CS 273P | Machine Learning

Homework 3

Problem 1

Data input

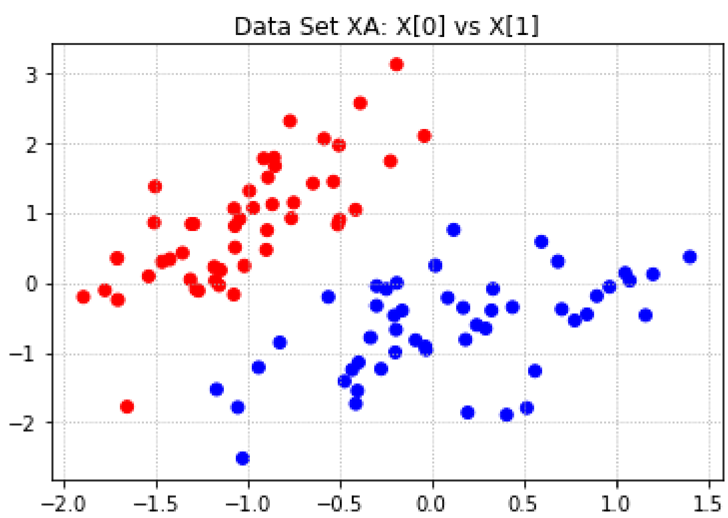
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
from logisticClassify2 import *
import warnings
warnings.filterwarnings('ignore')

#To ensure function can also draw using plt
%matplotlib inline
np.random.seed(0)
```

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 #YA-values with y<2(0,1)
XB, YB = X[Y>0,:], Y[Y>0] # get class 1 vs 2 #YB-values with y>0(1,2)
```

Problem 1 (a)

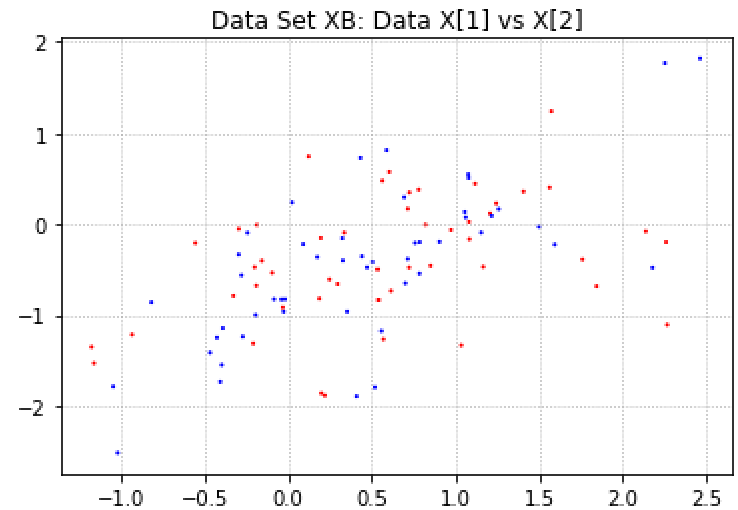
Scatter plots for XA and XB.

In [3]:
```
#Scatter plot for XA
col1=[]
for i in range(0,YA.size):
    if YA[i]==0:
        col1.append('r')
    elif YA[i]==1:
        col1.append('b')
    else:
        col1.append('g')
col2=[]
for i in range(0,YB.size):
    if YB[i]==0:
        col2.append('r')
    elif YB[i]==1:
        col2.append('b')
    else:
        col2.append('g')

plt.title('Data Set XA: X[0] vs X[1]')
plt.scatter(XA[:,0],XA[:,1],c=col1)
plt.grid(linestyle='dotted')
#ax=plt.axis()
plt.show()

print ('Linearly separable')
```




```
Linearly separable
```

In [4]:
```
#Scatter plot for XB
plt.scatter(XB[:,0],XB[:,1],YA==0,c='r')
plt.scatter(XB[:,0],XB[:,1],YA==1,c='b')
plt.title('Data Set XB: Data X[1] vs X[2]')
plt.grid(linestyle='dotted')
plt.show()
print('Not Linearly separable.')
```




```
Not Linearly separable.
```

Problem 1 (b)

