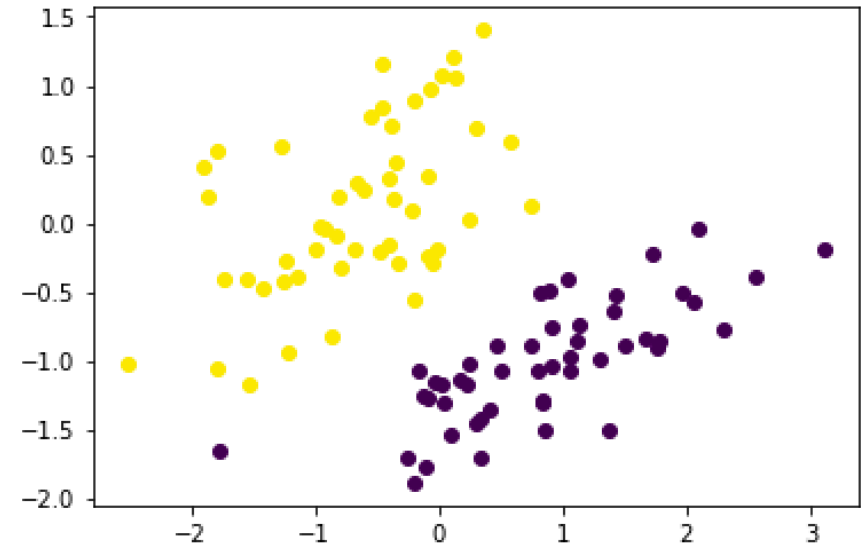
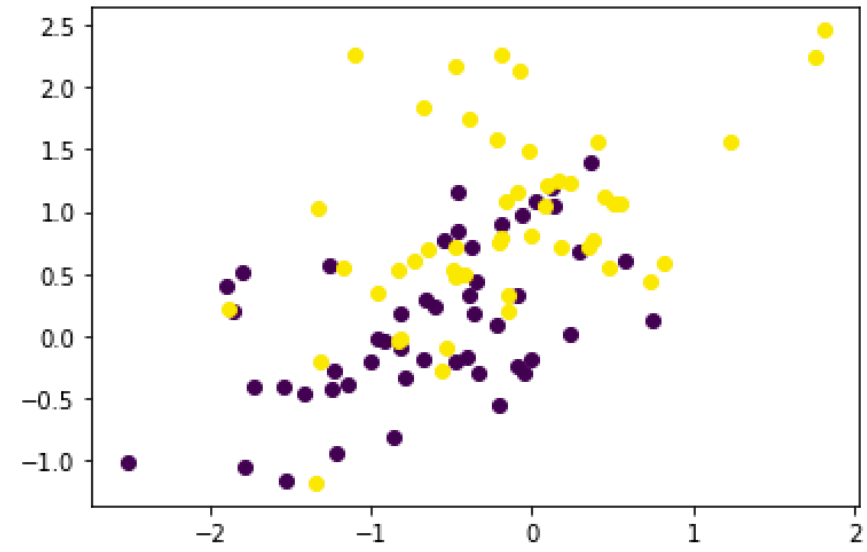
Mohamed Kharaev
43121144

