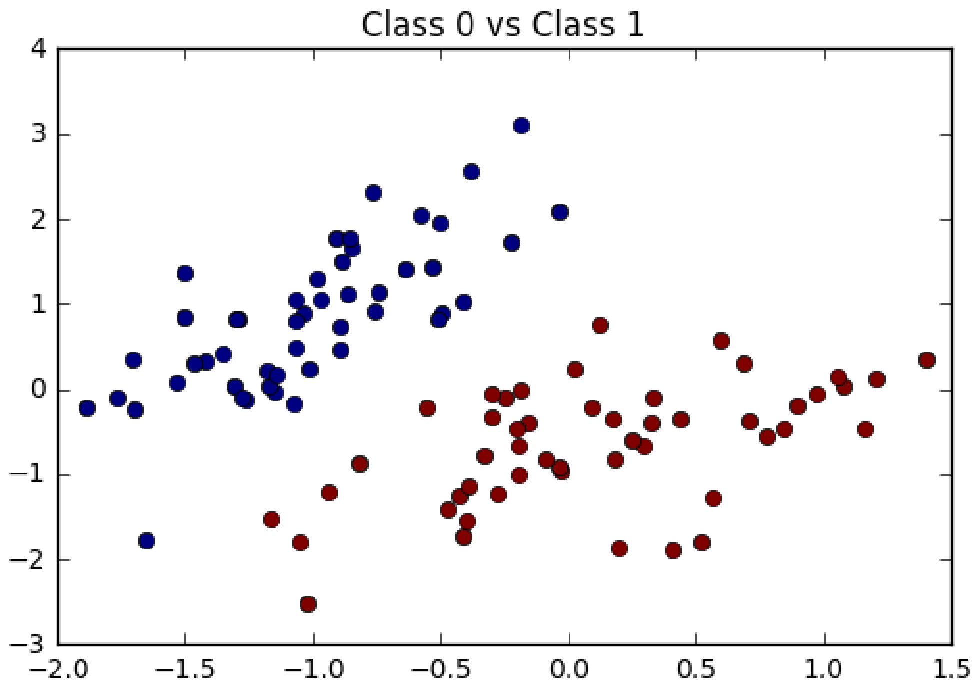
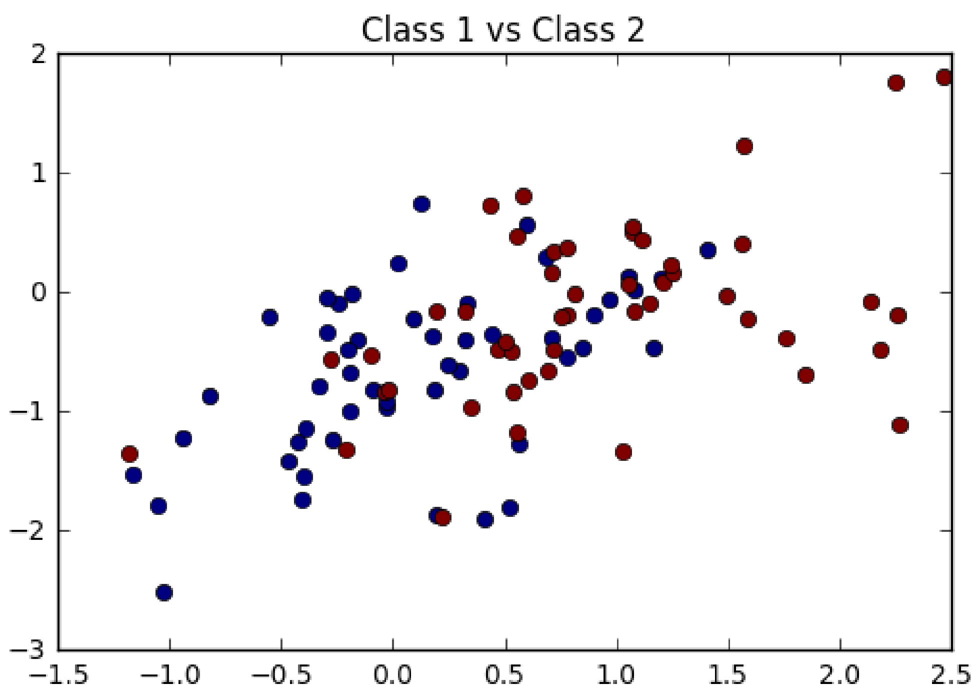
CS 178 HW 3