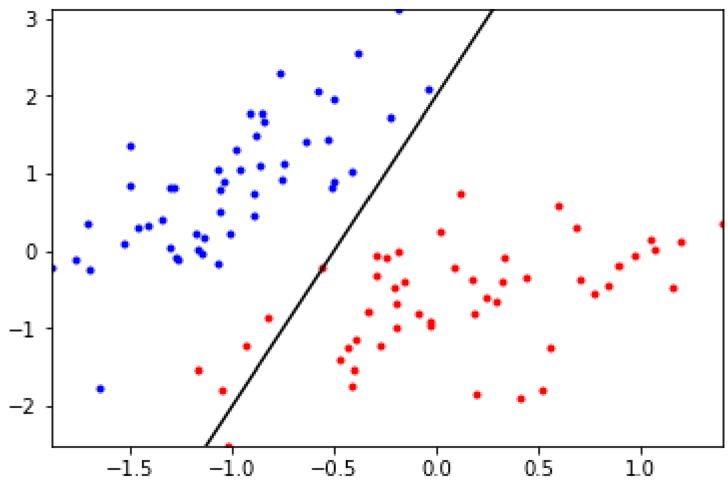
Definition for plotBoundary() function:

def plotBoundary(self,X,Y): """ Plot the (linear) decision boundary of the classifier, along with data """

```
    #print(len(self.theta))
    if len(self.theta) != 3: raise ValueError('Data & model must be 2D');
    ax = X.min(0),X.max(0); ax = (ax[0][0],ax[1][0],ax[0][1],ax[1][1]);
    ## find points on decision boundary defined by theta0 + theta1 X1 + thet
a2 X2 == 0
    x1b = np.array([ax[0],ax[1]]);  # at X1 = points in x1b
    x2b = -1*(self.theta[0]+x1b*self.theta[1])/self.theta[2];      # find x2
 values as a function of x1's values
    ## Now plot the data and the resulting boundary:
    A = Y==self.classes[0];                                              # a
nd 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();
```

Initializing learner for XA.

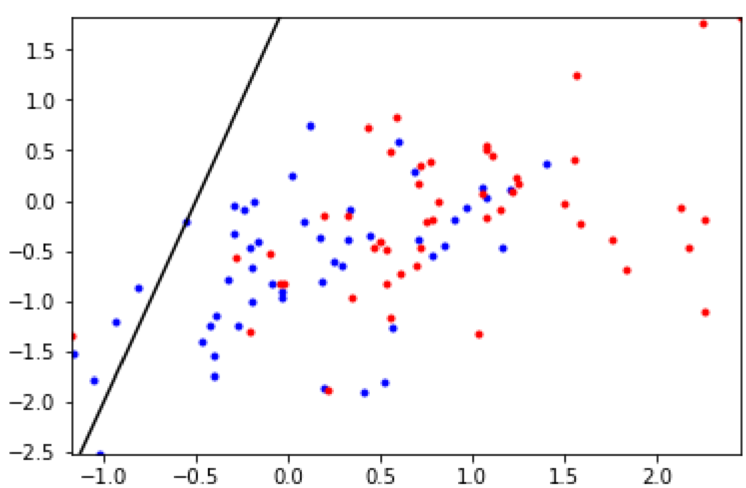
In [5]:
```
learner= logisticClassify2();
learner.classes=np.unique(YA)
wts=np.array([0.5,1,-0.25]);
learner.theta=wts;
learner.plotBoundary(XA,YA)
```



Classifier sign(.5 + 1x1 − .25x2) with theta values [0.5,1,-0.25] holds suitable for XA.

Initializing learner for XB and assigning initial weights

In [6]:
```
learnerb=logisticClassify2();
learnerb.classes=np.unique(YB)
wts=np.array([0.5,1,-0.25]);
learnerb.theta=wts;
learnerb.plotBoundary(XB,YB)
```



Classifier sign(.5 + 1x1 − .25x2) with theta values [0.5,1,-0.25] does not give a good estimate for XB.