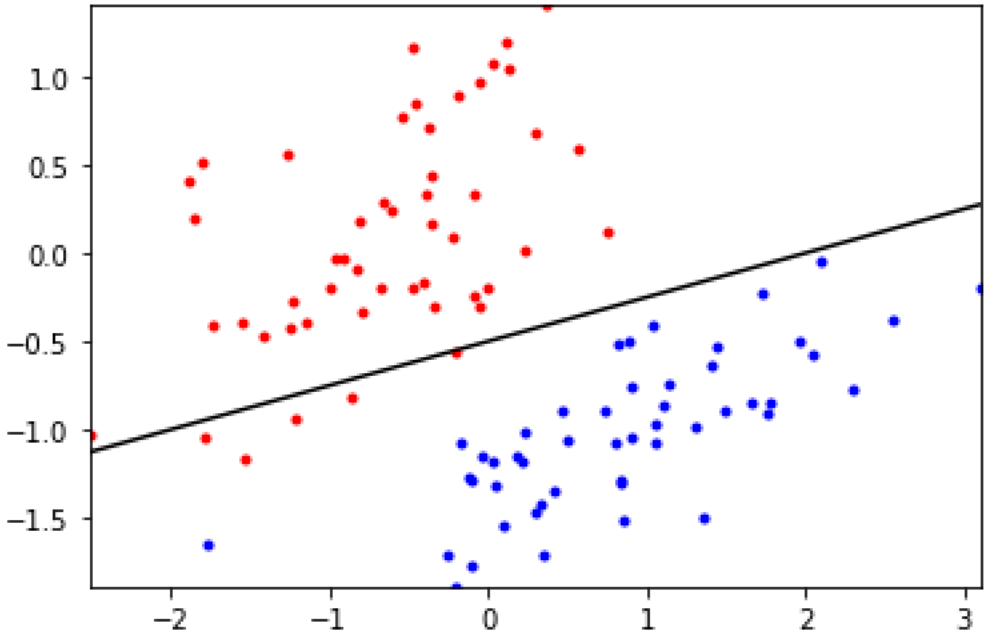
CS 178 HW 3

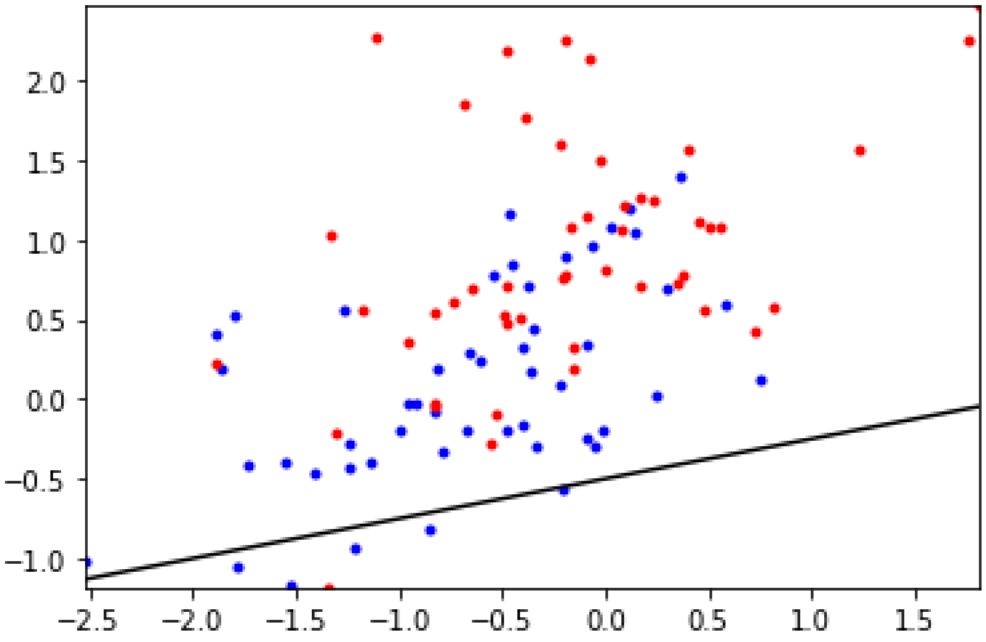
Part 1





Part 2


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

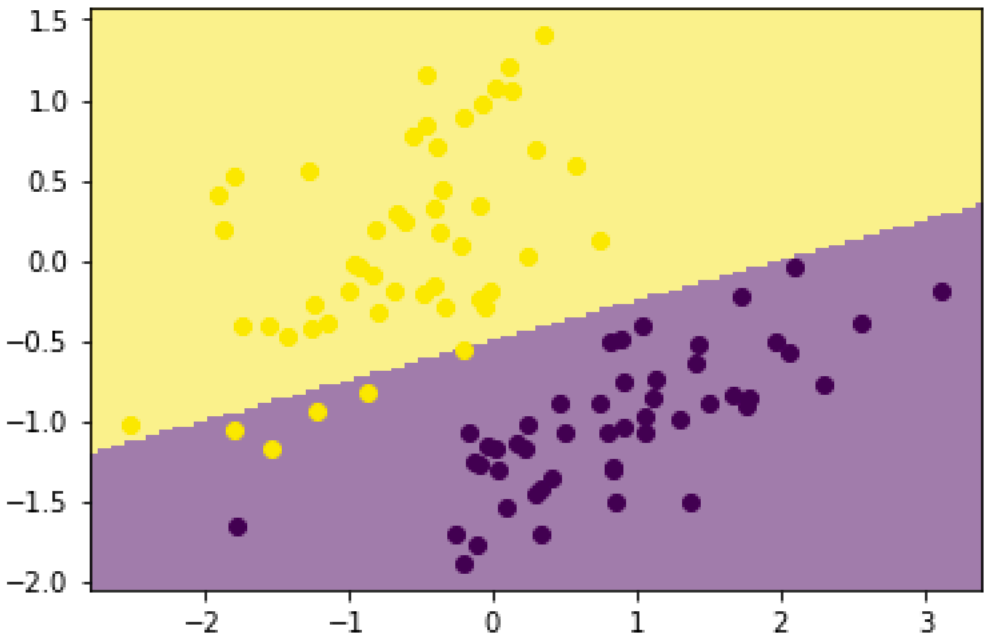

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

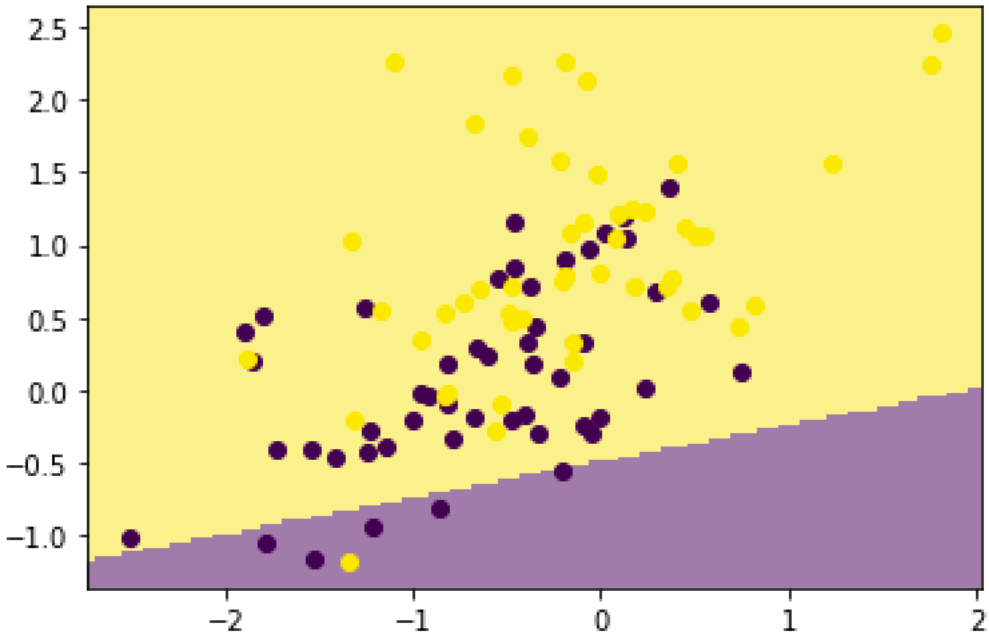
Part 3
ERROR: 0.050505050505050504

ERROR: 0.46464646464646464

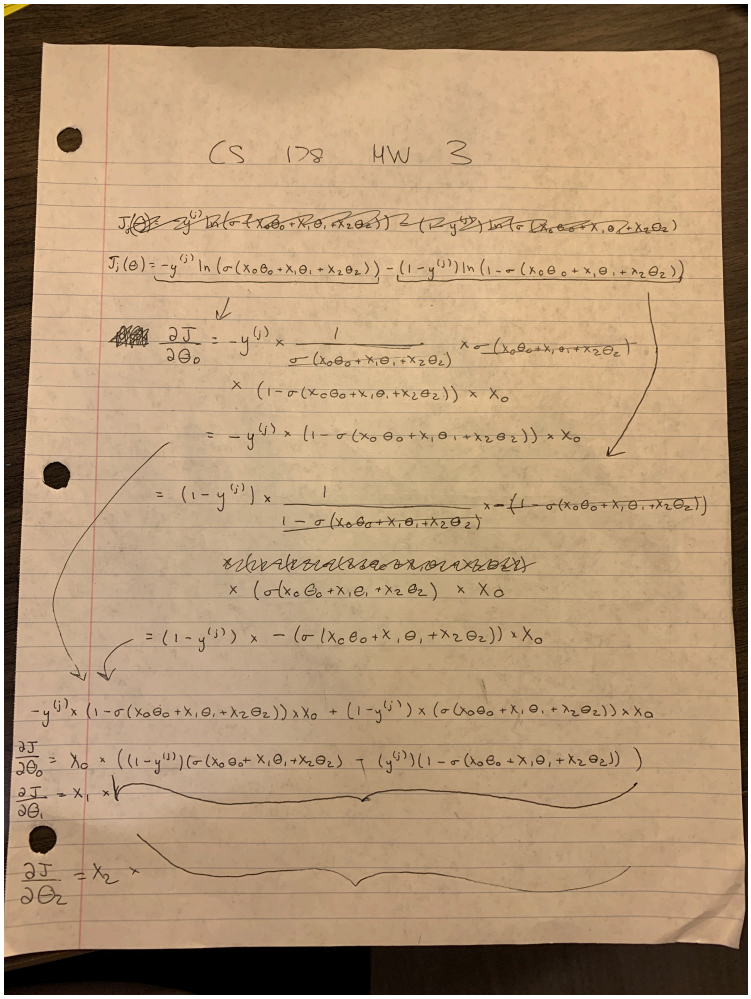
```python
def predict(self, X):
    """ Return the predictied class of each data point in X"""
    ## TODO: compute linear response r[i] = theta0 + theta1 X[i,1] + theta2 X[i,2] + ... for each i
    ## TODO: if z[i] > 0, predict class 1:  Yhat[i] = self.classes[1]
