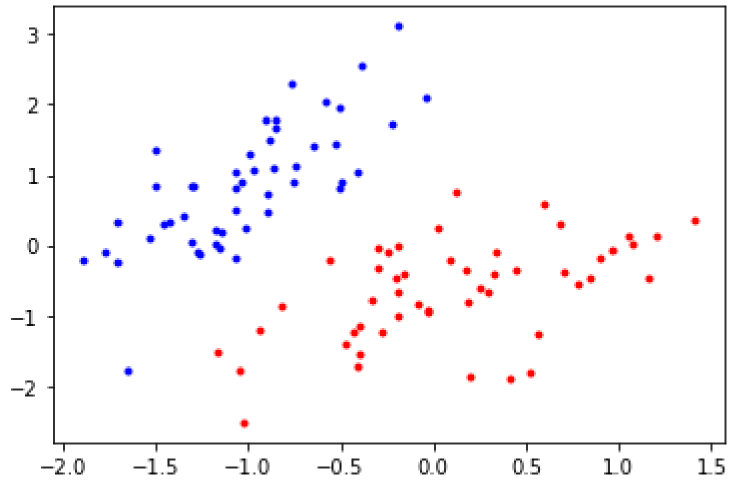
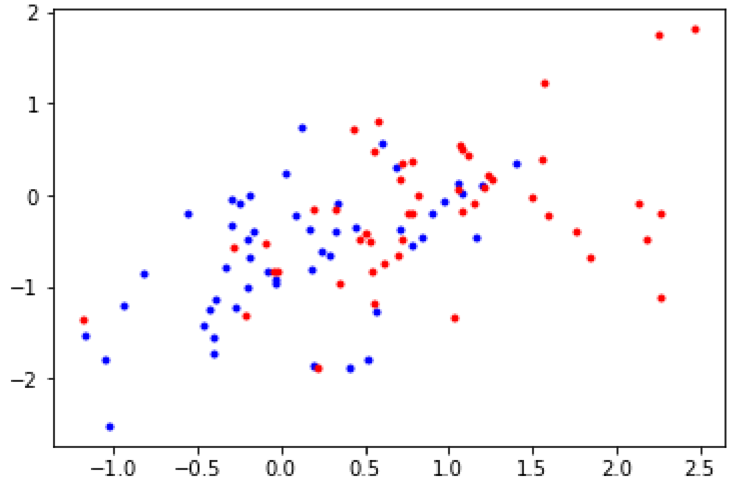
1a.

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] # get first two features & target
X,Y = ml.shuffleData(X,Y) # order randomly rather than by class label
X,_ = ml.rescale(X) # rescale to improve numerical stability, speed converg

