CS178 Homework 3 Bryan Oliande 13179240

February 8, 2017

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import mltools as ml
        from logisticClassify2 import *
        %pylab inline
        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.transforms.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
```

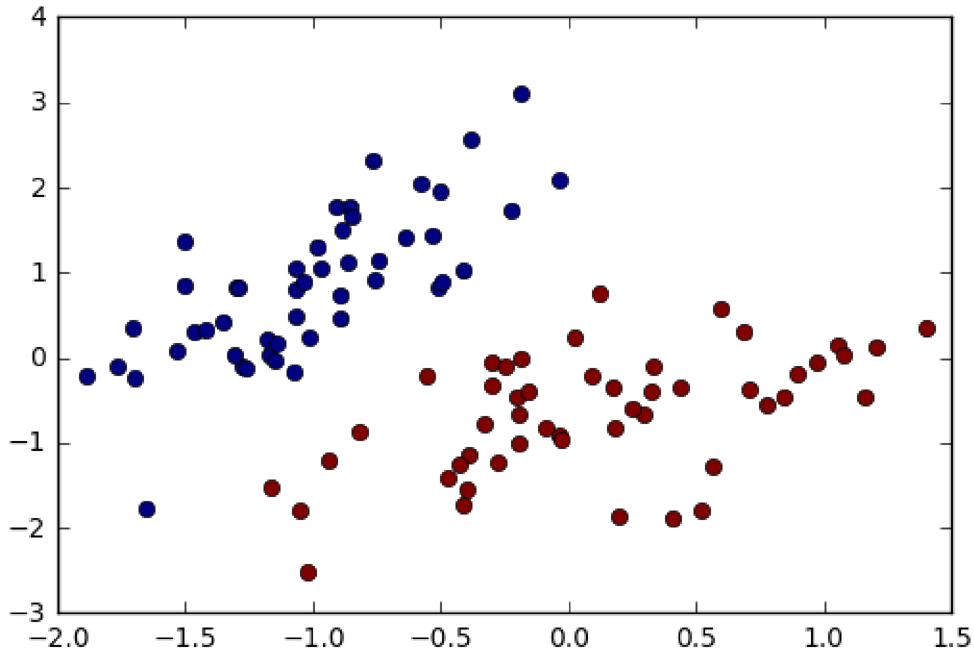
```
Populating the interactive namespace from numpy and matplotlib
```

Problem 1: Perceptrons and Logistic Regression:

(a) Show the two classes in a scatter plot (one for each data set) and verify that one data set is linearly separable and that the other is not.

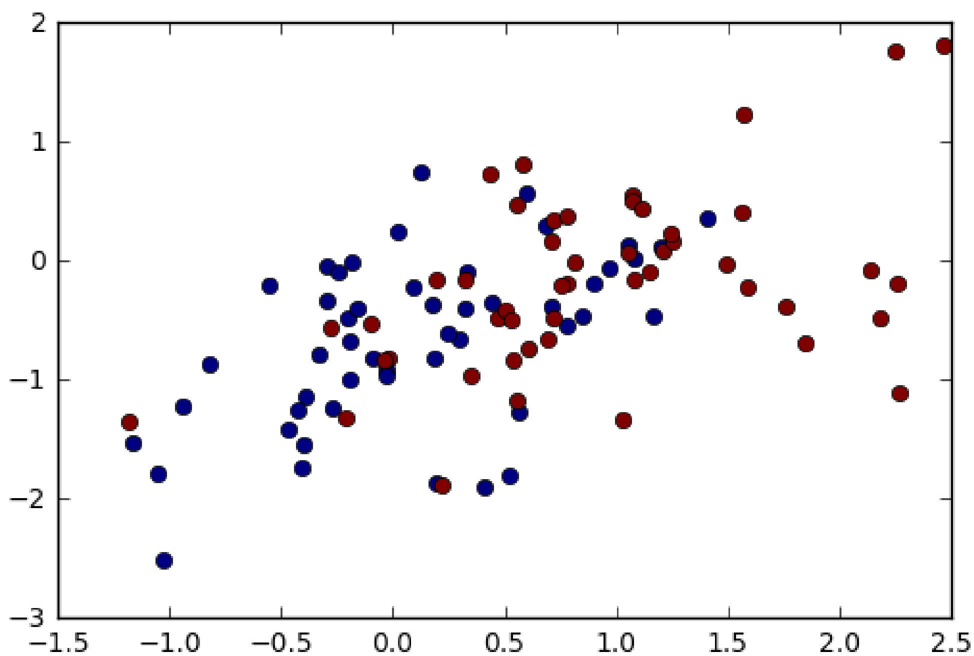
Data Set A:

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



