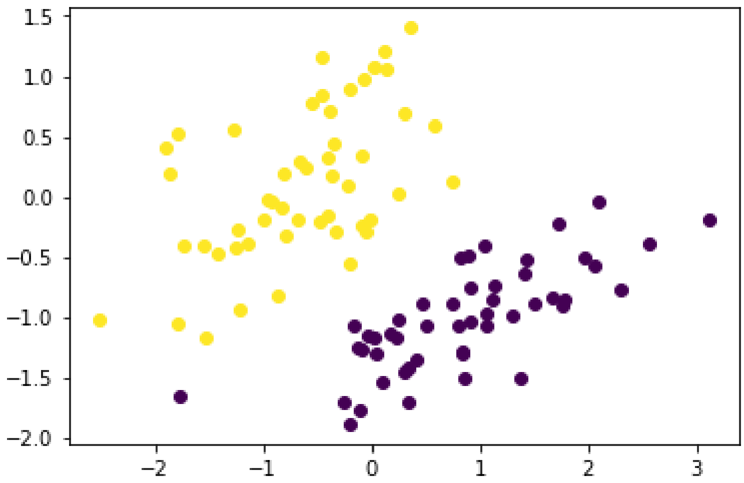
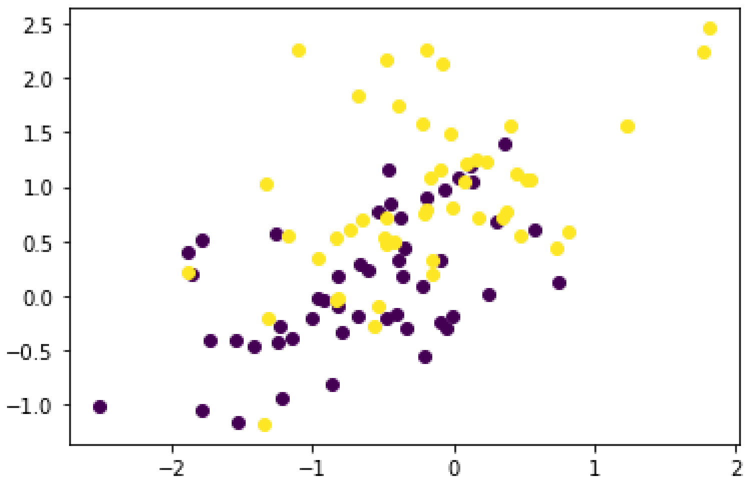
In [1]:
```
import numpy as np

import mltools as ml
import matplotlib.pyplot as plt



iris = np.genfromtxt("data/iris.txt",delimiter=None)
X, Y = iris[:,0:2], iris[:,-1]
X,Y = ml.shuffleData(X,Y)
X,_ = ml.transforms.rescale(X)
XA, YA = X[Y<2,:], Y[Y<2]
XB, YB = X[Y>0,:], Y[Y>0]
```

In [2]:
```
ml.plotClassify2D(None,XA,YA)
plt.show()
```



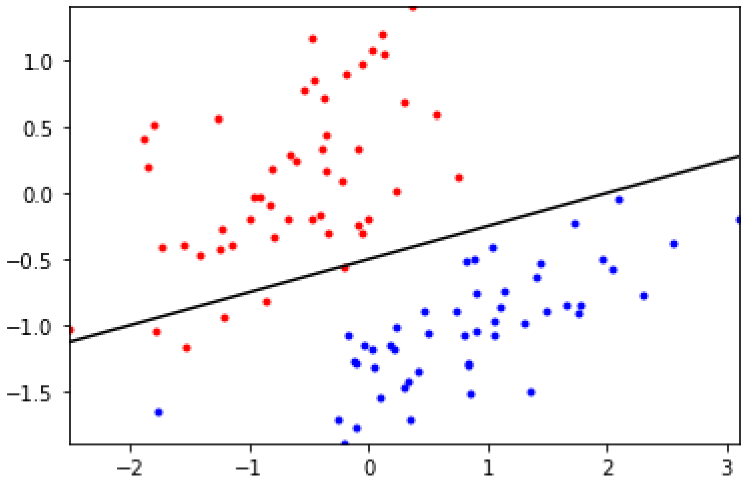
In [3]:
```
ml.plotClassify2D(None,XB,YB)
plt.show()
```