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
```

In [2]:
```
plt.plot(X[Y==0,0], X[Y==0,1], 'b.', X[Y==1,0], X[Y == 1,1], 'r.');
```



In [3]:
```
plt.plot(X[Y==1,0], X[Y==1,1], 'b.', X[Y==2,0], X[Y == 2,1], 'r.');
```



The first data set is linearly differentiable

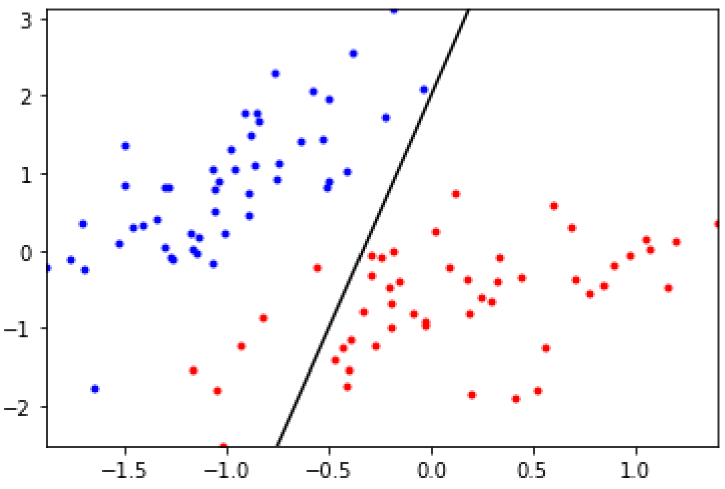
1b.

In [4]:
```
def plotBoundary(self,X,Y):
    #print(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]);
    ## TODO: find points on decision boundary defined by theta0 + theta1 X1
    x1b = np.array([ax[0], ax[1]]);  # at X1 = points in x1b
    x2min = -(self.theta[0] + self.theta[1] * ax[0])/self.theta[2]
    x2max = -(self.theta[0] + self.theta[1] * ax[1])/self.theta[2]
    x2b = np.array([x2min, x2max])
    # 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.axi
```

In [5]:
```
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([2,6,-1]); # TODO: fill in values
learner.theta = wts; # set the learner's parameters
```

In [7]:
```
learner.plotBoundary(XA,YA)
```

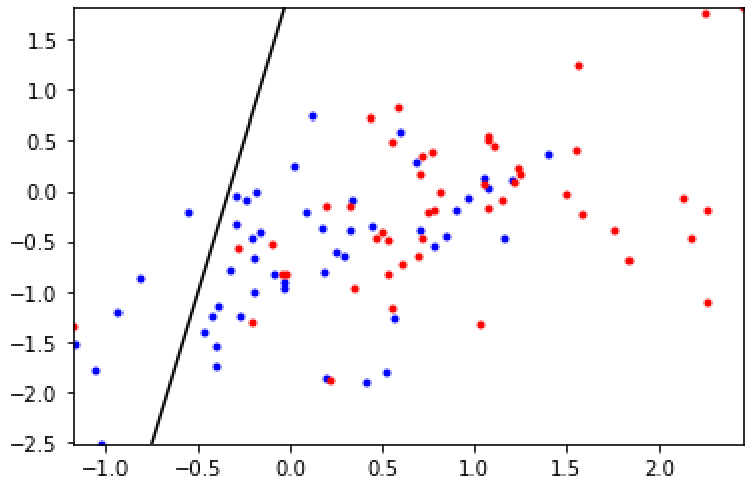
```
[-1.88809931  1.40400472]
[-1.88809931 -2.51410964] [1.40400472 3.11521318]
-1.8880993063510247 1.4040047204812869 -2.5141096425672345 3.115213177801
313
```



In [8]:
```
import mltools as ml
from logisticClassify2 import *

learner = logisticClassify2(); # create "blank" learner
learner.classes = np.unique(YB); # define class labels using YA or YB
wts = np.array([2,6,-1]); # TODO: fill in values
learner.theta = wts; # set the learner's parameters
```

In [9]:
```
learner.plotBoundary(XB,YB)
```

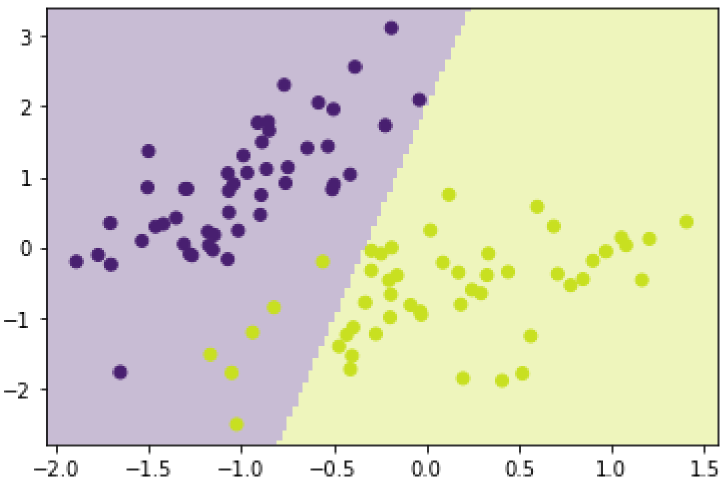
```
[-1.18047423  2.46307253]
[-1.18047423 -2.51410964] [2.46307253 1.81554298]
-1.1804742313591114 2.4630725284309425 -2.5141096425672345 1.815542978610
9256
```



1c and 1d.

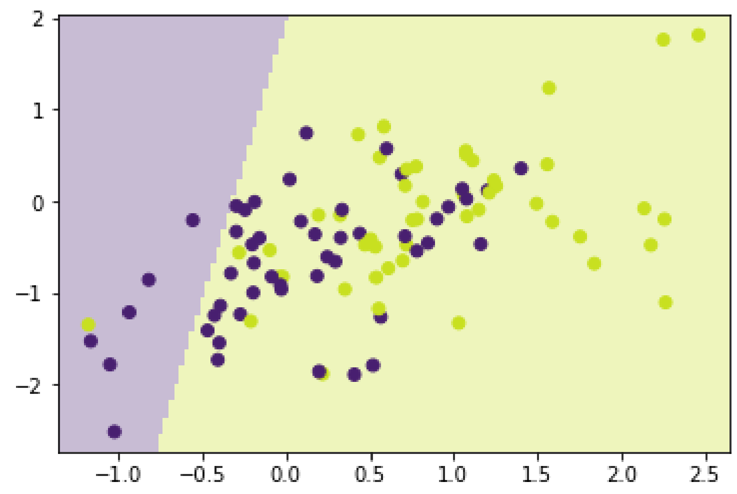
In [10]:
```
import numpy as np
def predict(X, classes):
    """ Return the predictied class of each data point in X"""
    #raise NotImplementedError
    theta = [2,6,-1]
    ## TODO: compute linear response r[i] = theta0 + theta1 X[i,1] + theta2
    Yhat = []
    for i in range(len(X)):
        r = theta[0] + theta[1] * X[i,0] + theta[2] * X[i,1]
        if(r > 0):
            Yhat.append(float(classes[1]))
        else:
            Yhat.append(float(classes[0]))
    ## TODO: if z[i] > 0, predict class 1:  Yhat[i] = self.classes[1]
    ##       else predict class 0:  Yhat[i] = self.classes[0]
    return Yhat
```

In [11]:
```
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(np.array([2,6,-1])); # TODO: fill in values
learner.theta = wts; # set the learner's parameters