Problem 1 (c):

Predict function completion:

In [7]:
```
YAhat=learner.predict(XA);
YBhat=learner.predict(XB);
print('Error Data set A: ',learner.err(XA,YA))
print('Error Data set B: ',learner.err(XB,YB));
```

```
Error Data set A:  0.050505050505050504
Error Data set B:  0.5454545454545454
```

Code for predict function:

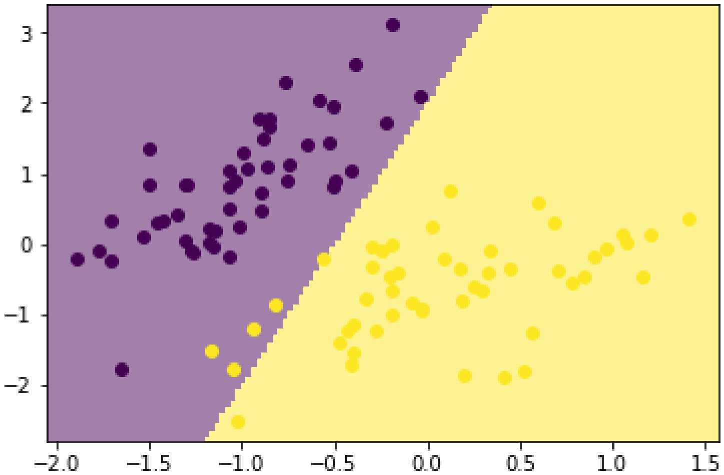
def predict(self, X): """ Return the predictied class of each data point in X"""

```
    ## compute linear response r[i] = theta0 + theta1 X[i,1] + theta2 X[i,2]
 + ... for each i
    ## if z[i] > 0, predict class 1:  Yhat[i] = self.classes[1]
    ##       else predict class 0:  Yhat[i] = self.classes[0]
    XX=np.c_[np.ones([X.shape[0],1]),X]
    r=XX.dot(self.theta);
    Yhat=np.zeros([X.shape[0],1])
    Yhat[r>0]=self.classes[1]
    Yhat[r<=0]=self.classes[0]
    return Yhat
```

Problem 1 (d)

Output predicted using given values YA

In [8]:
```
ml.plotClassify2D(learner,XA,YA)
```

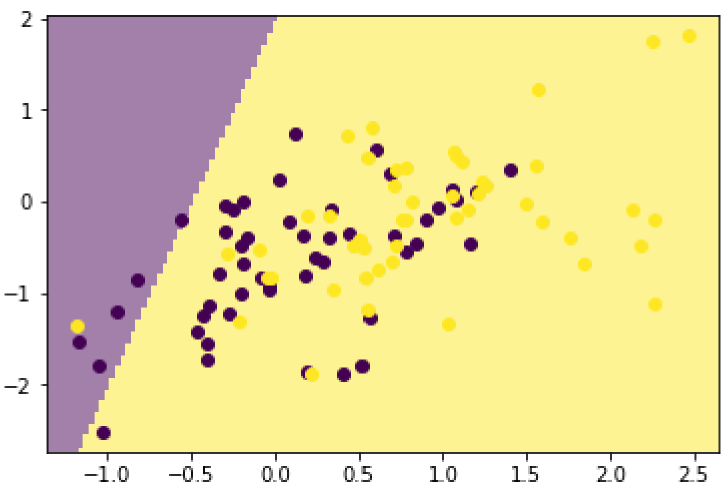


Output of predict code matches the one predicted by plotClassify2D.

Output of plotClassify2D for YA shows that the boundary separates the two data sets with reasonable accuracy. Thus,output predicted (using predict function) YAhat matches output YA.

Output predicted using given values YB

In [9]:
```
ml.plotClassify2D(learnerb,XB,YB)
```



Output of predict code somewhat matches the one predicted by plotClassify2D. Error is due to random/uneven distribution of data points.

Output of plotClassify2D for YB shows that the boundary separates the two data sets with some accuracy. Thus,output predicted (using predict function) YBhat matches output YB.

