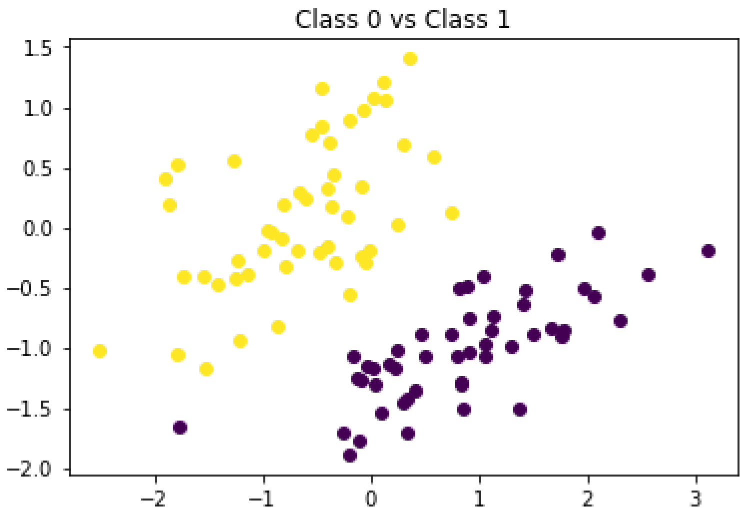
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
```

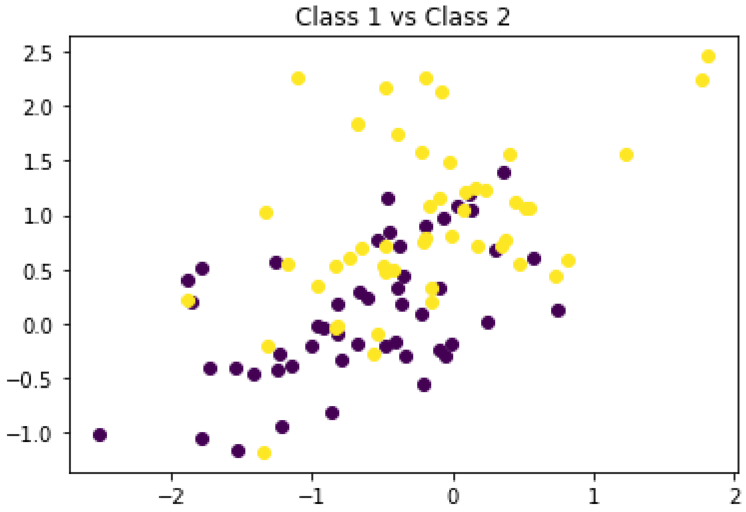
In [2]:
```
iris = np.genfromtxt("data/iris.txt",delimiter=None)
X, Y = iris[:,0:2], iris[:,-1] # get first two features & target
X,Y = ml.shuffleData(X,Y) # reorder randomly (important later)
X,_ = ml.rescale(X) # works much better on rescaled data
XA, YA = X[Y<2,:], Y[Y<2] # get class 0 vs 1
XB, YB = X[Y>0,:], Y[Y>0] # get class 1 vs 2
```

In [3]:
```
#Problem 1
plt.title("Class 0 vs Class 1")
ml.plotClassify2D(None,XA,YA)
plt.show()

plt.title("Class 1 vs Class 2")
ml.plotClassify2D(None,XB,YB)
plt.show()
#we can see clearly here Class 0 vs Class 1 is seperate, and Class 1 vs Class 2 is not
```