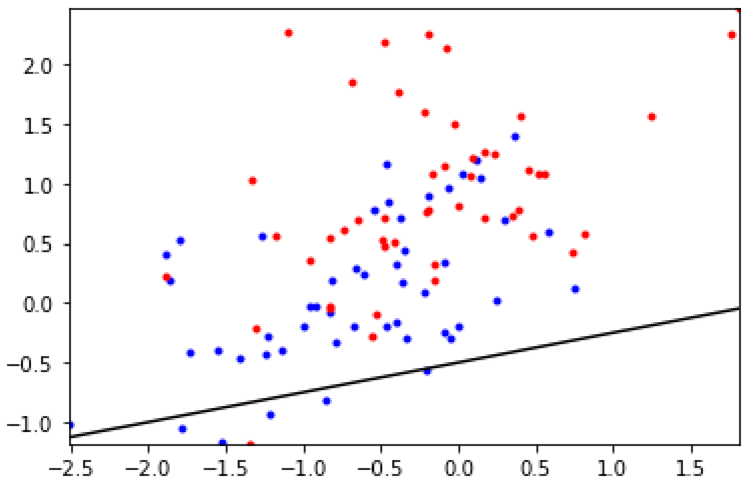


1.1 dataset 1 with XA YA is linearly seperatable

In [4]:
```
#1.2


#code is below
# lines in plot boundary
#x2b = (-self.theta[0]-self.theta[1]*x1b)/self.theta[2]

import mltools as ml
```

```
from logisticClassify2 import *




learner = logisticClassify2(); # create "blank" learner
learner.classes = np.unique(YA) # define class labels using YA or YB
wts = np.array([0.5,-0.25,1])
learner.theta = wts;
learner.plotBoundary(XA,YA)

plt.show()


learner = logisticClassify2(); # create "blank" learner
learner.classes = np.unique(YB) # define class labels using YA or YB
wts = np.array([0.5,-0.25,1])
learner.theta = wts;
learner.plotBoundary(XB,YB)
plt.show()
```





In [5]:

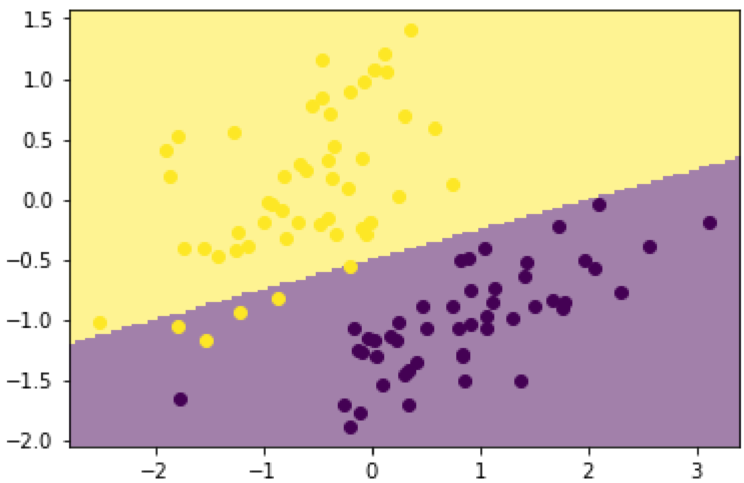
```
#1.3


# def predict(self, X):

#         """ Return the predicted class of each data point in X"""
#         Yhat=[]


#         for z in range(len(X)):
#             sum1 = 0
#             sum1 += self.theta[0]+self.theta[1]* X[z,0] + self.theta[2]* X[z,1]
```

