors. support vectors are (1,1),(2,0),(0,1),(1,0). for removing the (1,0) or (1,1), the new hyper-plane will be a vertical plane, which will leads to the new optimal-margin to increase to 1. for removing the (2,0),(0,1), the new hyper-plane will not change, so the new optimal-margin will not change .

Question3

I study together with yan yuling, and liu tianle to discusess about the lecture pdf, discussion code, and the online support vector machines study resources about its definitiona and some of its examples. Also we discuss about what each hw questions need to make sure we don't misunderstand questions.