    ##       else predict class 0: 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

Part 4



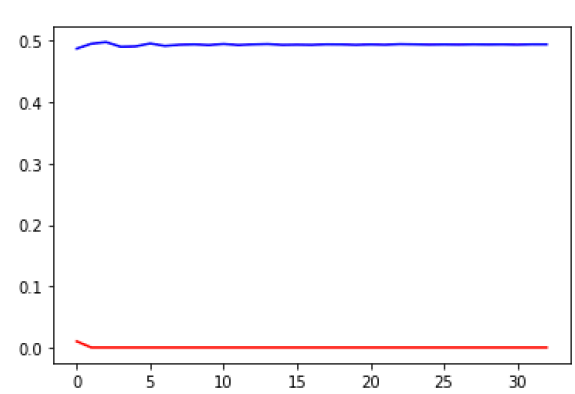


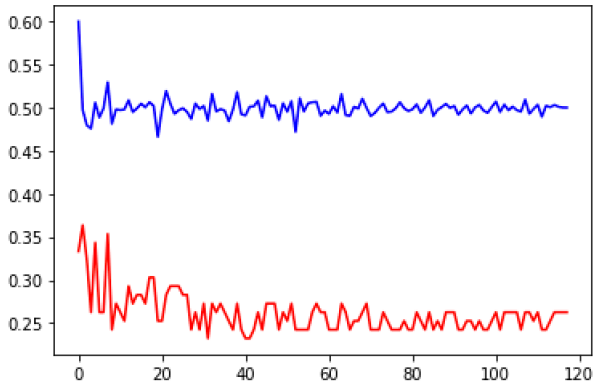
Part 5

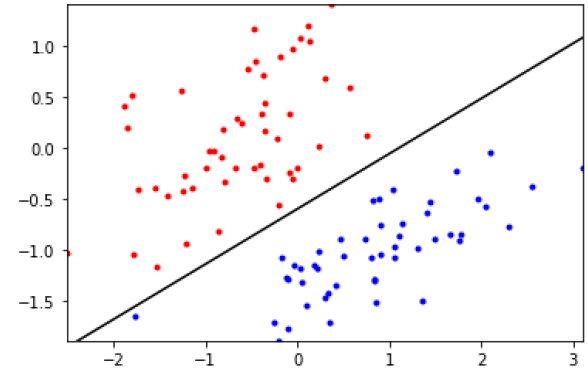


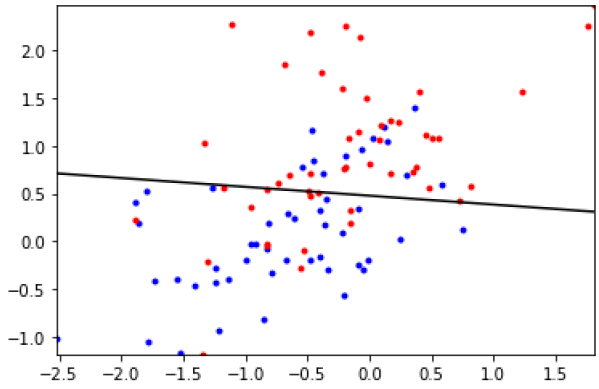
```
def train(self, X, Y, initStep=1.0, stopTol=1e-4, stopEpochs=5000, plot=None):
    """ Train the logistic regression using stochastic gradient descent """
    M,N = X.shape;                  # initialize the model if necessary:
    self.classes = np.unique(Y);       # Y may have two classes, any 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:
        Jsur = 0
        for i in np.random.permutation(M):
            ri    = (self.theta[0] * XX[i,0]) + (self.theta[1] * XX[i, 1]) + (self.theta[2] * XX[i, 2]);     # TODO: compute linear response r(x)
            gradi = np.array(
                    [XX[i,0] * ((1 - YY[i]) * (self.sigmoid(ri)) - (YY[i]) * (1 - self.sigmoid(ri))),
                     XX[i,1] * ((1 - YY[i]) * (self.sigmoid(ri)) - (YY[i]) * (1 - self.sigmoid(ri))),
                     XX[i,2] * ((1 - YY[i]) * (self.sigmoid(ri)) - (YY[i]) * (1 - self.sigmoid(ri)))]);     # TODO: compute gradient of NLL loss
            self.theta -= stepsize * gradi;  # take a gradient step
            if (YY[i] == 1):
                Jsur = np.add(Jsur, np.log10(self.sigmoid(ri)))
            else:
                Jsur = np.add(Jsur, 1 - np.log10(1 - self.sigmoid(ri)))

