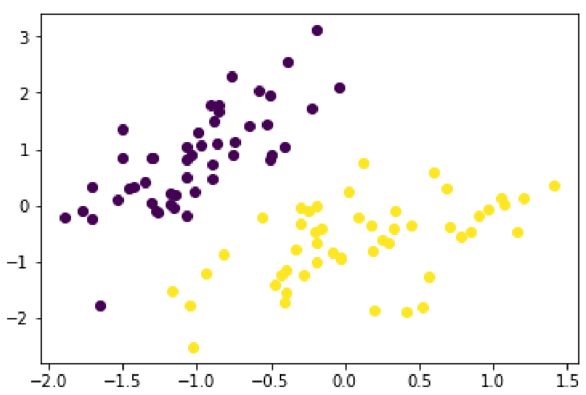
Problem 1: Logistic Regression

1.1:

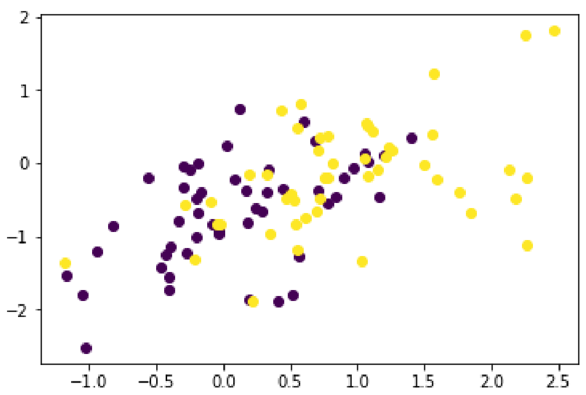
In [9]:

```
import numpy as np
import mltools as ml
import matplotlib.pyplot as plt

iris = np.genfromtxt("iris.txt",delimiter=None)
X, Y = iris[:,0:2], iris[:,-1]   # get first two features & target
X,Y  = ml.shuffleData(X,Y)       # reorder randomly rather than by class label
X,_  = ml.transforms.rescale(X)  # rescale to improve numerical stability, 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

print("Dataset A:")
ml.plotClassify2D(None, XA, YA)
plt.show()
print("Dataset B:")
ml.plotClassify2D(None, XB, YB)
```

```
Dataset A:
```




```
Dataset B:
```



The first dataset is much more linearly separable because there is a gap between the different colored points.

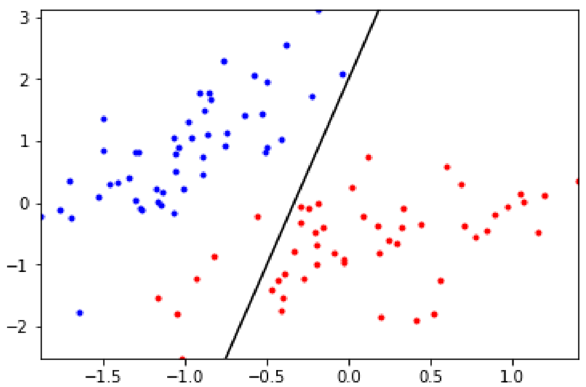
1.2:

In [11]:

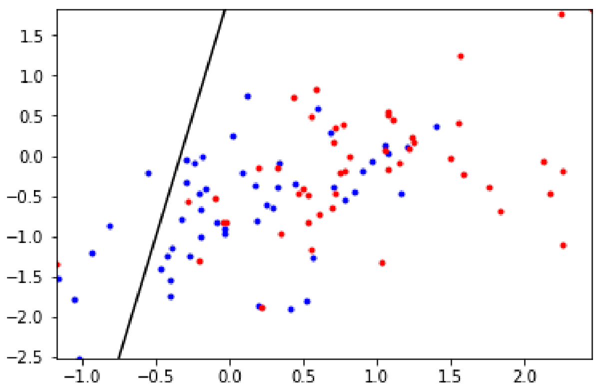
```
import mltools as ml
from logisticClassify2 import *
```

```
def myPlotBoundary(self, X,Y):
    """ Plot the (linear) decision boundary of the classifier, along with data """
    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]);
    ## TODO: find points on decision boundary defined by theta0 + theta1 X1 + theta2 X2 == 0
    x1b = np.array([ax[0],ax[1]]);  # at X1 = points in x1b
    x2b = np.array([(-self.theta[1]*x1b[0]-self.theta[0])/self.theta[2],(-self.theta[1]*x1b[1]-self
.theta[0])/self.theta[2]])      # TODO find x2 values as a function of x1's values
    ## 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();


print("Decision Boundary of Dataset A:")
learner = logisticClassify2()
learner.classes = np.unique(YA)       # store the class values for this problem
wts = np.array([2,6,-1]); # TODO: fill in values
learner.theta = wts;
myPlotBoundary(learner,XA,YA)
plt.show()
print("Decision Boundary of Dataset B:")
learners = logisticClassify2()
learners.classes = np.unique(YB)       # store the class values for this problem
wts = np.array([2,6,-1]); # TODO: fill in values
learners.theta = wts;
myPlotBoundary(learners,XB,YB)
```

```
Decision Boundary of Dataset A:
```



```
Decision Boundary of Dataset B:
```



1.3:

In [13]:

```
# Should go in your logistic2 class:
def myPredict(self,X):
    """ Return the predictied class of each data point in X"""
    Yhat = []
    a = 0
    for x in X:
```

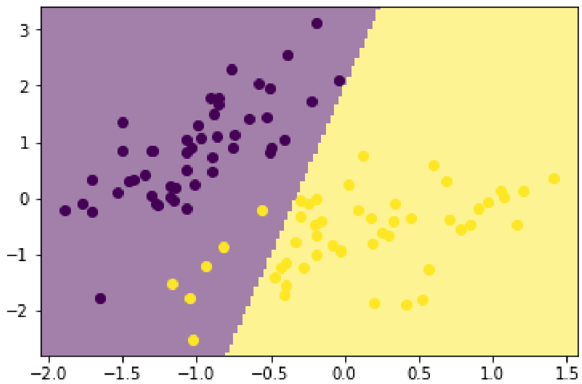
```
    for v in X:
        o = self.theta[2]*X[a,1] + self.theta[1]*X[a,0] + self.theta[0]
        if o > 0:
            Yhat.append(self.classes[1])
        else:
            Yhat.append(self.classes[0])
        a += 1
    ## TODO: compute linear response r[i] = theta0 + theta1 X[i,1] + theta2 X[i,2]  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]
    return np.array(Yhat)


# Update our shell classifier definition
class logisticClassify2(ml.classifier):
    classes = []
    theta = np.array( [-1, 0, 0] )   # initialize theta to something
    plotBoundary = myPlotBoundary    #
    predict = myPredict
    train = None

learnerA = logisticClassify2()
learnerA.classes = np.unique(YA)       # store the class values for this problem
learnerA.theta = np.array([2,6,-1]);   # TODO: insert hard-coded values
print("Dataset A: ")
print(learnerA.err(XA, YA))
learnerB = logisticClassify2()
learnerB.classes = np.unique(YB)       # store the class values for this problem
learnerB.theta = np.array([2,6,-1]);
print("Dataset B: ")
print(learnerB.err(XB, YB))
```

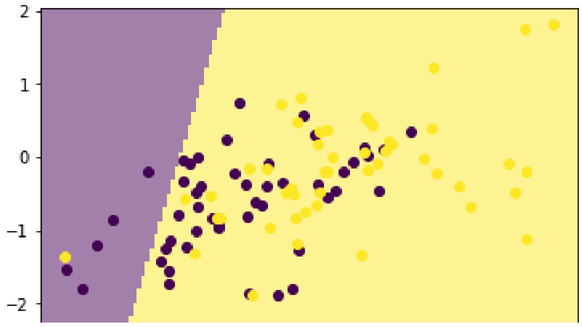
```
Dataset A:
0.06060606060606061
Dataset B:
0.45454545454545453
```