February 8, 2017

1 Kevin Phung, 47881547

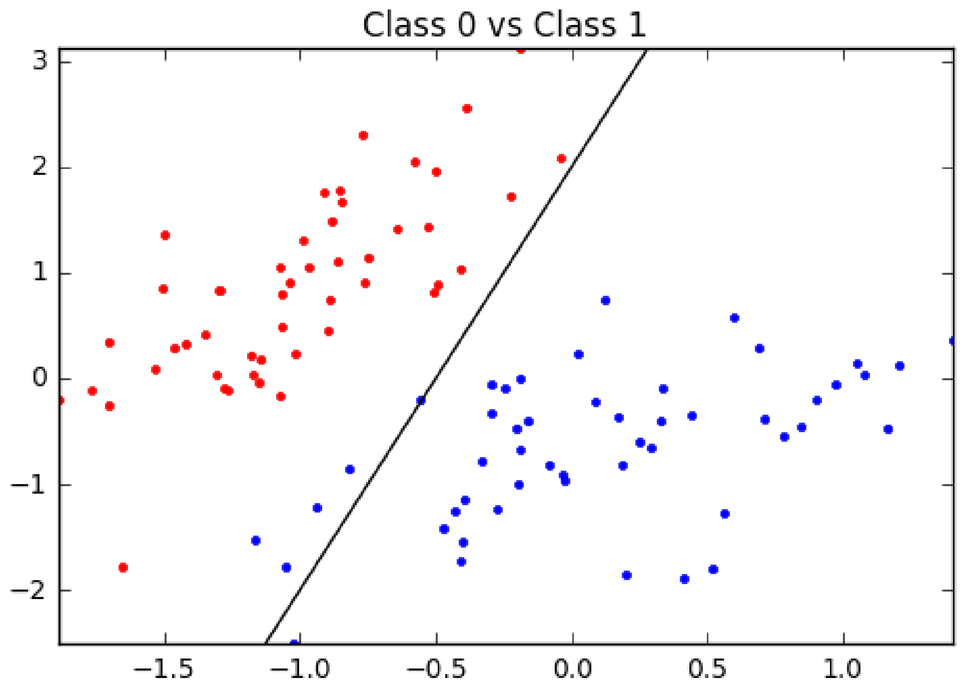
```
In [1]: # Kevin Phung 47881547

        import numpy as np
        import matplotlib.pyplot as plt
        import mltools as ml
        import logisticClassify2 as lC

        # Part A

        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

        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()
```