Problem 1 (e)

1e.jpg

Screen%20Shot%202018-05-13%20at%208.26.24%20PM.png

Problem 1 (f) :

Complete train() function

def train(self, X, Y, initStep=1., 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);

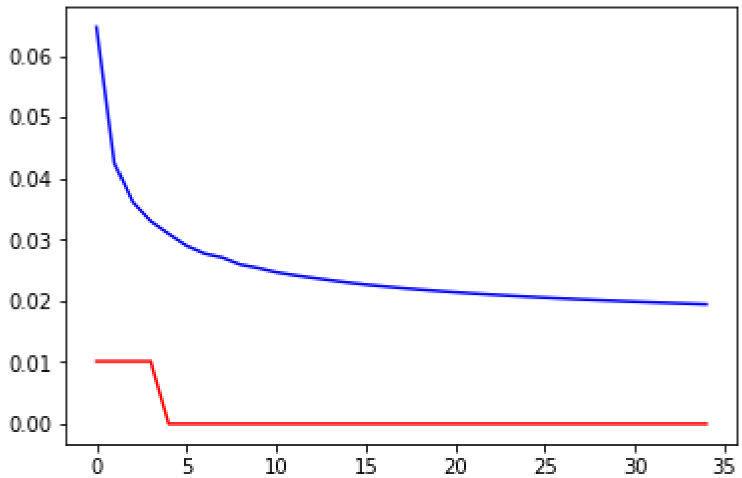
```
    # init loop variables:
    epoch=0; done=False; Jnll=[]; J01=[];
    while not done:
        stepsize, epoch = initStep*2.0/(2.0+epoch), epoch+1; # update stepsi
ze
        # Do an SGD pass through the entire data set:
        for i in np.random.permutation(M):
            ri    = XX[i].dot(self.theta)     # compute linear response r(x)
            si    = 1./(1.+np.exp(-ri))
            gradi = -(1-si)*XX[i,:] if YY[i] else si*XX[i,:];     #compute g
radient of NLL loss
            self.theta -= stepsize * gradi;  # take a gradient step

        J01.append( self.err(X,Y) )  # evaluate the current error rate
        ## compute surrogate loss (logistic negative log-likelihood)
        ##  Jsur = sum_i [ (log si) if yi==1 else (log(1-si)) ]
        S = 1./(1.+np.exp(-(XX.dot(self.theta))))
        Jsur = -np.mean(YY*np.log(S)+(1-YY)*np.log(1-S))
        Jnll.append(Jsur) # TODO evaluate the current NLL loss
        ## For debugging: you may want to print current parameters & losses
        # print self.theta, ' => ', Jsur[-1], ' / ', J01[-1]
        # raw_input()   # pause for keystroke
        # check stopping criteria: exit if exceeded # of epochs ( > stopEpoc
hs)
        # or if Jnll not changing between epochs ( < stopTol )
        done = epoch>=stopEpochs or (epoch>1 and abs(Jnll[-1]-Jnll[-2])< sto
pTol);
    plt.figure(1);plt.clf(); plt.plot(Jnll,'b-',J01,'r-'); plt.draw();    #
 plot losses
    if N==2: plt.figure(2);plt.clf(); self.plotBoundary(X,Y); plt.draw(); #
 & predictor if 2D
    plt.pause(.01);                    # let OS draw the plot
```

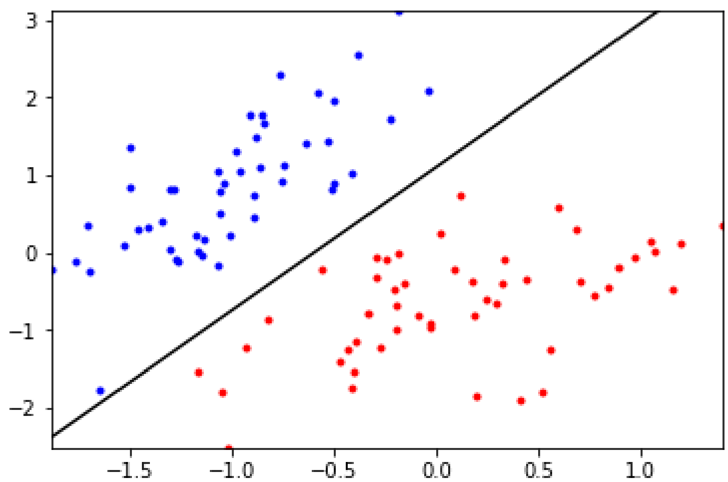
Problem 1 (g)

Data Set XA:

Parameters to train() function: train(X,Y,alpha=0,initStep=1.0,stopTol=1e-4,stopEpochs=500,plot=None):