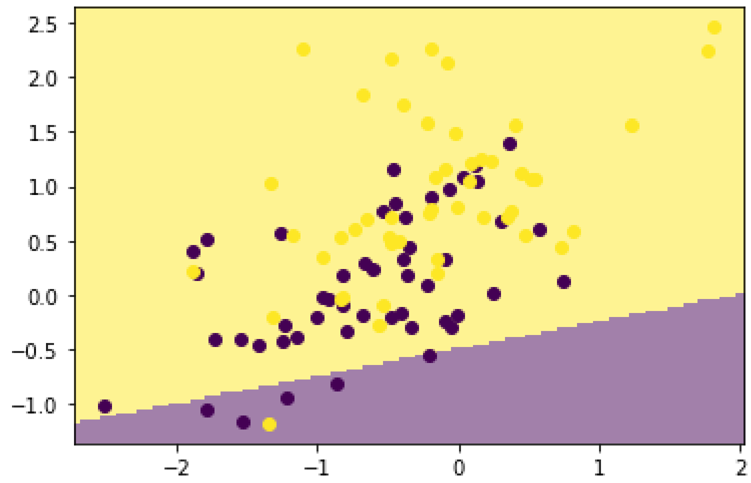
```
#             #print(sum1)
#             if sum1 > 0:
#                 Yhat.append(self.classes[1])
#             else:
#                 Yhat.append(self.classes[0])

#         Yhat1=np.array(Yhat)
#         #print(Yhat1)



#         return Yhat1


learnerA = logisticClassify2()
learnerA.classes = np.unique(YA)
wts = np.array([0.5,-0.25,1])
learnerA.theta = wts;
p1=learnerA.err(XA,YA)

print ("A error rate",learnerA.err(XA,YA))


learnerB = logisticClassify2()
wts= np.array([0.5,-0.25,1])
learnerB.classes = np.unique(YB)
learnerB.theta = wts;



print ("B error rate ",learnerB.err(XB,YB))
```

```
A error rate 0.050505050505050504
B error rate  0.464646464646464
```

In [6]:

```
1.4 #they are consist with before
ml.plotClassify2D(learnerA,XA,YA)
plt.show()
ml.plotClassify2D(learnerB,XB,YB)
plt.show()
```





In [7]:
```
#1.5
#as the prove shows
```

$y^{i} = 1 \quad \log(\sigma(x^{(i)} \cdot \theta))$

$y^{i} = 0 \quad \log(1 - \sigma(x^{(i)} \cdot \theta))$

therefore gradient becomes.

$y^{(i)} = 1 \quad (1 - \sigma(x^{(i)} \cdot \theta))\, x^{(i)}$

$y^{(i)} = 0 \quad -\sigma(x^{(i)} \cdot \theta)\, x^{(i)}$

Combine together

$$\text{gradient} = -y^{(i)}(1 - \sigma(x^{(i)} \cdot \theta))\, x^{(i)} + (1 - y^{(i)})\, \sigma(x^{(i)} \cdot \theta)\, x^{(i)}$$

$$= -y^{(i)} x^{(i)} + y^{(i)} \sigma(x^{(i)} \cdot \theta) x^{(i)} + \sigma(x^{(i)} \cdot \theta) x^{(i)} - y^{(i)} \sigma(x^{(i)} \cdot \theta) x^{(i)}$$

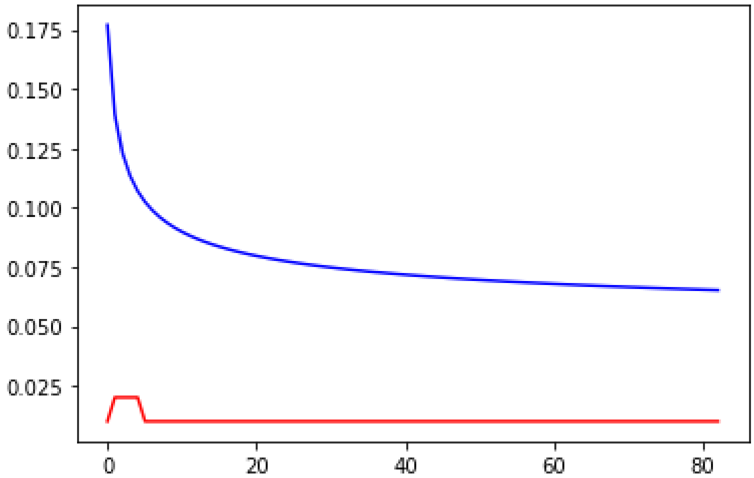
$$= -y^{(i)} x^{(i)} + \sigma(x^{(i)} \cdot \theta)\, x^{(i)}$$

In [8]:
```
#1.6
#in the code below is from train function

def train(self, X, Y, initStep=1.0, stopTol=1e-4, stopEpochs=5000, plot=None):
        """ Train the logistic regression using stochastic gradient descent """
        M,N = X.shape;                      # initialize the model if necessary:
```