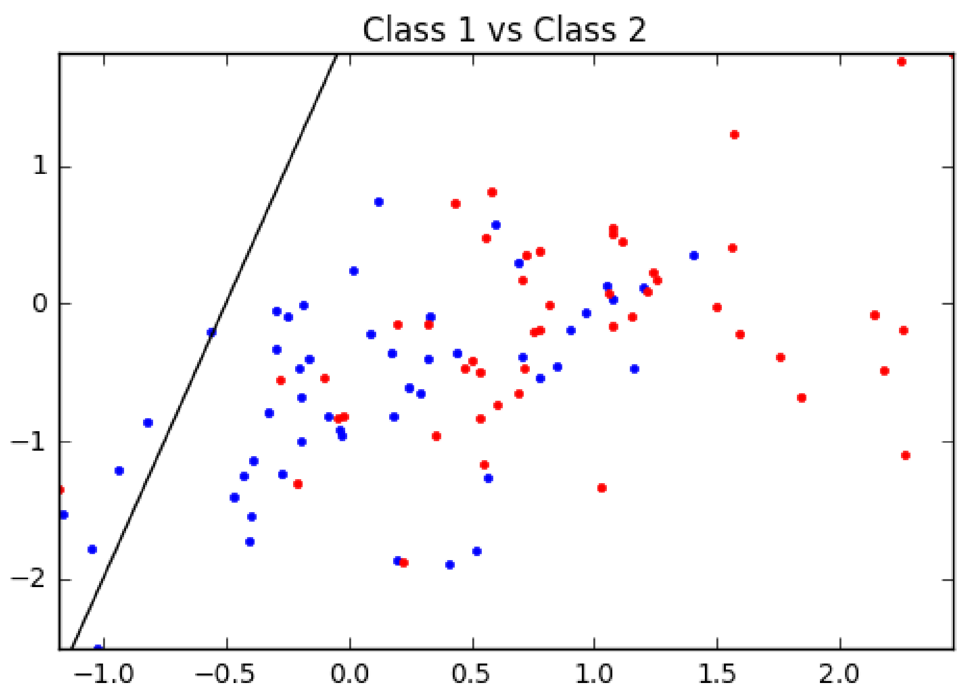

Class 0 vs Class 1
4
3
2
1
0
−1
−2
−3
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Class 1 vs Class 2
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

```
In [2]: # Part B
```

```
learnerA = lC.logisticClassify2();
learnerA.classes = np.unique(YA)
wts = np.array([.5,1,-.25])
learnerA.theta = wts

plt.title("Class 0 vs Class 1")
learnerA.plotBoundary(XA,YA)
plt.show()


learnerB = lC.logisticClassify2();
learnerB.classes = np.unique(YB)
wts = np.array([.5,1,-.25])
learnerB.theta = wts

plt.title("Class 1 vs Class 2")
learnerB.plotBoundary(XB,YB)
plt.show()

#  def plotBoundary(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
#        x2bx = (-self.theta[0] - self.theta[1]*x1b[0])/self.theta[2]
#        x2by = (-self.theta[0] - self.theta[1]*x1b[1])/self.theta[2]
#        x2b = np.array([x2bx,x2by])# TODO find x2
#                                  values as a function of x1's values
#        ## Now plot the data and the resulting boundary:
#        A = Y==1;                          # 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();
#
```