In [10]:
```
#learner.train(XA, YA, initStep=1e-1,stopEpochs=1000,stopTol=1e-5);
learner.train(XA, YA, initStep=1.0,stopEpochs=500,stopTol=1e-4);
```

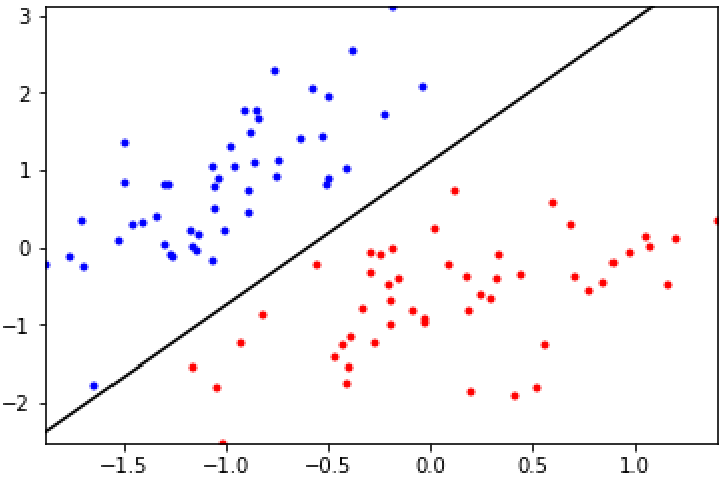




Alpha is 0,since it has no use here. If step size was not 1, we would observe many variations or no variation at all.

Final boundary for XA

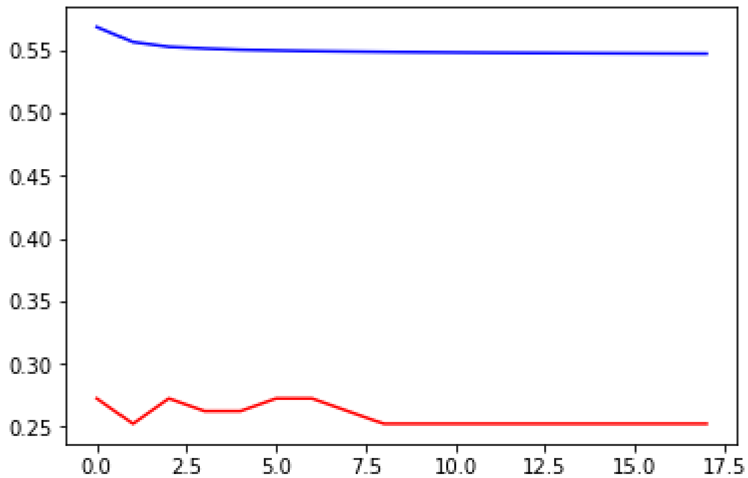
In [11]:
```
learner.plotBoundary(XA,YA)
```

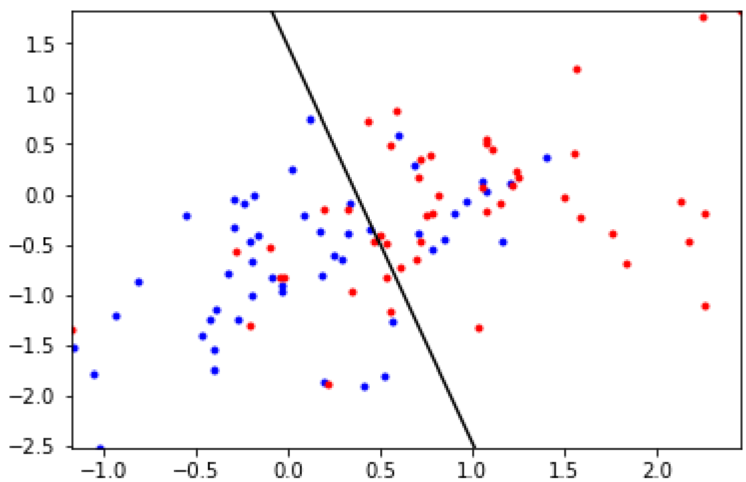


Data Set XB:

Parameters to train() function: train(X,Y,alpha=0,initStep=0.05,stopTol=1e-4,stopEpochs=200,plot=None):