ml.plotClassify2D(learner,XA,YA)
plt.show()

a = learner.predict(XA)
err = 0

for x in range(len(YA)):
    if(a[x] != YA[x]):
        err += 1
print("Estimated error:", err/len(YA))

print("Actual error:", learner.err(XA,YA))
```



```
Estimated error: 0.06060606060606061
Actual error: 0.06060606060606061
```

In [12]:
```
import mltools as ml
from logisticClassify2 import *

learner = logisticClassify2(); # create "blank" learner
learner.classes = np.unique(YB); # define class labels using YA or YB
wts = np.array([2,6,-1]); # TODO: fill in values
learner.theta = wts; # set the learner's parameters

ml.plotClassify2D(learner,XB,YB)
plt.show()

b = learner.predict(XB)
err = 0

for x in range(len(YB)):
    if(b[x] != YB[x]):
        err += 1

print("Estimated error:", err/len(YB))

print("Actual error:", learner.err(XB,YB))
```



```
Estimated error: 0.45454545454545453
Actual error: 0.45454545454545453
```

1e.

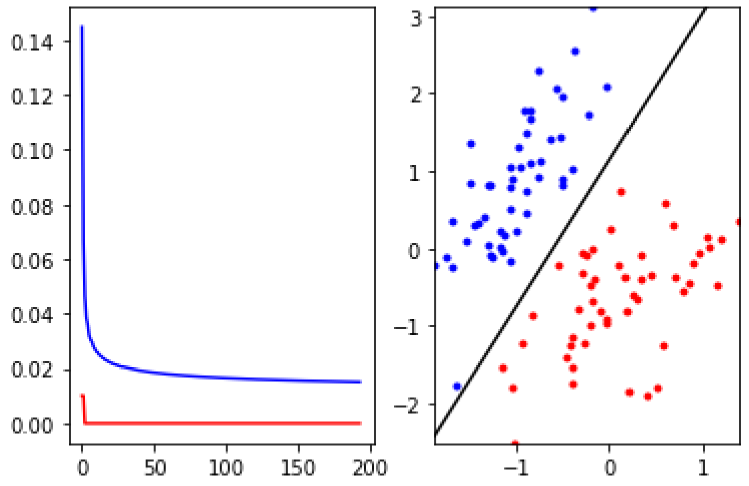
```
Jj(θ) = -y(j) * log(σ(x(j)· θ)) - (1 - y(j)) * log(1 - (σ(x(j)· θ))