Class 0 vs Class 1
3
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0

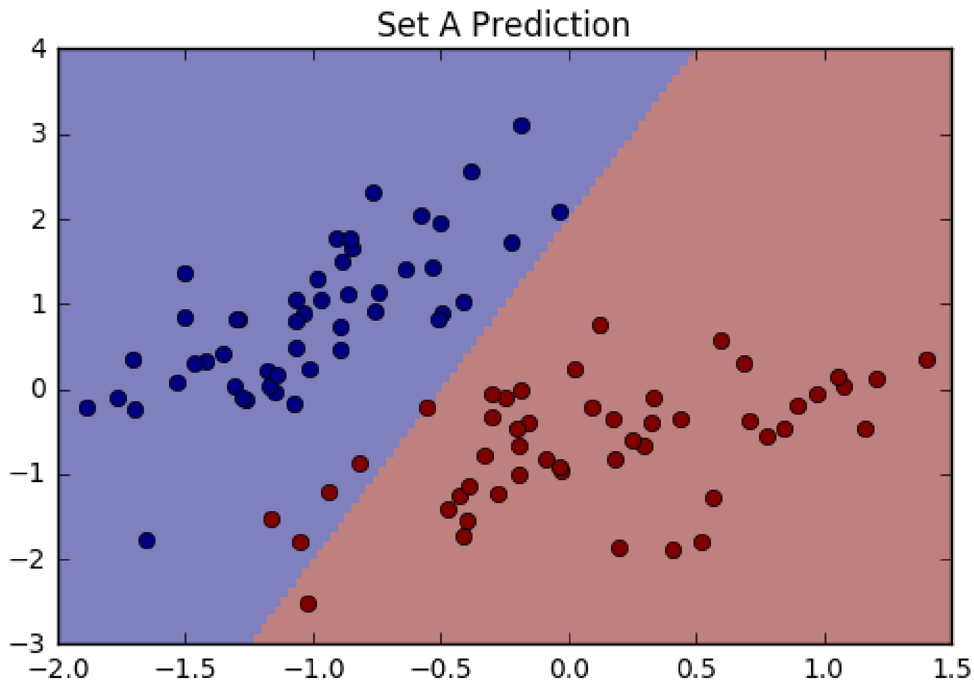

Class 1 vs Class 2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

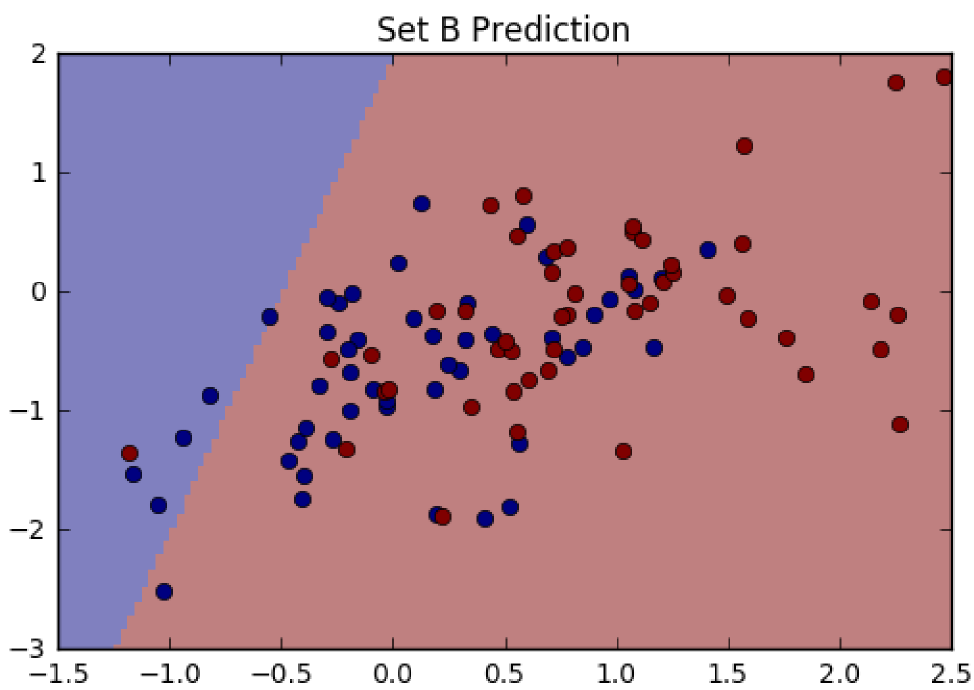
```
In [3]: # Part C
```

```
errTrainA = learnerA.err(XA,YA)
errTrainB = learnerB.err(XB,YB)
print("Set A Error Rate: "+str(errTrainA))
print("Set B Error Rate: "+str(errTrainB))
```

```
Set A Error Rate: 0.0505050505051
Set B Error Rate: 0.464646464646
```

In [4]:
```
# Part D

plt.title("Set A Prediction")
ml.plotClassify2D(learnerA,XA,YA)
plt.show()

plt.title("Set B Prediction")
ml.plotClassify2D(learnerB,XB,YB)
plt.show()

#    def predict(self, X):
#        """ Return the predictied class of each data point in X"""
#
#        r = []
#        Yhat = []
#        for i,l in enumerate(X):
#            r.append(self.theta[0] + self.theta[1]*X[i,0]
#                                   + self.theta[2]*X[i,1])
#            if r[i] > 0:
#                Yhat.append(self.classes[1])
#            else:
#                Yhat.append(self.classes[0])
#        Yhat = np.array(Yhat)
#
#        return Yhat
```