```
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the
'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the
'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
```

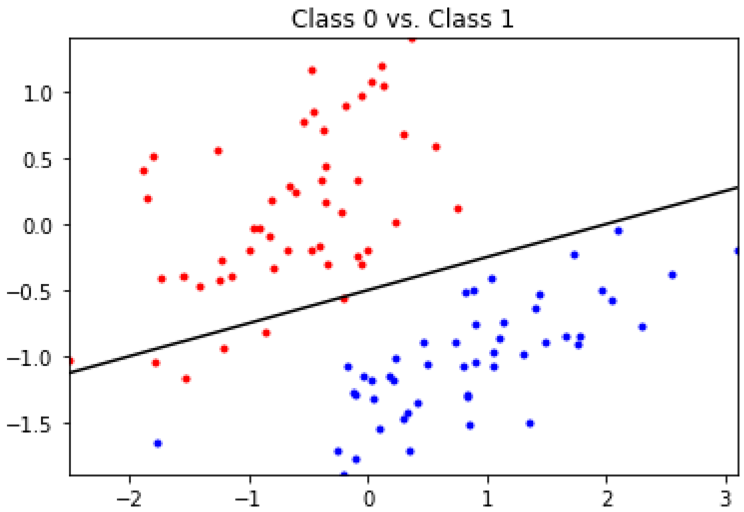


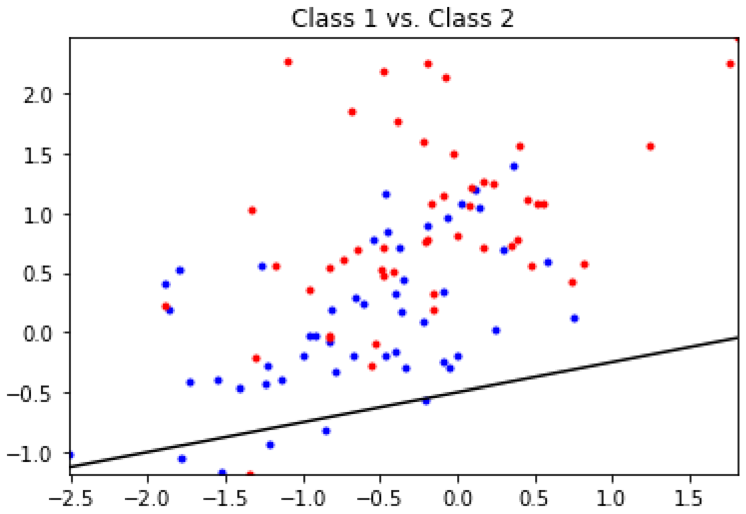

```
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the
'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the
'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
```


Class 1 vs Class 2
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2

In [4]:
```
#Problem 2
from logisticClassify2 import *

plt.figure()
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]); # TODO: fill in values
learner.theta = wts; # fill in the theta values
learner.plotBoundary(XA,YA)
plt.title('Class 0 vs. Class 1')

plt.figure()
learner.classes = np.unique(YB) # define class labels using YA or YB
wts = np.array([0.5,-0.25,1]); # TODO: fill in values
learner.theta = wts; # set the learner's parameters
learner.plotBoundary(XB,YB)
plt.title('Class 1 vs. Class 2')
```

Out[4]:
```
Text(0.5, 1.0, 'Class 1 vs. Class 2')
```



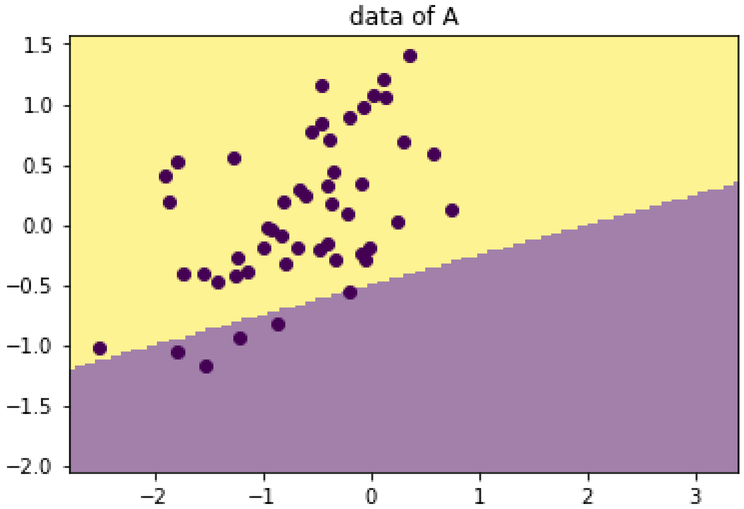


In [5]:
```
# Problem 3
learner = logisticClassify2()
wts = np.array([0.5,-0.25,1])
learner.theta = wts
ErrorA = learner.err(XA,YA)
print('Errors for set A:', ErrorA)

learner.classes= np.unique(YB)
ErrorB = learner.err(XB,YB)
print('Errors for set B:', ErrorB)
```