If y(j) = 1
Jj(θ) = -log(σ(x(j)· θ))
∇J(θ) = -((1/(σ(x(j)· θ)) * (σ(x(j)· θ) * (1 - σ(x(j)· θ) * (x(j)))
∇J(θ) = -((1 - σ(x(j)· θ) * (x(j)))
∇J(θ) = -x(j) + (x(j) * σ(x(j)· θ))
∇J(θ) = (σ(x(j)· θ) - 1) * x(j)
∇J(θ) = (σ(x(j)· θ) - y(j)) * x(j)

If j = 0
Jj(θ) = − log(1 - (σ(x(j)· θ)))
∇J(θ) = -((1/(1 - (σ(x(j)· θ))) * (-(σ(x(j)· θ)) * (1 - (σ(x(j)· θ)) *
(x(j)))
∇J(θ) = -((-σ(x(j)· θ) * (x(j)))
∇J(θ) = -(x(j) * (-σ(x(j)· θ)))
∇J(θ) = (σ(x(j)· θ)) * x(j)
∇J(θ) = (σ(x(j)· θ) - 0) * x(j)
∇J(θ) = (σ(x(j)· θ) - y(j)) * x(j)
```

```
1f.
```

```
In [ ]: def train(self, X, Y, initStep=1.0, stopTol=1e-4, stopEpochs=5000, plot=Non
                """ Train the logistic regression using stochastic gradient descent
                from IPython import display
                import math
                M,N = X.shape;                     # initialize the model if necess
                self.classes = np.unique(Y);       # Y may have two classes, any va
                XX = np.hstack((np.ones((M,1)),X)) # XX is X, but with an extra col
                YY = ml.toIndex(Y,self.classes);   # YY is Y, but with canonical va
                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 s
                    # Do an SGD pass through the entire data set:
                    s = [0] * M
                    for i in np.random.permutation(M):
                        ri = self.theta[0] * XX[i,0] + self.theta[1] * XX[i,1] + se
                        si = pow((1.0 + math.exp(-ri)), -1)
                        s[i] = si
                        a = []
                        for x in range(len(XX[i])):
                            a.append((si - YY[i]) * XX[i][x])
                        for x in range(len(self.theta)):
                            self.theta[x] -= (stepsize * a[x])
                        # take a gradient step
                    #print("b")
                    J01.append( self.err(X,Y) )  # evaluate the current error rate

                    ## TODO: compute surrogate loss (logistic negative log-likeliho
                    ## Jsur = - sum_i [ (log si) if yi==1 else (log(1-si)) ]
                    Jsur = []
                    for x in range(len(s)):
                        if(YY[x] == 1):
                            Jsur.append(np.log(s[x]))
                        else:
                            Jsur.append(np.log(1 - s[x]))
                    Jsur = -1 * sum(Jsur)
                    #Jnll.append(-np.log((1 - s[x])))# TODO evaluate the current NL
                    Jnll.append(Jsur/M)

                    display.clear_output(wait=True);   # clear display if using jup
                    plt.subplot(1,2,1); plt.cla(); plt.plot(Jnll,'b-',J01,'r-');
                    if N==2: plt.subplot(1,2,2); plt.cla(); self.plotBoundary(X,Y);
                    plt.pause(.01);                    # let OS draw the plot