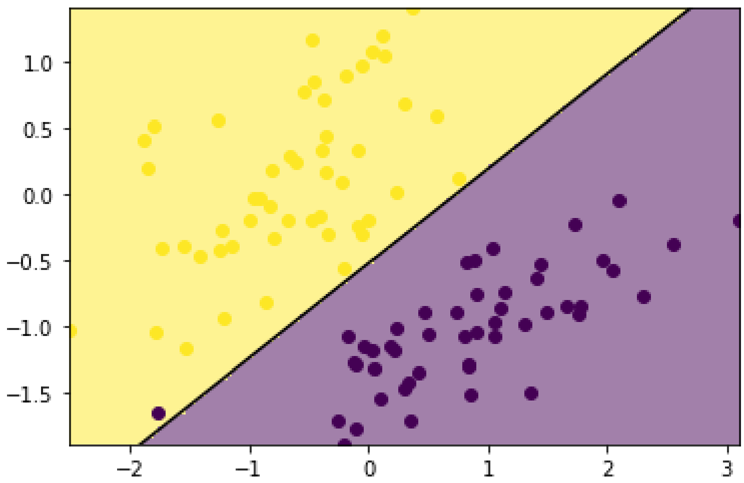
```
        self.classes = np.unique(Y);       # Y may have two classes, any values
        XX = np.hstack((np.ones((M,1)),X)) # XX is X, but with an extra column of one
        YY = ml.toIndex(Y,self.classes);   # YY is Y, but with canonical values 0 or 
        if len(self.theta)!=N+1: self.theta=np.random.rand(N+1);
        # init loop variables:
        epoch=0; done=False; Jnll=[]; J01=[];
        while not done:
            stepsize, epoch = initStep*2.0/(2.0+epoch), epoch+1; # update stepsize
            # Do an SGD pass through the entire data set:
            for i in np.random.permutation(M):

                sum1 = self.theta[0]* XX[i][0]+self.theta[1]* XX[i][1] + self.theta[2


                ri =sum1



                # TODO: compute linear response r(x)
                if YY[i] == 1:
                    gradi =  -(1.-1./(1.+np.exp(-ri)))*XX[i]
                else:
                    gradi =  1./(1.+np.exp(-ri))*XX[i]


                # TODO: compute gradient of NLL loss
                self.theta -= stepsize * gradi;  # take a gradient step

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

            ## TODO: compute surrogate loss (logistic negative log-likelihood)
            ##  Jsur = sum_i [ (log si) if yi==1 else (log(1-si)) ]

            list1=[]

            Jsur1 =[]
            for i in np.random.permutation(M):
                if YY[i]==1:
                    sum1 = self.theta[0]* XX[i][0]+self.theta[1]* XX[i][1] + self.the
                    Si = 1./(1.+np.exp(-(sum1)))
                    Jsur1.append(np.log(Si))
                else:
                    sum1 = self.theta[0]* XX[i][0]+self.theta[1]* XX[i][1] + self.the
                    Si = 1./(1.+np.exp(-(sum1)))
                    Jsur1.append(np.log(1.0-Si))
            a = -np.mean(np.array(Jsur1))



            Jnll.append( a ) # evaluate the current NLL loss


            if N==2:
                pass

            ## 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 epochs ( > stopEpoch

            if epoch>=stopEpochs:
                done =True
            elif epoch>1 and epoch<stopEpochs:
                if (abs(Jnll[-2]-Jnll[-1])<stopTol):
```

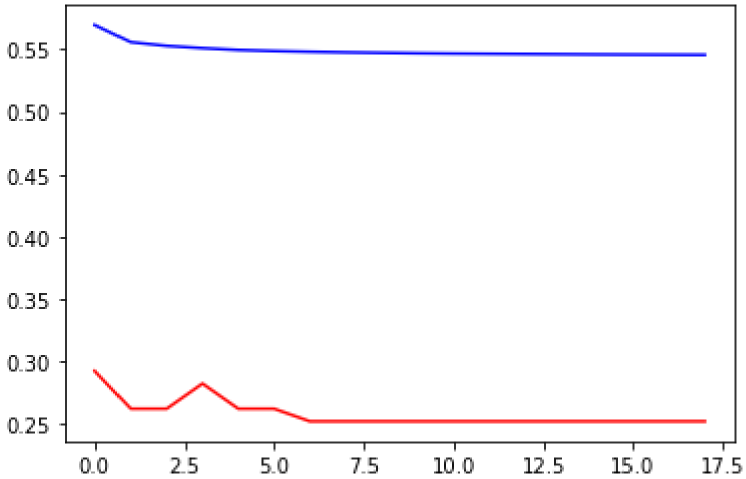
```
                    done =True

            plt.plot(Jnll,'b-',J01,'r-');
            plt.show();
            self.plotBoundary(X,Y);
            plt.draw();
```