Data Set B:

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

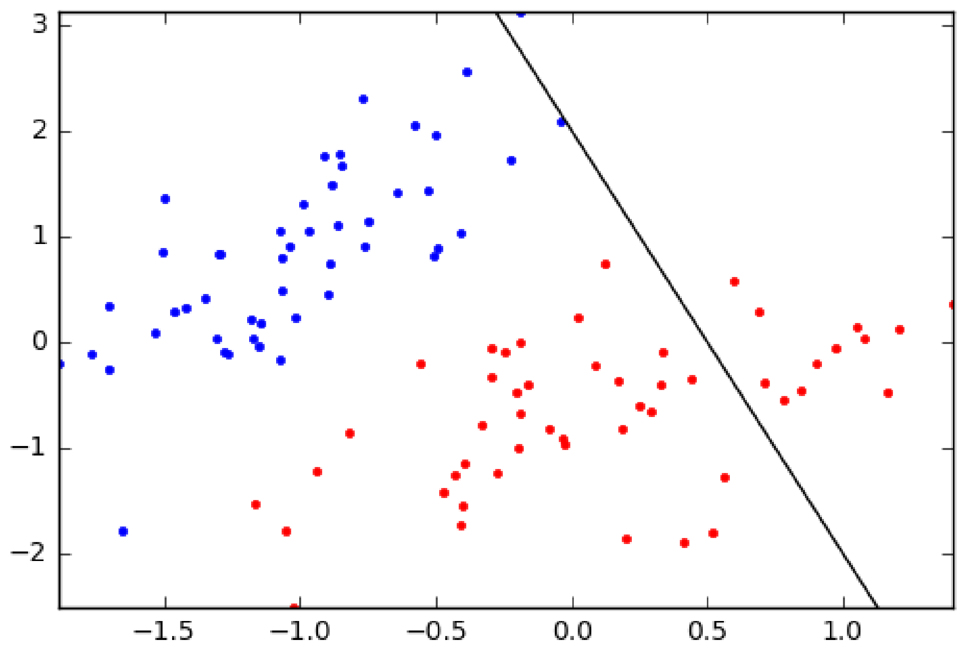


(b) I filled in the function plotBoundary() in logisticClassify2.py as such:

def plotBoundary(self,X,Y): “”" Plot the (linear) decision boundary of the classifier, along with data “”" ax = X.min(0),X.max(0); ax = (ax[0][0],ax[1][0],ax[0][1],ax[1][1]); x1b = np.array([ax[0],ax[1]]); # at X1 = points in x1b
x2b = ((x1b * self.theta[1]) - (self.theta[0])) / self.theta[2] A = Y == self.classes[0] self.classes = np.unique(Y) plt.plot(X[A,0],X[A,1],‘b.’,X[-A,0],X[-A,1],‘r.’,x1b,x2b,‘k-’); plt.axis(ax); plt.draw();

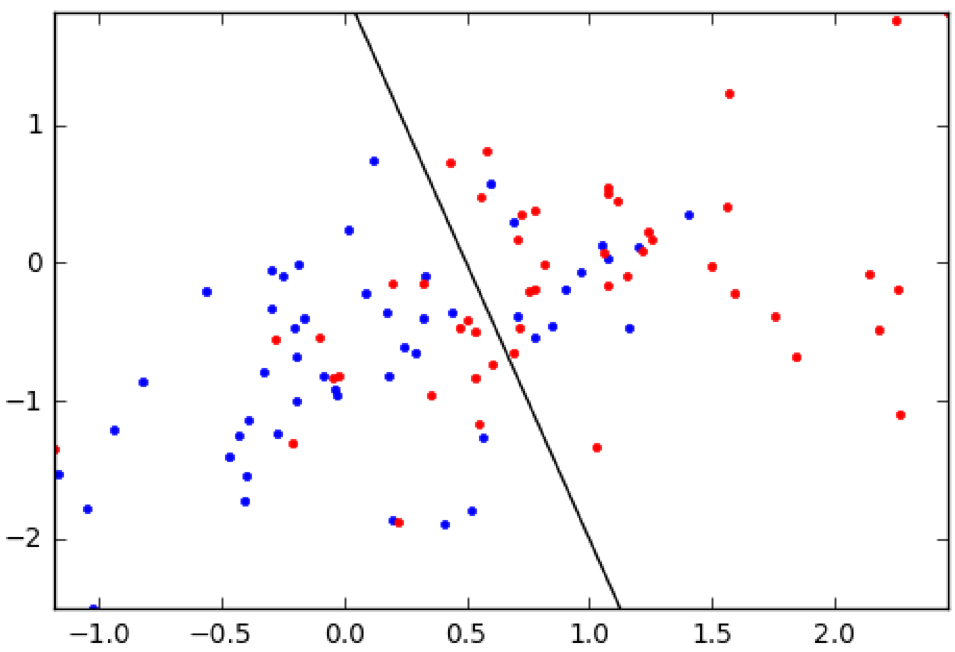
Plotting the A data with plotBoundary():