In [12]:
```
#learnerb.train(XA, YA, initStep=1e-1,stopEpochs=1000,stopTol=1e-5);
learnerb.train(XB, YB, initStep=0.05,stopEpochs=200,stopTol=1e-4);
```

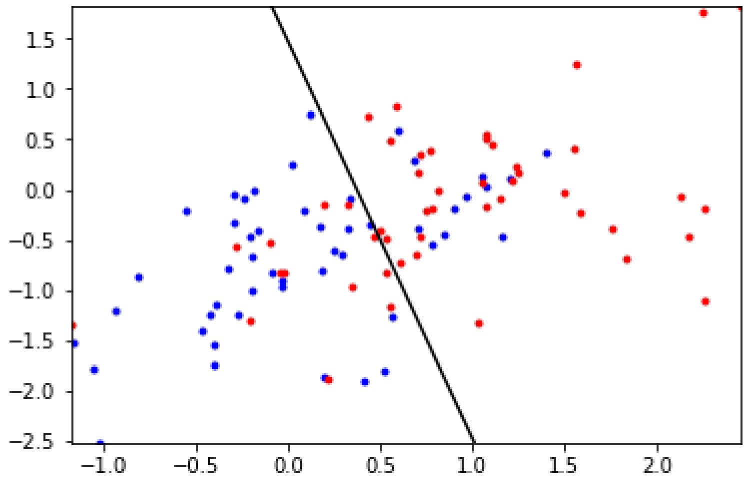




Alpha is 0,since it has no use here. If step size was not 0.05, we would observe many variations or no variation at all. Number of iterations bounded to 200, due to time constraints. Final boundary for XB:

Final boundary for XB

In [13]:
```
learnerb.plotBoundary(XB,YB)
```



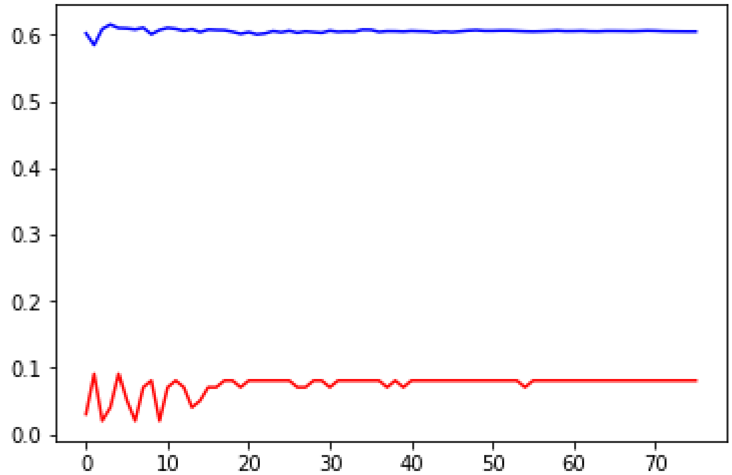
Problem 1 (h):