```
Errors for set A: 0.050505050505050504
Errors for set B: 0.5454545454545454
```

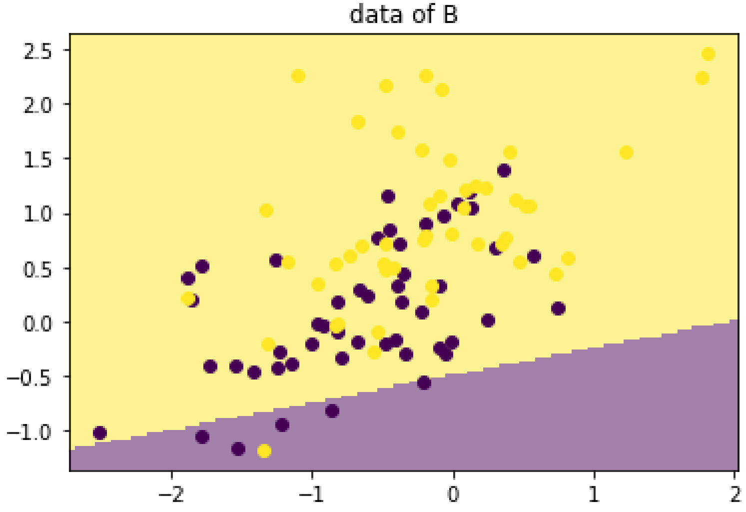
In [6]:
```
#Problem 4
plt.title('data of A')
ml.plotClassify2D(learner, XA,YA)
```

```
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the 'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
```



In [7]:
```
plt.title('data of B')
ml.plotClassify2D(learner, XB,YB)
```