```
In [4]: learner = logisticClassify2(); #create a new learner
        learner.classes = np.unique(YA) #[0 1]
        wts = np.array([0.5,1,-0.25]);
        learner.theta = wts #setting the parameters
        learner.plotBoundary(XA, YA)
```



Plotting the B data with plotBoundary():

```
In [5]: learner = logisticClassify2(); #create a new learner
        learner.classes = np.unique(YB) #[1 2]
        wts = np.array([0.5,1,-0.25]);
        learner.theta = wts #setting the parameters
        learner.plotBoundary(XB, YB)
```



(c)Complete the logisticClassify2.predict function. Compute and report the error rate of the classifier in the previous part on both data sets A and B.

```
def predict(self, X):
    """ Return the predictied class of each data point in X"""
    num_rows, num_cols = X.shape
    r = np.zeros(shape=(num_rows))
    Yhat = np.zeros(shape=(num_rows))

    for i in range(0, num_rows):
        r[i] = self.theta[0] + self.theta[1]*X[i,0] + self.theta[2]*X[i,1]
        if r[i] > 0:
            Yhat[i] = self.classes[1]
        else:
            Yhat[i] = self.classes[0]

    return Yhat
```

```
In [6]: learner = logisticClassify2(); #create a new learner
        learner.classes = np.unique(YA) #[0 1]
        wts = np.array([0.5,1,-0.25]);
        learner.theta = wts #setting the parameters
        YhatA = learner.predict(XA)

        learner = logisticClassify2(); #create a new learner
        learner.classes = np.unique(YB) #[1 2]
        wts = np.array([0.5,1,-0.25]);
```

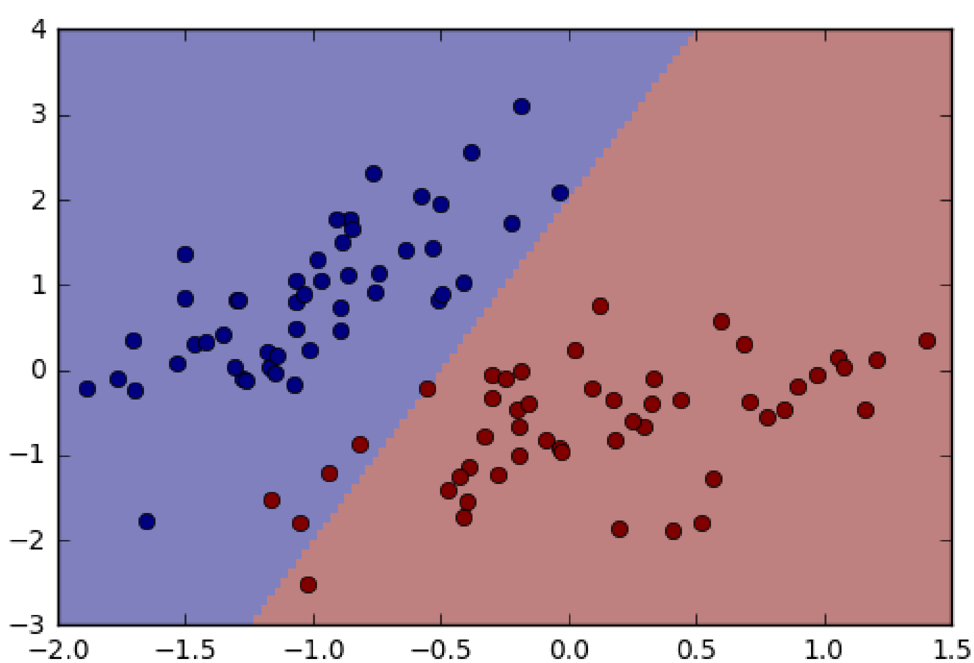
```
        learner.theta = wts #setting the parameters
        YhatB = learner.predict(XB)
```