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

                    # TODO check stopping criteria: exit if exceeded # of epochs (
                    if (len(Jnll) >= 2):
                        if((epoch > stopEpochs) or (abs(Jnll[-1] - Jnll[-2]) < stop
                            done = True
                    else:
                        if(epoch > stopEpochs):
                            done = True
```

In [18]:
```
learnerA = logisticClassify2()
learnerA.theta = np.array([0.,0.,0.]);
learnerA.train(XA,YA,initStep=1.0,stopEpochs=1000,stopTol=1e-5);
```

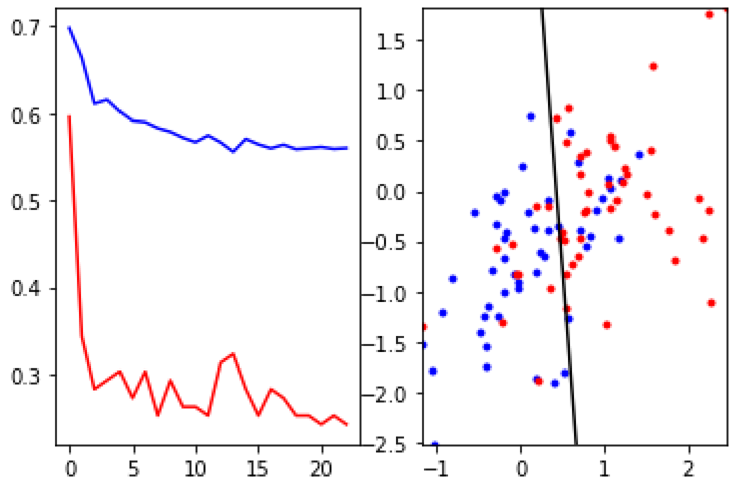
```
[-1.88809931  1.40400472]
[-1.88809931 -2.51410964] [1.40400472 3.11521318]
-1.8880993063510247 1.4040047204812869 -2.5141096425672345 3.115213177801
313
```



For the first data set, I chose a relatively large initial step since I knew that a large step would improve intial negative log likelihood loss significantly. I chose a relatively small stop condition for stopTol and an ample number of Epochs to make sure the linear classifier linearly diffentiated the two class types. The large initial step led to a quick find of the linear classifier.

In [20]:
```
learnerA = logisticClassify2()
learnerA.theta = np.array([0.,0.,0.]);
learnerA.train(XB,YB,initStep=1.0,stopEpochs=1000,stopTol=1e-3);
```

```
[-1.18047423  2.46307253]
[-1.18047423 -2.51410964] [2.46307253 1.81554298]
-1.1804742313591114 2.4630725284309425 -2.5141096425672345 1.815542978610
9256
```



For the second data set, I chose the same relatively large initial step, which improved the error rate significantly, but I chose a larger stopTol because the second data set is extremely muddled and not linearly differentiable, so running too many Epochs would not help the classifier much and at some point, the classifier would be stuck in a dead end and never be satisfactory.

2a. T(a + bx1) is a horizontal line. It can split one point by going over or under. If the 2 points in figure b are of the same class, it can go under or over both points. If the 2 points are two different class, it can go in between them. For figure c, if all 3 points are of one class, the line can go under or over. However if points (2,2) and (4,8) are one class and (6,4) is another class there is no horizontal split in which (2,2) and (4,8) can be on the same side without (6,4) also being on that side since the y value of (6,4) is between (2,2) and (4,8)

The VC dimension of this classifier is 2.

2b. T((a * b)x1 + (c/a)x2) is a line with slope and offset, so it can obviously split one point by going on either side of it. It can split two points, or leave two points on one side, so it can shatter two points. It can also shatter 3 points easily with the case of all points being the same class, and the 3 points being split into 2 of one class and 1 of another, since a line can split 3 points into one and two in any configuration. It cannot shatter 4 points. Assume points opposite each other are chosen to be one class, and the other two opposite points which would cross a line made by the first two points were chosen to be another class. You can place the line so 1 pair of same class points are on one side, but the two points of the other class will inevitably be split since the path between them crosses the path between the other points.

The VC dimension of this classifier is 3.

2c. T((x1 - a)^2 + (x2 - b)^2 + c).
The shape of this equation is an circle, whose center can be anywhere and whose size is variable. It can shatter one point by placing the point inside or outside of the circle by changing the location or size. It can also shatter two points by leaving two points inside or outside the circle. Additionally, it can change the location or size of circle to anywhere so that one point is inside the circle and one is outside. For three points of one class, the circle can be somewhere else so that no points are inside the circle, or the circle can change location and change to place all points inside the circle. Three points can be split into one point inside and two outside by making the circle have a small radius and centering it at the point. For two points inside the circle, the circle can be made large enough to acommodate two points at the edge of the circle, but I can't tell for sure whether the points in figure c are sufficiently far enough so that if I need a circle of a large radius to accommodate two points, the third point would be far enough from the arc of the circle to avoid being in it. However, it seems like the points are distinct enough, so I will say that the classifier can shatter the points in figure c. Figure d cannot be shattered by this classifier since if you choose the points at x = 2, x = 4, and x = 8 to be one class, and the point at x = 6 to be another, the circle will be extremely large, and the point at x = 6 will not be distinct or far enough from all the point to not be included in the circle.

The VC dimension of this classifier is 3.

Statement of Collaboration: Discussed the structure of ax in problem 1.2 with Luke Yi.

4. I carved a jack-o-lantern and studied for a midterm on Halloween.