```
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the
'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
C:\Users\10630\273p\assignment 3\Assignment 3\mltools\plot.py:61: UserWarning: color is redundantly defined by the
'color' keyword argument and the fmt string "ko" (-> color='k'). The keyword argument will take precedence.
  axis.plot( X[Y==c,0],X[Y==c,1], 'ko', color=cmap(cvals[i]), **kwargs )
```

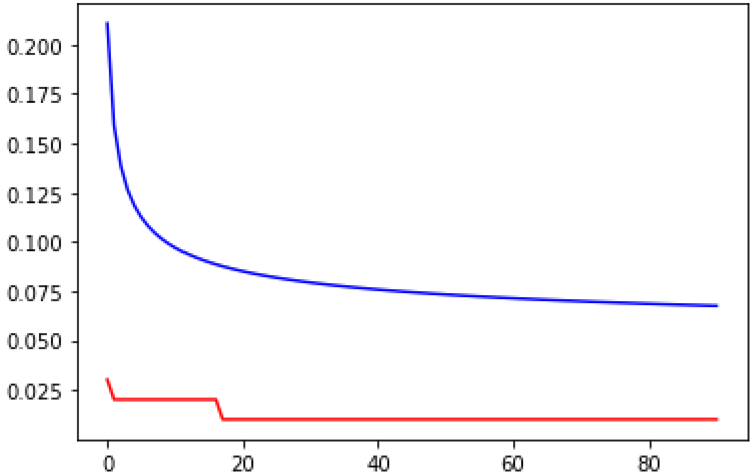


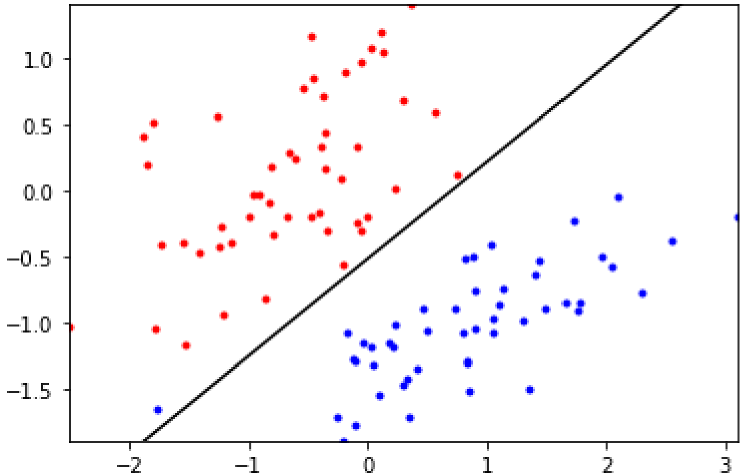
In [8]:
```
#Problem 5
# for theta 0, the answer is x0(-y(j)+sigma(x(j)(theta.T)))
# for theta 1, the answer is x1(-y(j)+sigma(x(j)(theta.T)))
# for theta 2, the answer is x2(-y(j)+sigma(x(j)(theta.T)))
```

In [9]:
```
#Problem 6
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 ones
    YY = ml.toIndex(Y,self.classes);   # YY is Y, but with canonical 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; # update stepsize
        # Do an SGD pass through the entire data set:
        for i in np.random.permutation(M):
            ri = 1.0 / (1.0 + np.exp(-XX[i].dot(self.theta))) # TODO: compute linear response r(x)
            gradi = -(1-ri)*XX[i,:] if YY[i] else 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)) ]
        Jsur = -np.mean(YY*np.log(1.0/(1.0+np.exp(-(XX.dot(self.theta)))))+(1-YY)*np.log(1-(1.0/(1.0+np.exp(-(XX.dot(s
        Jnll.append( Jsur ) # TODO evaluate the current NLL loss


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

In [10]:
```
# The stepsize I chose for problem 7 is 0.1. For the reason that when the stepsize is 0.1 the graph would be better.
```

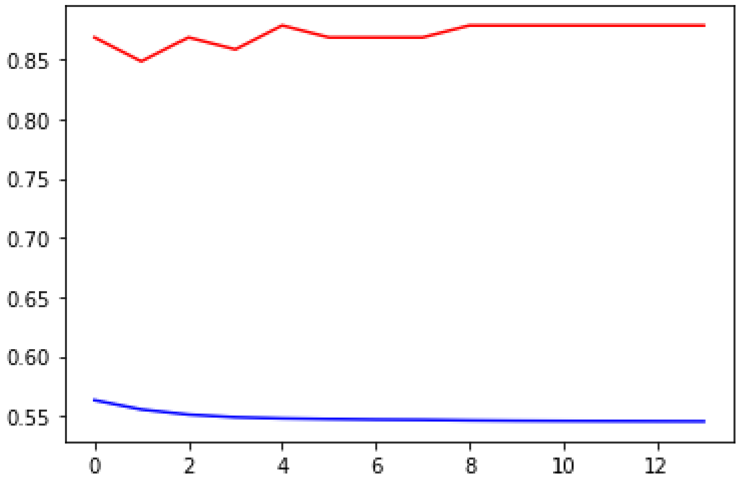
In [11]:
```
#Problem 7
learner.theta = np.array([0.,0.,0.]);
learner.train(XA,YA,initStep=1e-1,stopEpochs=5000,stopTol=1e-4);
print("Training error rate: ",learner.err(XA,YA))
```