```
In [7]:  errorA = np.mean( YhatA != YA )
         errorB = np.mean( YhatB != YB )

        print('Error A: ' + str(errorA) + '\n' )
        print('Error B: ' + str(errorB) + '\n' )
```

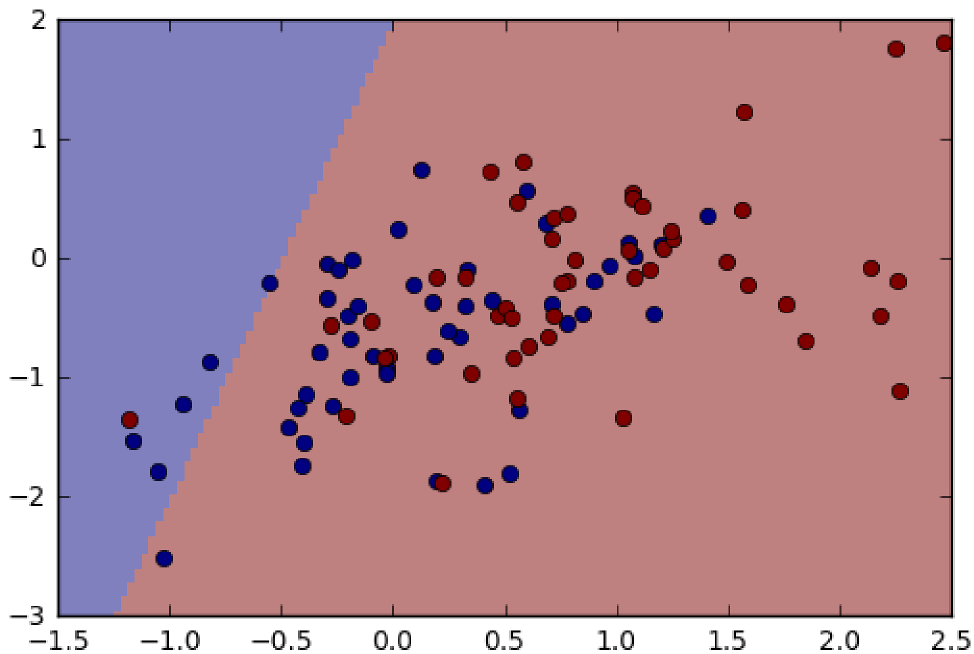
```
Error A: 0.0505050505051

Error B: 0.464646464646
```

(d) Verify that your predict code matches your boundary plot by using plotClassify2D with your manually constructed earner on the two data sets.

```
In [8]: ###FOR SET A:

        learner = logisticClassify2(); #create a new learner
        learner.classes = np.unique(YA) #[0 1]
        wts = np.array([0.5,1,-0.25]);
        learner.theta = wts #setting the parameters
        ml.plot.plotClassify2D(learner, XA, YA)
```



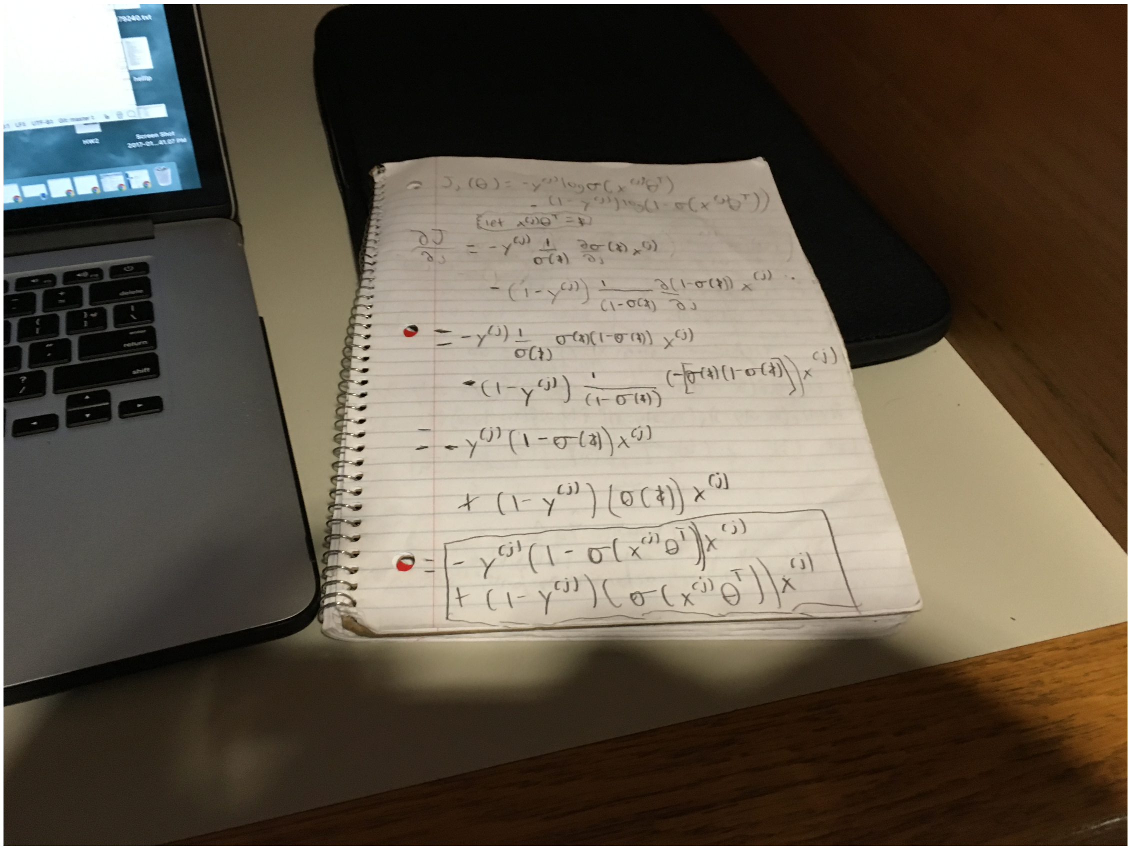
```
In [9]: ###FOR SET B:

        learner = logisticClassify2(); #create a new learner
        learner.classes = np.unique(YB) #[1 2]
        wts = np.array([0.5,1,-0.25]);
        learner.theta = wts #setting the parameters
        ml.plot.plotClassify2D(learner, XB, YB)
```



(e)Here is what I derived for the gradient of J
IMG_0154.JPG

```
In [10]: from IPython.display import Image
         Image(filename='IMG_0156.JPG')
```

```
Out[10]:
```



(f)I implemented the train() function in logisticClassify2.py as such:

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 value

```
XX = np.hstack((np.ones((M,1)),X))
YY = ml.toIndex(Y,self.classes);   # YY is Y, but with canonical values 0 or
r = [None] * M
if len(self.theta)!=N+1: self.theta=np.random.rand(N+1);
epoch=0; done=False; Jnll=[]; J01=[];

while not done:
    stepsize = initStep*2.0/(2.0+epoch)
    epoch = epoch+1;
    JSur = 0;
    for i in range(0,M):
        r[i] =  self.theta[0]*XX[i][0] + self.theta[1]*XX[i][1] + self.theta[2]
        if self.classes[1] == 2: #had to add this for part B
            if Y[i] == 2:
                Y[i] = 1
            else:
                Y[i] = 0
```

```
            gradi = -Y[i]*(1 - sig(r[i]))*XX[i] + (1-Y[i])*(sig(r[i]))*XX[i]
            self.theta -= stepsize * gradi;
            if Y[i] == 1 or sig(r[i]) == 1: #was getting div0 error
                    JSur -= np.log(sig(r[i]))
            else:
                    JSur -= np.log(1-sig(r[i]))

            if self.classes[1] == 2: #part B
                if Y[i] == 1:
                    Y[i] = 2
                else:
                    Y[i] = 1

        J01.append( self.err(X,Y) )
        JSur /= 99
        Jnll.append(JSur)

        plt.hold(False); plt.figure(1); plt.plot(Jnll,'b-',J01,'r-'); plt.draw();
        if N==2: plt.hold(False); plt.figure(2); self.plotBoundary(X,Y);         p
        plt.pause(.01);
        if epoch > stopEpochs or (epoch > 1 and (np.absolute(Jnll[epoch - 1] - Jnll
            done = True;
```

0.0.1 ADDING METHODS TO HELP

def sig(z): # logistic sigmoid