In [38]:
```
#1.7
# I want to train is with less stopEpochs, so I reduce the default 5000 to 2000, I set
# the stoptol is using the default value because it is already small enough, the
# initstep is not to small or large because the result can converge
learnerA = logisticClassify2()

learnerA.train(XA,YA,initStep=.1,stopEpochs=2000,stopTol=1e-4);
ml.plotClassify2D(learnerA,XA,YA)
print("A error rate: ",learnerA.err(XA,YA))
plt.show()


learnerB = logisticClassify2()

learnerB.train(XB,YB,initStep=.1,stopEpochs=2000,stopTol=1e-4);
ml.plotClassify2D(learnerB,XB,YB)
print("B error rate: ",learnerB.err(XB,YB))
plt.show()
```

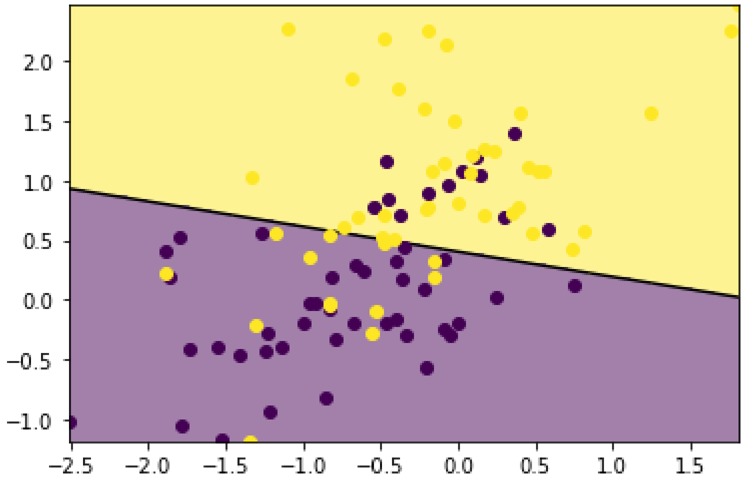


A error rate: 0.010101010101010102



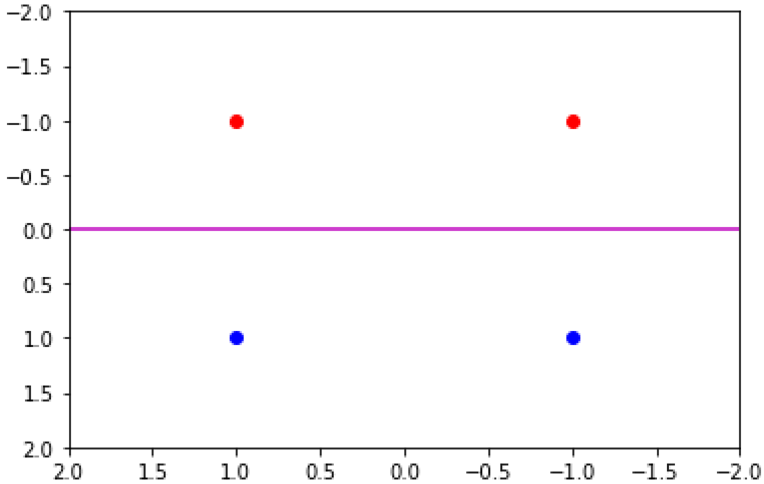


B error rate: 0.25252525252525254



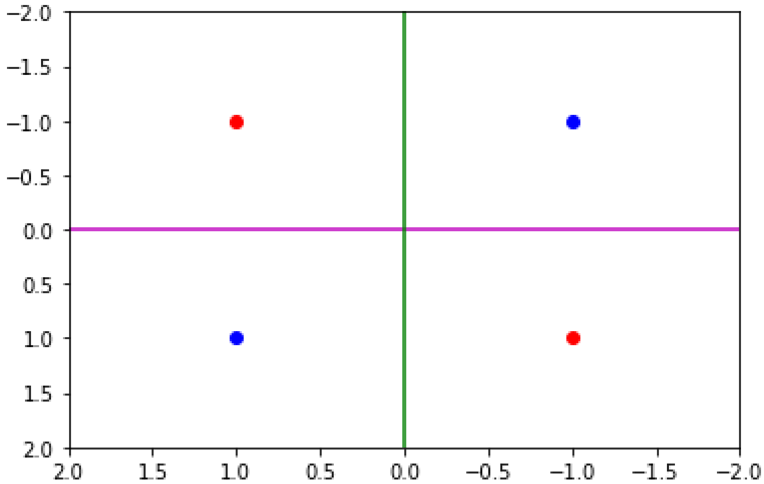
In [77]:

```
#Problem 2.1

xcoord = np.asarray([-1, 1,])
ycoord =np.asarray([1, 1])

plt.plot(xcoord,ycoord,'bo' )
plt.plot(xcoord,-ycoord,'ro' )


xcoord = np.asarray([99,-99])
ycoord = np.asarray([0,0])
plt.xlim(2,-2)
plt.ylim(2,-2)
plt.plot(xcoord, ycoord, '-m')
plt.show()
```



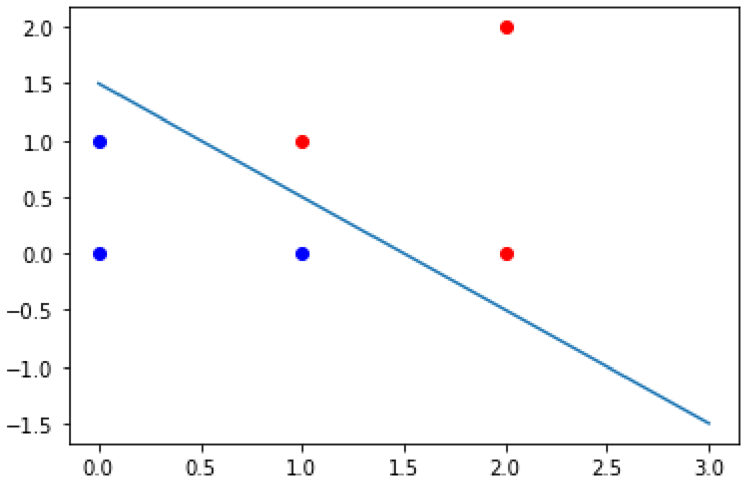