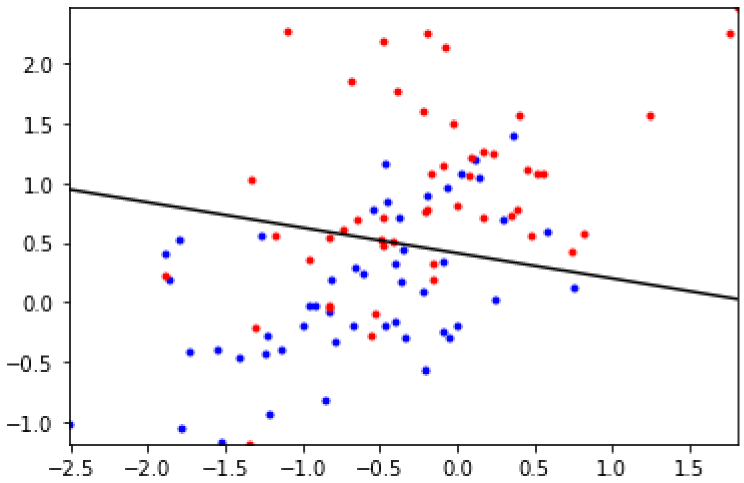




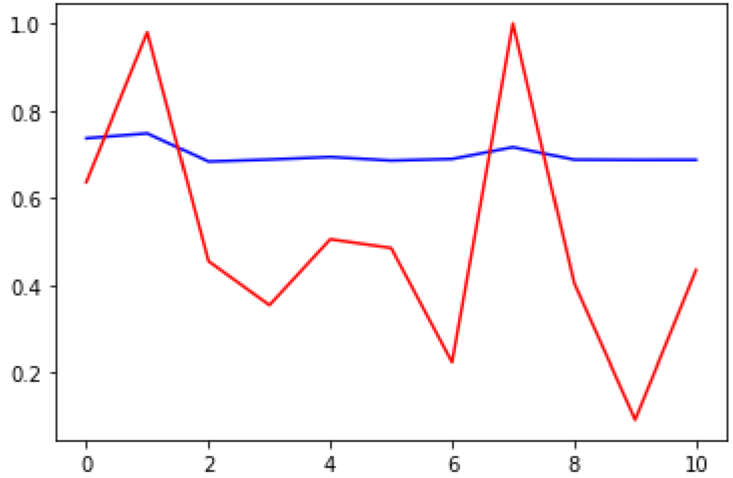

```
Training error rate:  0.010101010101010102
```

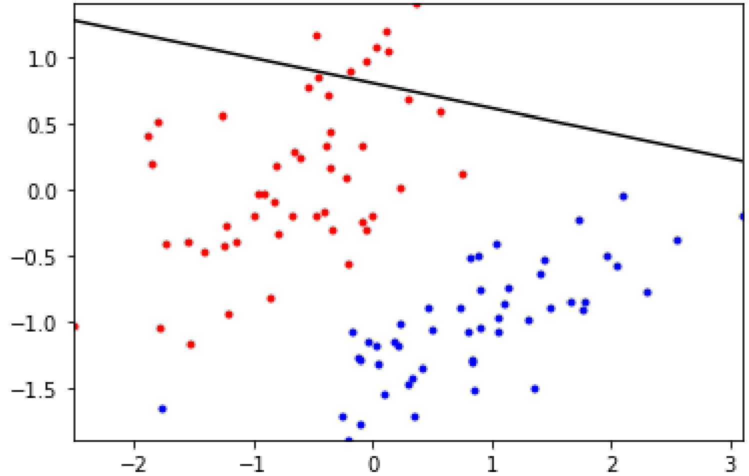
In [12]:
```
learner.theta = np.array([0.,0.,0.]);
learner.train(XB,YB,initStep=1e-1,stopEpochs=5000,stopTol=1e-4);
```