```
return 1.0 / (1.0 + np.exp(-z) ) # in [0,1]
```

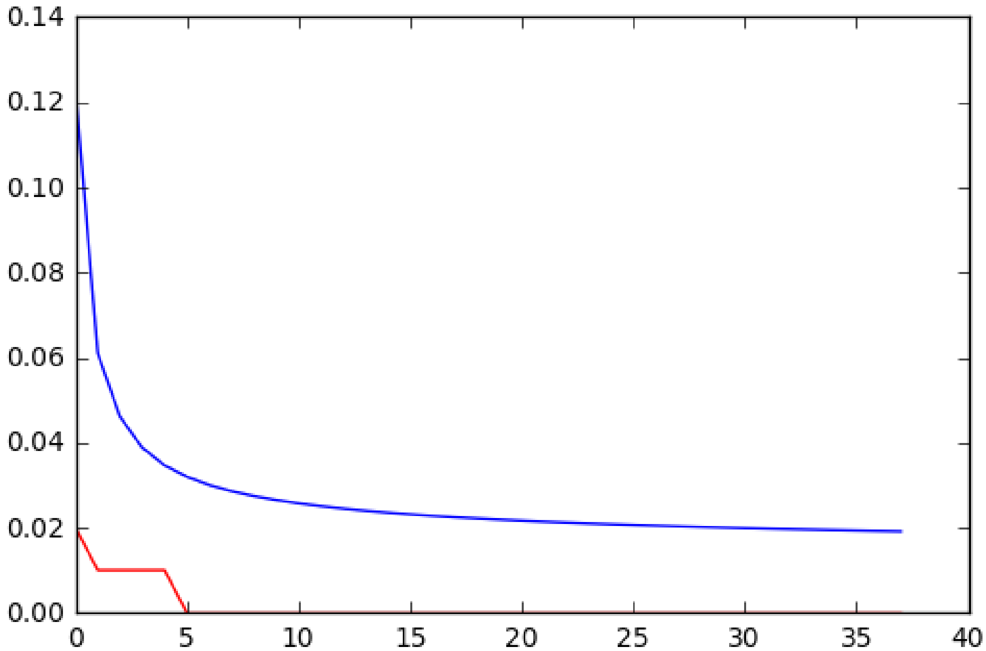
def dsig(z): # its derivative at z

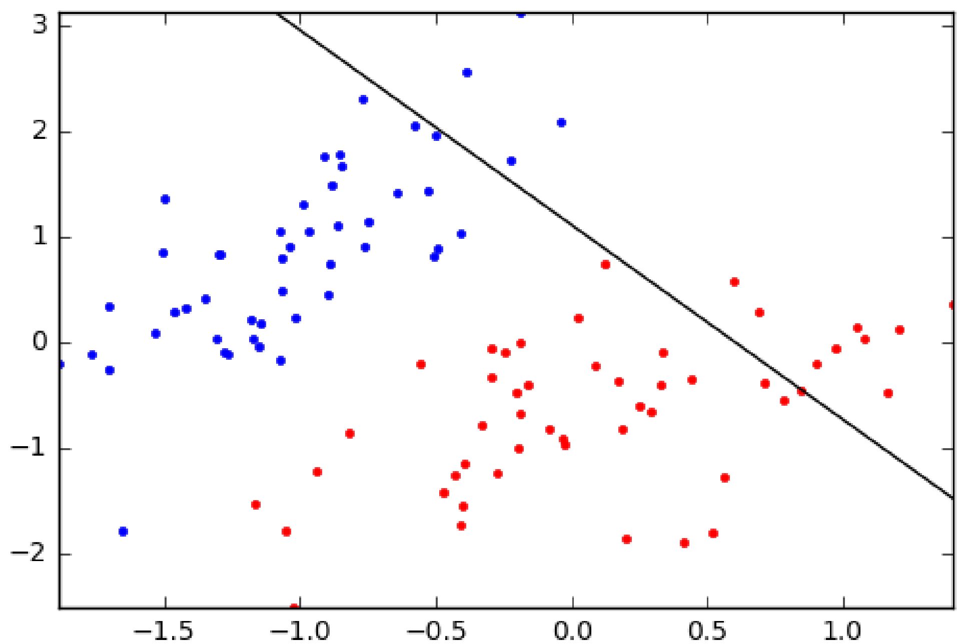
```
return sig(z) * (1-sig(z))
```

(g) Run the logistic regression classifier on both data sets (A and B). Describe your parameter choices (stepsize, etc) and show a plot showing the convergence of the surrogate loss and error rate (e.g. the loss values as a function of epoch during gradient descent)

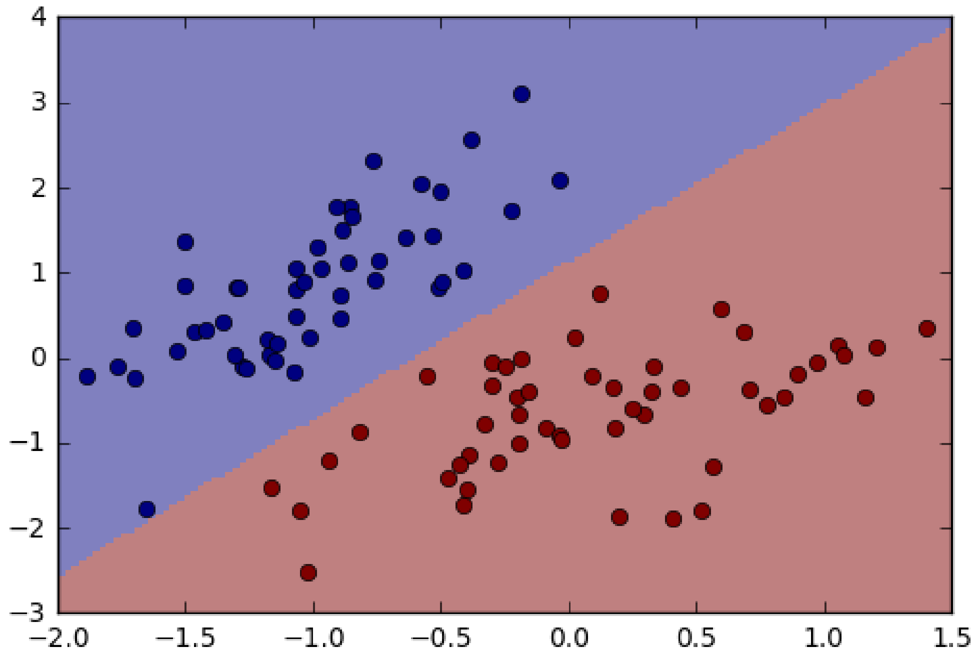
Below are the results of the above code. Stepsize, etc was given to us in the template. I just used the gradient I derived above and used the formulas in the slides as my guideline for implementation.