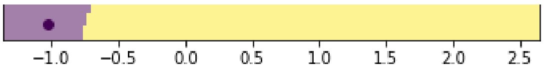
1.4:

In [14]:

```
ml.plotClassify2D(learnerA, XA, YA)
plt.show()
ml.plotClassify2D(learnerB, XB, YB)
plt.show()
```







1.5:

Jj(θ)=−y^(j)logσ(x^(j)·θ) − (1−y^(j))log(1−σ(x^(j)·θ)) = -y^(j) *(1/σ(x^(j)* θ) *σ(x^(j)·θ)* (1 - σ(x^(j)·θ)) *x^(j) - (1-y(j))* (1/(1 -σ(x^(j) *θ)) -* σ(x^(j)·θ) *(1 - σ(x^(j)·θ))* x^(j) Cancel out similar variables = -y^(j) *x^(j)* (1 - σ(x^(j)θ)) - x^(j) *(1-y(j))* -σ(x^(j)* θ)) = x^(j)(-y^(j) + σy^(j) (x^(j)θ)) + σ(x^(j)θ)) - σy^(j)(x^(j)θ))) = (σ(x^(j)θ) - y^(j)x^(j)

1.6:

In [25]:

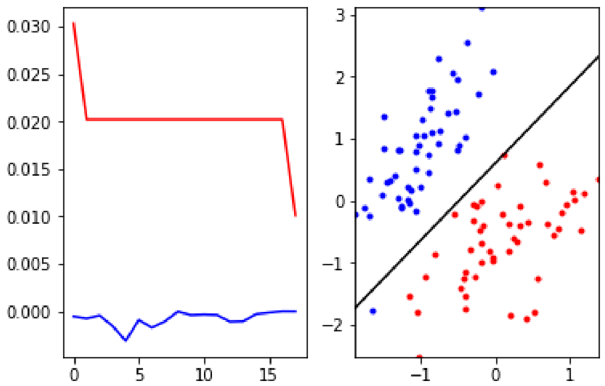
```
def sigmoid(x):
        return 1. / (1 + np.exp(-x))

def myTrain(self, X, Y, initStep=1.0, stopTol=1e-4, stopEpochs=5000, plot=None):
    """ Train the logistic regression using stochastic gradient descent """
    from IPython import display
    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     = self.theta[2] * X[i, 1] + self.theta[1] * X[i, 0] + self.theta [0]      # TODO: c
mpute linear response r(x)
            gradilist = [ sigmoid(ri) - YY[i], X[i,0] * (sigmoid(ri) - YY[i]), X[i,1] * (sigmoid(ri)
- YY[i])]

            gradi = np.array(gradilist)      # 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)) ]
        if (YY[i] == 1):
            Jsur = np.sum(np.log(sigmoid(ri)))
        else:
            Jsur = np.sum(np.log(1 - sigmoid(ri)))
        d = Jsur/M
        Jnll.append(d) # TODO evaluate the current NLL loss
        display.clear_output(wait=True);
        plt.subplot(1,2,1); plt.cla(); plt.plot(Jnll,'b-',J01,'r-'); # plot losses
        if N==2: plt.subplot(1,2,2); plt.cla(); self.plotBoundary(X,Y); # & 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 epochs ( > stopEpochs)
        if (epoch > stopEpochs):
            done = True
        if (len(Jnll) > 1):
            if np.abs(Jnll[-2] - Jnll[-1]) < stopTol:
                done = True
           # or if Jnll not changing between epochs ( < stopTol )
```

1.7:

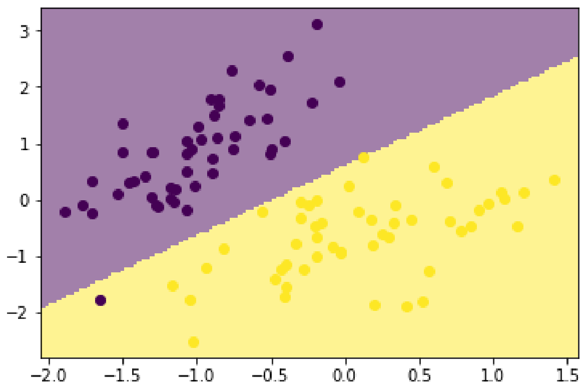
In [26]:

```
class logisticClassify2(ml.classifier):
    classes = []
```

```
    theta = np.array( [-1, 0, 0] )    # initialize theta to something
    plotBoundary = myPlotBoundary     #
    predict = myPredict               # Now all parts are implemented
    train = myTrain

learnerA = logisticClassify2()
learnerA.theta = np.array([0.,0.,0.]);
learnerA.train(XA,YA,initStep=1e-1,stopEpochs=1000,stopTol=1e-5);
ml.plotClassify2D(learnerA,XA,YA)
print("Training error rate: ",learnerA.err(XA,YA))
plt.show()
```

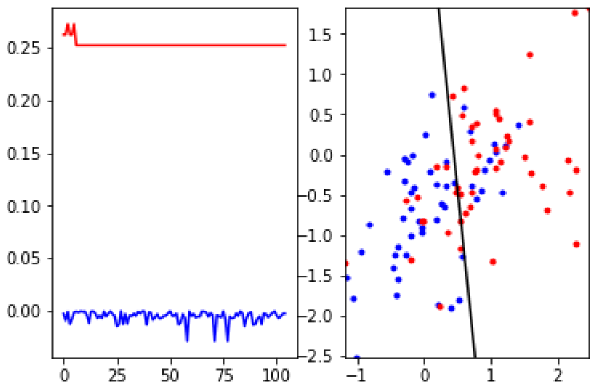


('Training error rate: ', 0.010101010101010102)

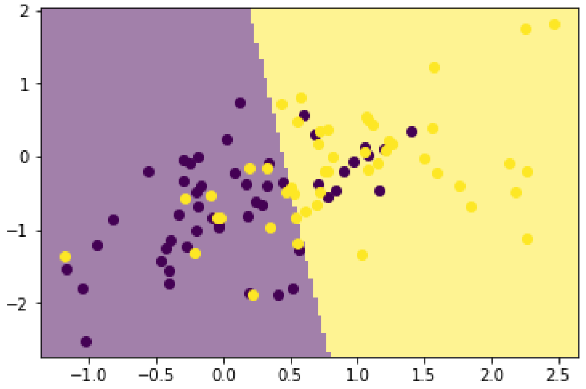


In [27]:

```
learnerB = logisticClassify2()
learnerB.theta = np.array([0.,0.,0.]);
learnerB.train(XB,YB,initStep=1e-1,stopEpochs=1000,stopTol=1e-5);
ml.plotClassify2D(learnerB,XB,YB)
print("Training error rate: ",learnerB.err(XB,YB))
plt.show()
```



('Training error rate: ', 0.25252525252525254)



Problem 2:

1.T(a + bx1): A and B can be shattered because the points can be separted into +1 and -1. C and D cannot be because there are arrangements for the dots being the same or different that don't allow them to be split into +1 and -1.

2.T((a∗b)x1+(c/a)x2): This learner is similar to the first in the sense that it is a line. It can separate A and B but can't separate C and D. A line has to be able to split the data in +1 and -1 and for C and D, points that are similar can end up on the same side which means they can't be shattered.

3.T((x1−a)2+(x2−b)2+c): A, B, and C can be shattered because a circle can separate the data points into ones that are similar and different. D can't be shattered because if the last 3 points are the same and the first is different, then a circle won't effectively separate the data.

Problem 3: Statement of Collaboration

I asked a few questions on Piazza and looked at the others' questions and responses on Piazza. I also asked a friend who's good at math to explain gradient descent to me and she did.