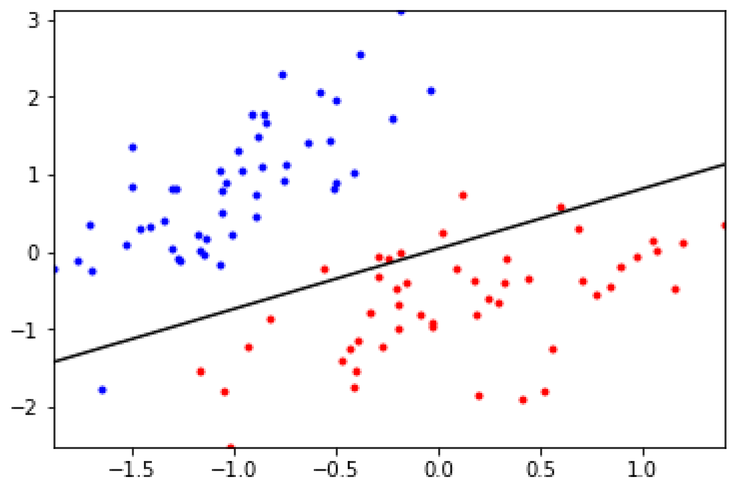
def myregtrain(self,X,Y, initStep=1.,stopTol=1e-4,stopEpochs=5000,alpha=0.,plot=None): """ Train the logistic regression using stochastic gradient descent """ M,N = X.shape; # initialize the model if necessary: self.classes = np.unique(Y); # Y may have two classes, any values XX = np.hstack((np.ones((M,1)),X)) # 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 stepsi
ze
        # Do an SGD pass through the entire data set:
        for i in np.random.permutation(M):
            ri = XX[i].dot(self.theta)     # compute linear response r(x)
            si    = 1./(1.+np.exp(-ri))
            gradi = -(1-si)*XX[i,:] if YY[i] else si*XX[i,:]  # compute grad
ient of NLL loss
            gradi += 2.*alpha*self.theta # gradient of the additional L2 reg
ularization term
            self.theta -= stepsize * gradi;  # take a gradient step
        J01.append( self.err(X,Y) )  # evaluate the current error rate
        ## compute surrogate loss (logistic negative log-likelihood)
        ##  Jnll = sum_i [ (log si) if yi==1 else (log(1-si)) ]
        S = 1./(1.+np.exp(-(XX.dot(self.theta))))
        Jsur = -np.mean(YY*np.log(S)+(1-YY)*np.log(1-S))
        Jnll.append( Jsur ) # evaluate the current NLL loss
        done = epoch>=stopEpochs or (epoch>1 and abs(Jnll[-1]-Jnll[-2])<stop
Tol);
                         # or if Jnll not changing between epochs ( < stopTo
l )
    plt.figure(1); plt.clf(); plt.plot(Jnll,'b-',J01,'r-'); plt.draw();
    # plot losses
    if N==2: plt.figure(2); plt.clf(); self.plotBoundary(X,Y); plt.draw();
    # & predictor if 2D