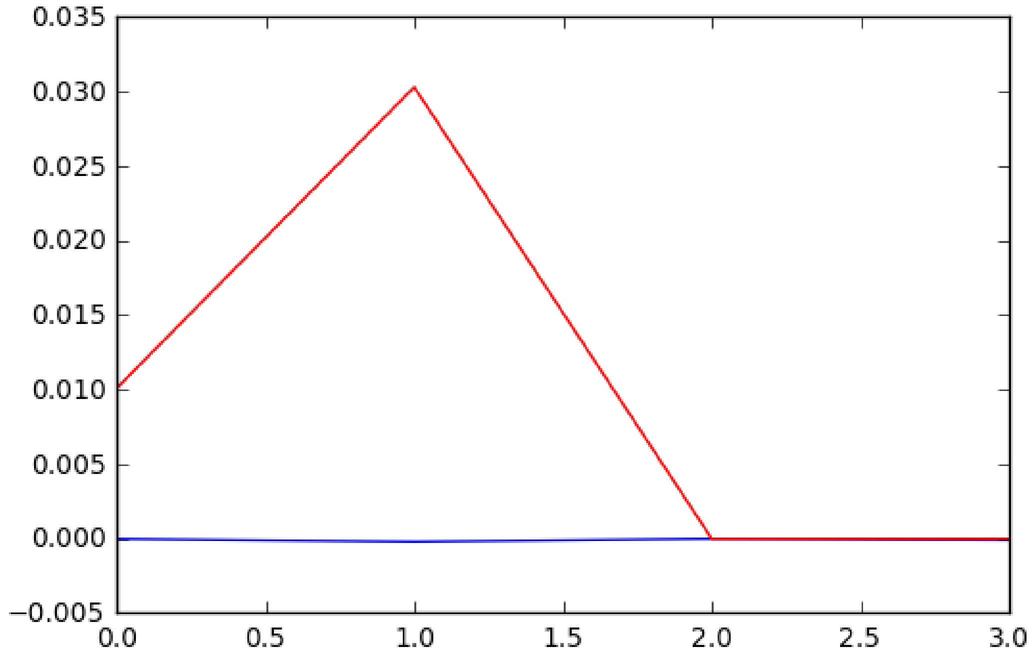


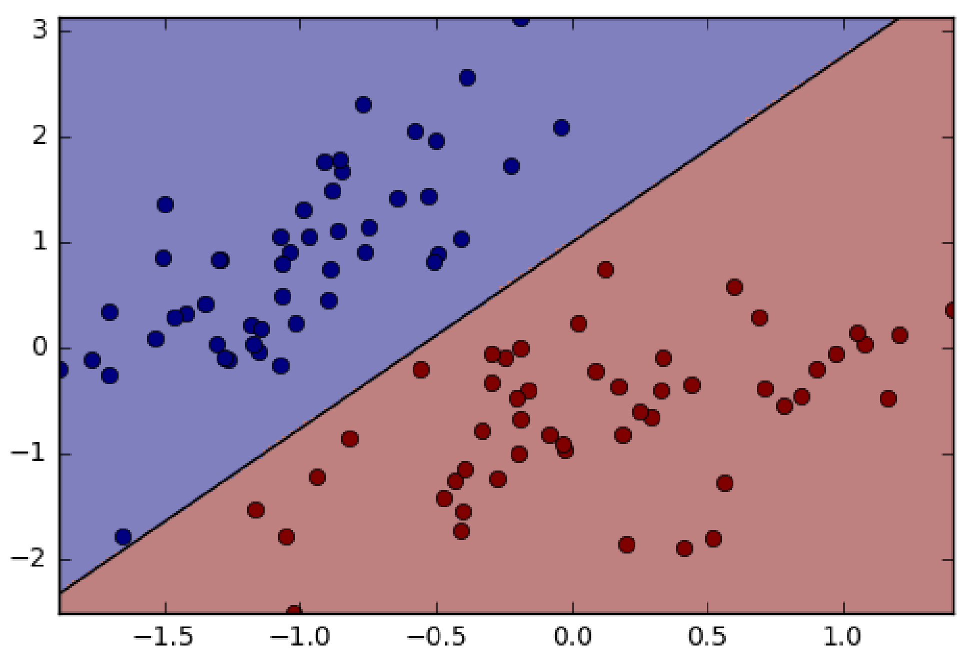

```
In [5]: # Part E
        # J/J(theta) = -y^(j)log(sigmoid(x^(j)theta^T))
```