```
In [11]: ##DATA SET A:
         learner = logisticClassify2(); #create a new learner
         learner.classes = np.unique(YA) #[0 1]
         wts = np.array([0.5,1,-0.25]);
         learner.theta = wts #setting the parameters
         learner.train(XA, YA)
```

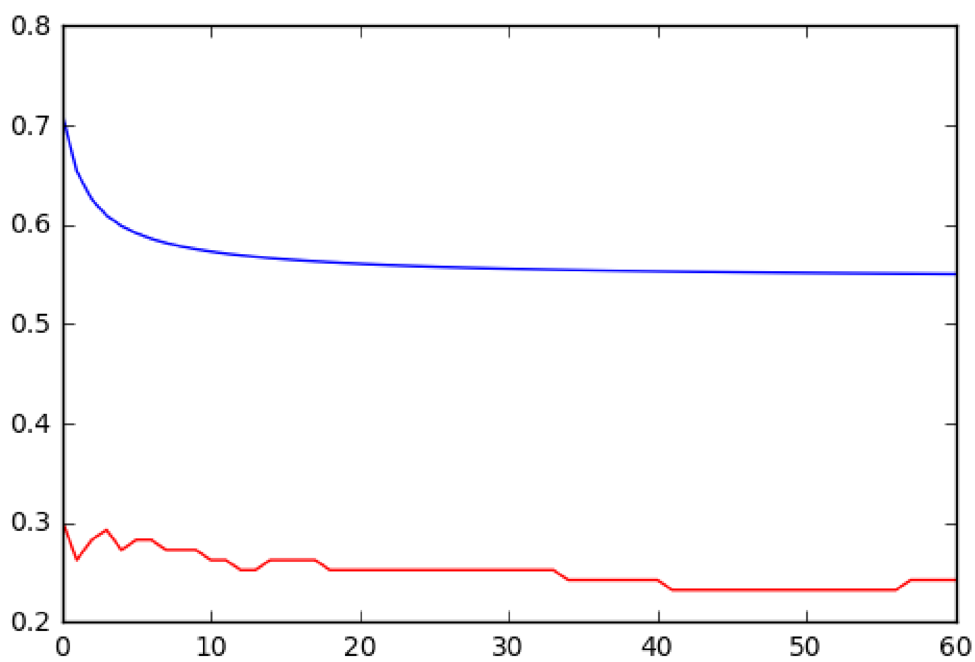


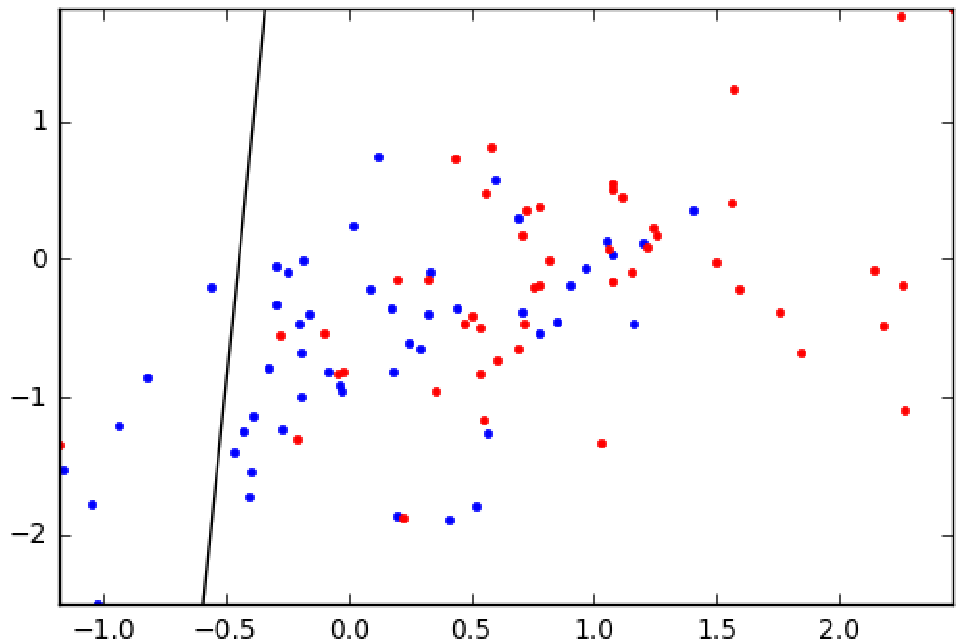