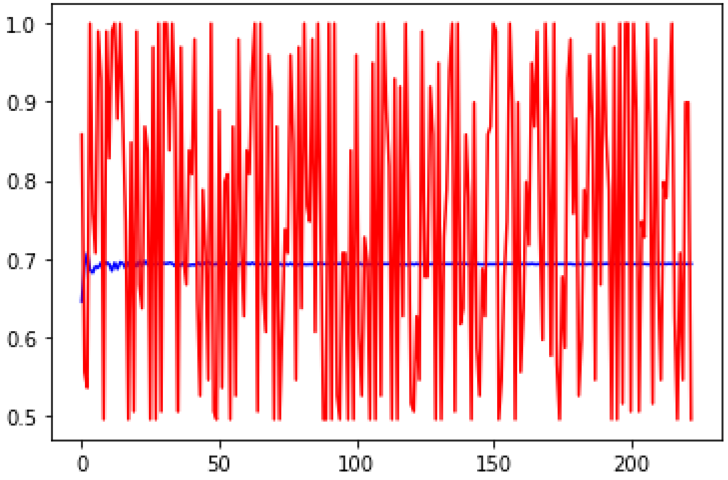


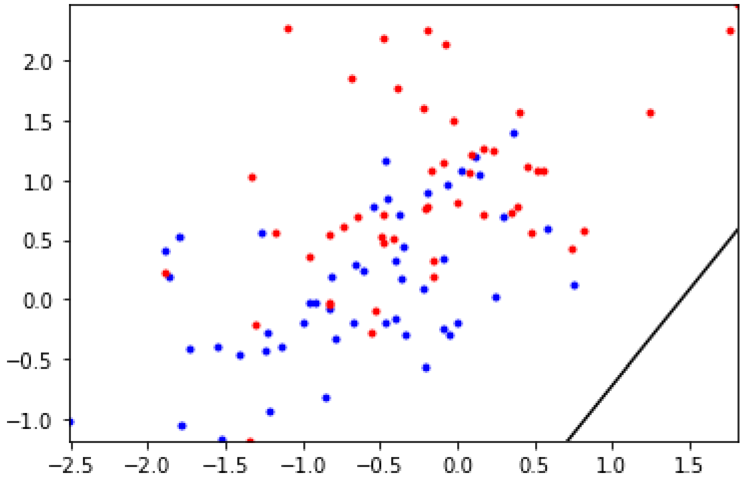
In [13]:
```
#Problem 8
learner.theta = np.array([0.,0.,0.]);
learner.l1train(XA,YA,initStep=.1,stopEpochs=5000,stopTol=1e-4,alpha=2.0)
```



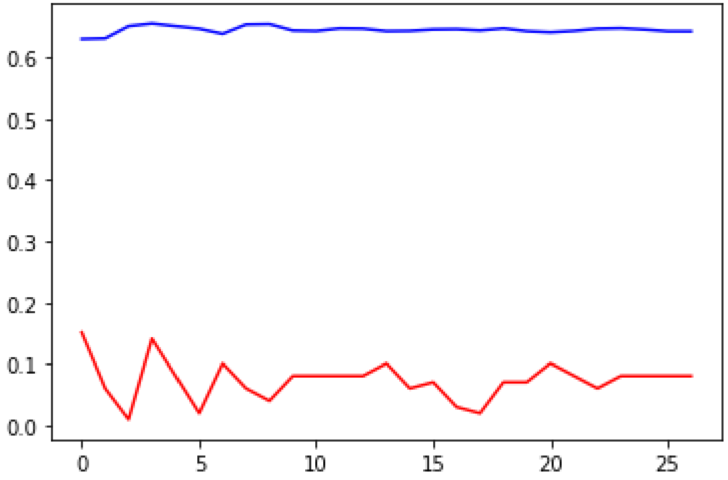


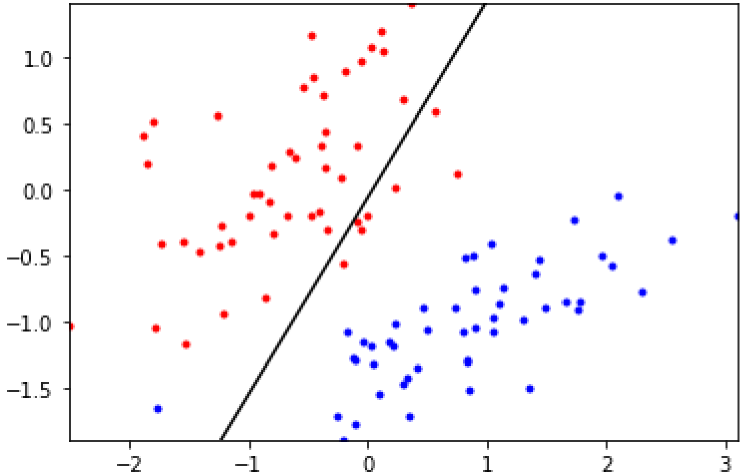
In [14]:
```
learner.theta = np.array([0.,0.,0.]);
learner.l1train(XB,YB,initStep=.1,stopEpochs=5000,stopTol=1e-4,alpha=2.0);
```