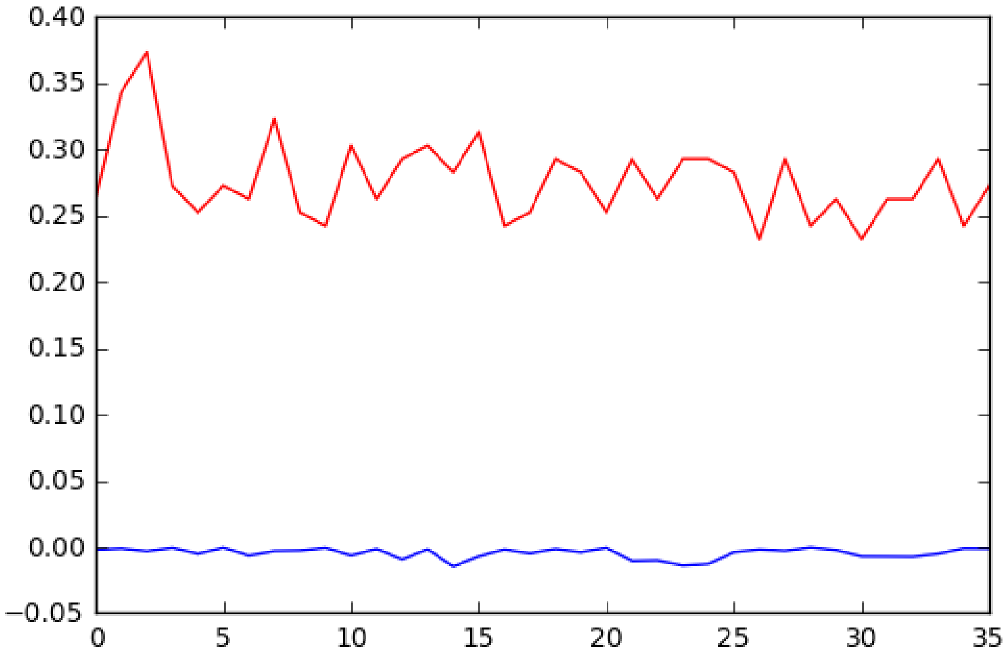
```
#                 - (1-y^(j))*log(1-sigmoid(x^(j)theta^T))
#             = -y^(j) * (1/sigmoid(x^J theta^T)) * sigmoid(x^J theta^T)
#                 * (1-sigmoid(x^J theta^T))
#                  - (1-y^J) * (1/ (1-sigmoid(x^J theta^T))
#                 * -sigmoid(x^J theta^T)* (1-sigmoid(x^J theta^T))
#             = (-y^(j)) * (1-sigmoid(x^J theta^T)
#                  + (1-y^j) * sigmoid(x^J theta^T)
```