First φ(x) makes point becomes (-1,1) (1,1) blue for y=1 ,(-1,-1) (+1,-1) red for y=-1 then hyperplan can divide x1x2=0 weight vector is [0,1] margin is 1*2 =2

In [54]:
```
#2.2
xcoord = np.asarray([-1,1])
ycoord =np.asarray([-1,1])

plt.plot(xcoord,ycoord,'bo' )

plt.plot(xcoord,-ycoord,'ro' )

xcoord = np.asarray([-99,99])
ycoord = np.asarray([0,0])
plt.xlim(2,-2)
plt.ylim(2,-2)
plt.plot(xcoord, ycoord, '-m')
plt.plot(ycoord, xcoord, '-g')
plt.show()
```



First φ(x) makes point becomes blue for y=1 ,red for y=-1 since w · φ(x) = 0, then hyperplan x1 x2 can both be x1=0, x2=0

In [72]:
```
#2.3

xcoord = np.asarray([0,1,0])
ycoord =np.asarray([0,0,1])

plt.plot(xcoord,ycoord,'bo' )

xcoord = np.asarray([1,2,2])
ycoord =np.asarray([1,2,0])

plt.plot(xcoord,ycoord,'ro' )

x=np.arange(0,4,1)
plt.plot(x,-x+1.5)



#plt.plot(xcoord, ycoord, '-m')
plt.show()
```



red points are class+1, blue points are class-1, two middle point make the speration line (0.5,1)(1.5,0), get y=-x+1.5

after change format, hyper plane is x1+x2 = 1.5

the margin distance is $\sqrt{2}/2$

2.4
points near the seperation line can be support vectors (1,0)(0,1)(1,1)(2,0)

Problem 3

I have do this by myself.