```
In [12]: ml.plot.plotClassify2D(learner, XA, YA)
```

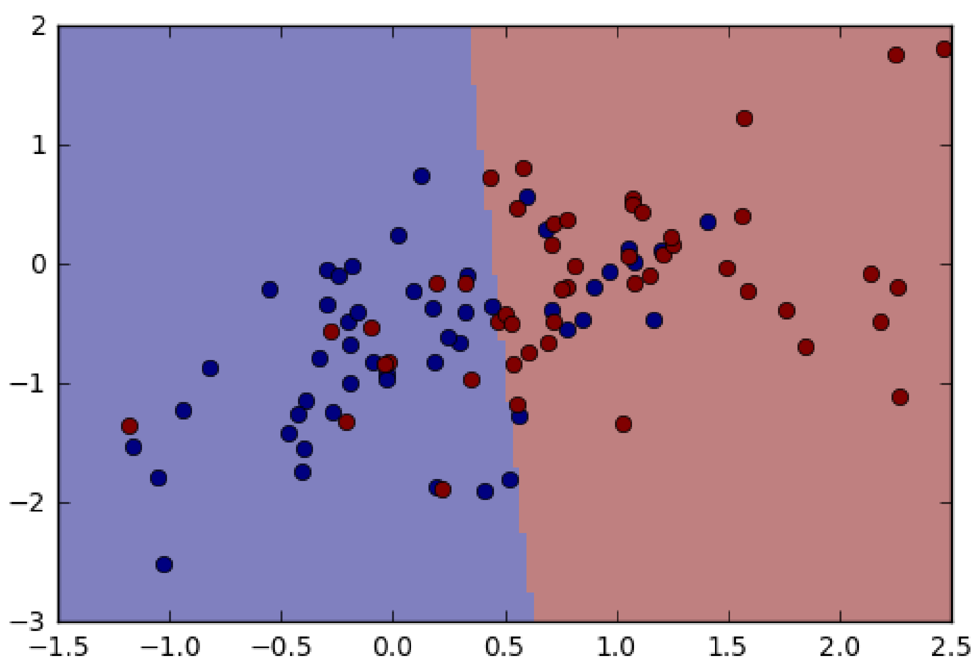



```
In [13]: ##DATA SET B:
         learner = logisticClassify2(); #create a new learner
         learner.classes = np.unique(YB) #[1 2]
         wts = np.array([0.5,1,-0.25]);
         learner.theta = wts #setting the parameters
         learner.train(XB, YB)
```






```
In [14]: ml.plot.plotClassify2D(learner, XB, YB)
```



Problem 2: Shattering and VC Dimenson:

(a) T(a +bx1): 2 parameters, 1 feature The decision boundary is a line (a + bx1 = 0) which predicts negative class on one side, positive class on the other. It can shatter a :(if point is - have line predict - on point's side, if it's + have it predict negative on it's opposite side) and b: (– place line away from points and predict negative on the point's side, -+ place between points and predict negative on - point's side, +- do the same thing as the -+ but oppositely, ++ do the same thing as – but oppositely) but not c and d: You can label the points in c and d in an XOR pattern which a line cannot shatter. => VC Dim = 2

(b) T((x1-a)ˆ2 + (x2-b)ˆ2 + c) 3 parameters, 3 features. The decision boundary is a circle that predicts class negative if the point is inside the circle and class positive outside the circle. It can shatter a: (if point is -, center circle around it, if it's positive, place circle away from point), b: (– have circle encapsulate both points, -+ have circle encapsulate first point, +- have circle encapsulate second point, ++ place circle away from both poits) and c :(same logic as b but with 3 points) but not d: (choose labels from left-to-right as - + + -). => VC Dim = 3

(c) T((a*b)x1 + (c/a)x2) 3 parameters, 2 features. While there are 3 parameters, they are not independent. Each feature has "one" (a*b for x1, c/a for x2) parameter, so the decision boundary is a line. Going through the same reasoning as part (a), it can shatter a and b but not c or d. Thus VC Dim = 2.

```
In [ ]:
```