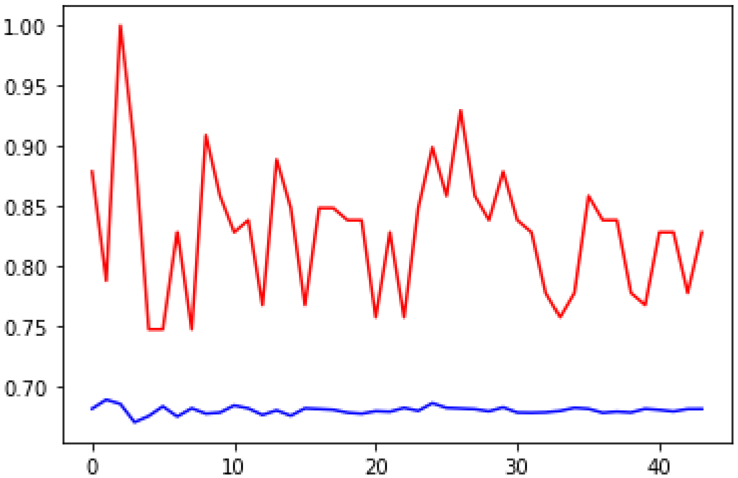


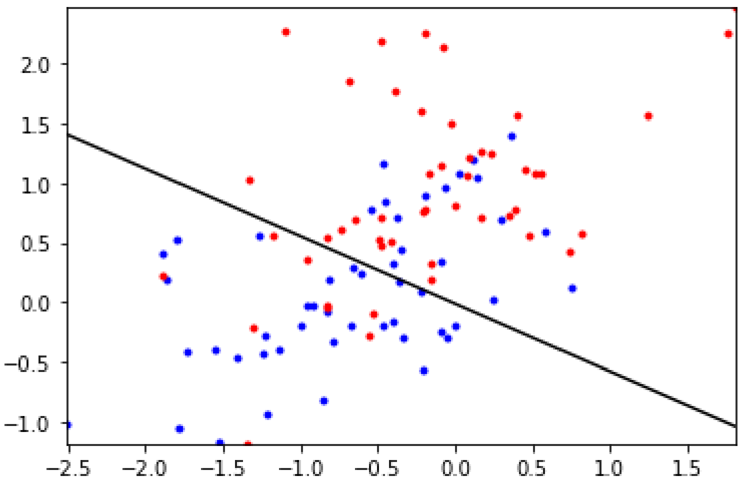
In [15]:
```
#Problem 9
learner.theta = np.array([0.,0.,0.]);
learner.l2train(XA,YA,initStep=1e-1,stopEpochs=5000,stopTol=1e-4,alpha=2.0);
```





In [16]:
```
learner.theta = np.array([0.,0.,0.])
learner.l2train(XB,YB,initStep=1e-1,stopEpochs=5000,stopTol=1e-4,alpha=2.0)
```





In [17]:
```
#problem 10
# I think we should pick L2 as the method here.
# the L1 graph has more outliers and it is not seperated well.
# The difference between them is that we can see that the graph of L1 has many fluctuations
# (because we used sign of theta to calculate based on L1).
# So the data would be up and down, that is the reason why we pick L2 as a better solution.
```

In [18]:
```
#Statement of Collaboration
#Yanghan-Deng
#Discussed with him about the stepsize choice problem.
```