    plt.pause(.01);
```

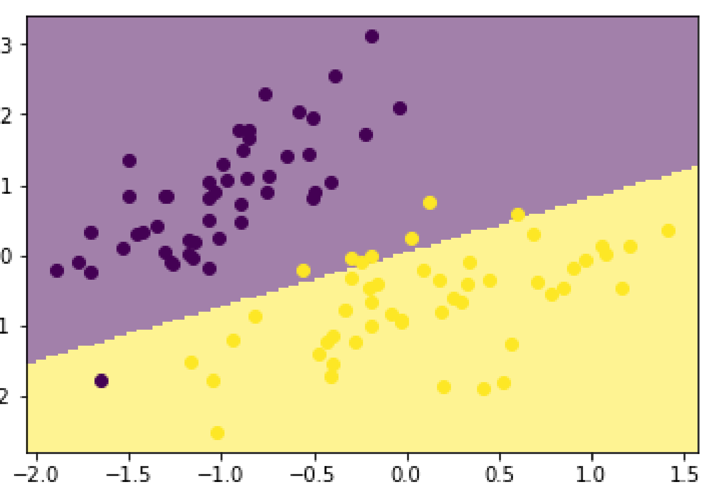
Learner A

In [14]:
```
learnerA = logisticClassify2();
wts=np.array([0.,0.,0.]);
learnerA.theta = wts
learnerA.myregtrain(XA, YA, initStep=1e-1,stopEpochs=1000,stopTol=1e-5,a
lpha=1.);
ml.plotClassify2D(learnerA,XA,YA)
print("Training error rate: ",learnerA.err(XA,YA))
```





Training error rate: 0.08080808080808081



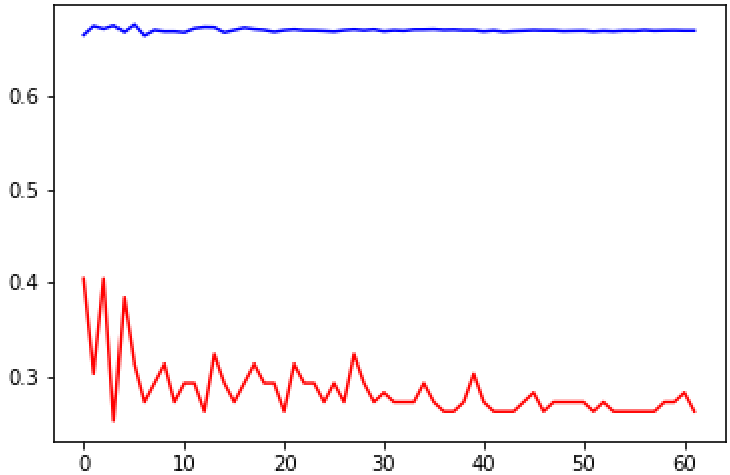
In [15]:
```
print(learner.theta,learnerA.theta)
```

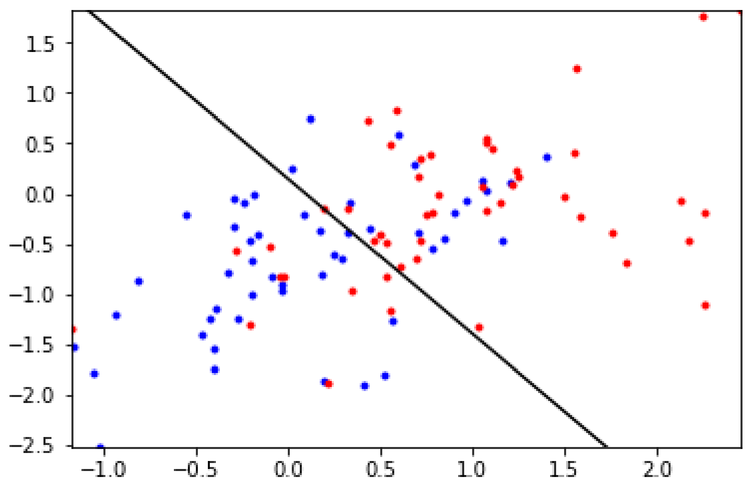
```
[ 4.39404387  7.33642527 -3.973653  ] [ 0.00639029  0.12477709 -0.16075
86 ]
```

We can see that value of theta have dropped considerably. Also, error has increased due to L2 regularization term.

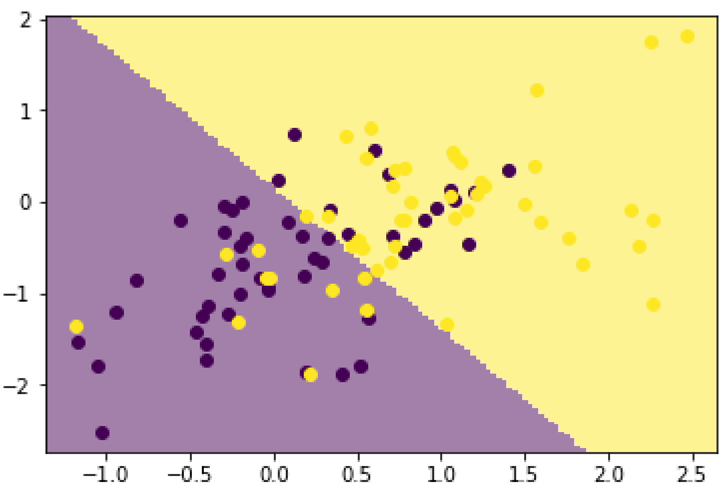
Learner B

In [16]:
```
learnerB2 = logisticClassify2();
wts=np.array([0.,0.,0.]);
learnerB2.theta = wts
learnerB2.myregtrain(XB, YB, initStep=1e-1,stopEpochs=1000,stopTol=1e-5,
alpha=1.);
ml.plotClassify2D(learnerB2,XB,YB)
print("Training error rate: ",learnerB2.err(XB,YB))
```





```
Training error rate:  0.26262626262626265
```



In [17]:
```
print(learnerb.theta,learnerB2.theta)
```

```
[-0.50786023  1.35789935  0.34555877] [-0.0084771   0.08792956  0.05711
623]
```

We can see that value of theta have dropped considerably. Also, error has increased due to L2 regularization term.

Problem 2:

2 a

Screen%20Shot%202018-05-13%20at%209.06.12%20PM.png

2 b

Screen%20Shot%202018-05-13%20at%209.06.21%20PM.png

2 c

Screen%20Shot%202018-05-13%20at%209.07.00%20PM.png

2 d

Screen%20Shot%202018-05-13%20at%209.08.36%20PM.png