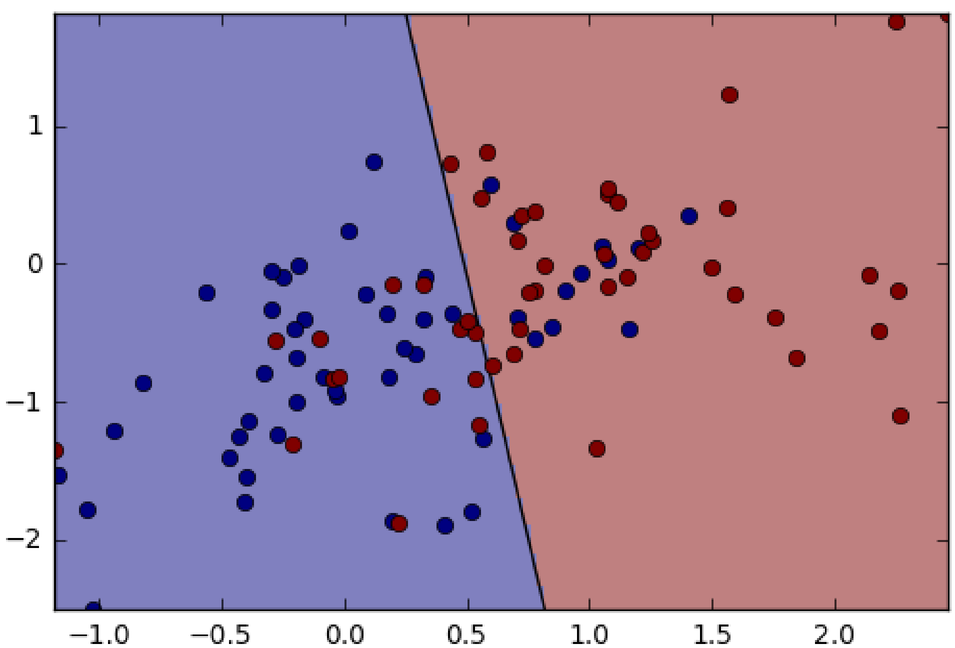
```
In [6]: # Part F/G
        learnerA2 = lC.logisticClassify2(XA,YA);
        ml.plotClassify2D(learnerA2,XA,YA)
        plt.show()

        learnerB2 = lC.logisticClassify2(XB,YB)
        ml.plotClassify2D(learnerB2,XB,YB)
        plt.show()

        #    def sign(self,r):
        #       return (1.0/(1.0/np.exp(-r)))
        #
        #     def train(self, X, Y, initStep=1.0, stopTol=1e-4, stopEpochs=5000, pl
        #        """ Train the logistic regression using stochastic gradient descen
        #
        #        M,N = X.shape;                     # initialize the model if neces
        #        self.classes = np.unique(Y);       # Y may have two classes, any v
        #        XX = np.hstack((np.ones((M,1)),X))   # XX is X, but with an extra
        #        YY = ml.toIndex(Y,self.classes);   # YY is Y, but with canonical v
        #        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 = 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[0] + self.theta[1] * X[i,0] + self.theta[2
        #
        #                si = self.sig(ri)
        #                theta1Grad = -YY[i] * (1 - si) + (1 - YY[i]) * si
        #                theta2Grad = -YY[i] * (1 - si) * X[i,0]
        #                             + (1 - YY[i]) * si * X[i,0]
        #                theta3Grad = -YY[i] * (1 - si) * X[i,1]
        #                             + (1 - YY[i]) * si * X[i,1]
        #                gradi = np.array([theta1Grad, theta2Grad, theta3Grad])
        #
        #                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-likelih
#             ##  Jsur = sum_i [ (log si) if yi==1 else (log(1-si)) ]
#
#             Jsur = np.sum([np.log(si) if YY[i] == 1 else np.log(1-si)])
#             Jnll.append(Jsur/M)
#
#
#             #plt.figure(1); plt.plot(Jnll,'b-',J01,'r-'); plt.draw();    #
#             #if N==2: plt.figure(2); self.plotBoundary(X,Y); plt.draw(); #
#             #plt.pause(.01);                    # let OS draw the plot
#
#             ## For debugging: you may want to print current parameters & l
#             # print self.theta, ' => ', Jsur[-1], ' / ', J01[-1]
#             # raw_input()   # pause for keystroke
#
#             # TODO check stopping criteria: exit if exceeded # of epochs
#             if (epoch > stopEpochs):
#                 done = True
#             if (epoch > 1 and abs(Jnll[-2]-Jnll[-1] < stopTol)):
#                 done = True
#
#         plt.figure(1); plt.plot(Jnll,'b-',J01,'r-'); plt.draw();    # plot
#         if N==2: plt.figure(2); self.plotBoundary(X,Y); plt.draw(); # & pr
#         plt.pause(.01);                    # let OS draw the plot

"""
    For my parameters for Set A, I chose XA and YA. For set B,
    I chose XB and YB
"""
```


0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−0.005
0.0
0.5
1.0
1.5
2.0
2.5
3.0


3
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0






```
Out[6]: '\n    For my parameters for Set A, I chose XA and YA. For set B,\n    I ch
```

```
In [ ]: # Problem 2: Shattering and VC Dimension
```

```python
# 2 Part A
"""
    Learner A = T(a+bx1) = line
    For learner A, data points in a and b can be shattered because we can
    separate them into -1 and 1 and they will be divided by the
    function T(a+bx1). C and D cannot be shattered because for C, if
    the first and last points are the same while the middle point
    is different it cannot be shattered. For D if the first and last
    points are the same it cannot be shattered.
"""

# 2 Part B
"""
    Learner B = T((x1-a)^2 + (x2-b)^2+c) = circle
    For learner B, data points in a,b, and c can be shattered because
    we can  separate them and they will be divided. D cannot be
    shattered  because the points can be on the same side of the
    circle but have different values.
"""

# 2 Part C
"""
    Learner C = T((a*b)x1 + (c/a)x2) = line
    For learner C, data points in a and b can be shattered because we can
    separate them into -1 and 1 and they will also be divided. C and D
    cannot be shattered if the points on the same side of the line are
    different.
"""
```