        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)) ]
        Jsur = Jsur / M
        Jnll.append( Jsur ) # TODO evaluate the current NLL loss
        plt.figure(1); plt.plot(Jnll,'b-',J01,'r-'); plt.draw();    # plot losses
        if N==2: plt.figure(2); self.plotBoundary(X,Y); plt.draw(); # & 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)
        # or if Jnll not changing between epochs ( < stopTol )
        if epoch > stopEpochs:
            done = True
        if len(Jnll) > 2:
            if abs(Jnll[-1] - Jnll[-2]) < stopTol:
                done = True

def sigmoid(self, r):
    return ((1 + math.exp(-1 * r)) ** -1);
```

Part 6










```
PROBLEM 2
-A-
        1. Yes, having an 'a' value of 0 and any positive 'b' value will make T[z] = 1 for any x1 > 0
        2. Yes, positive values for 'a', 'b', and 'c' will always make T[z] = 1 for any x1 > 0 && x2 > 0
        3. Yes, having any positive value for 'c' will make T[z] = 1 for any x1 and x2, because (x1 - a)^2 and (x2 - b)^2 will always be >= 0. The sum will be positive
        4. Yes, having positive values for 'a', 'b', 'c', and 'd' will make T[z] = 1 for any x1 > 0 && x2 > 0
-B-
        1. Yes, having an 'a' value of 0 and any positive 'b' value will make T[z] = 1 for any x1 > 0
        2. Yes, positive values for 'a', 'b', and 'c' will always make T[z] = 1 for any x1 > 0 && x2 > 0
        3. Yes, having any positive value for 'c' will make T[z] = 1 for any x1 and x2, because (x1 - a)^2 and (x2 - b)^2 will always be >= 0. The sum will be positive
        4. Yes, having positive values for 'a', 'b', 'c', and 'd' will make T[z] = 1 for any x1 > 0 && x2 > 0
-C-
        1. Yes, having an 'a' value of 0 and any positive 'b' value will make T[z] = 1 for any x1 > 0
        2. Yes, positive values for 'a', 'b', and 'c' will always make T[z] = 1 for any x1 > 0 && x2 > 0
        3. Yes, having any positive value for 'c' will make T[z] = 1 for any x1 and x2, because (x1 - a)^2 and (x2 - b)^2 will always be >= 0. The sum will be positive
        4. Yes, having positive values for 'a', 'b', 'c', and 'd' will make T[z] = 1 for any x1 > 0 && x2 > 0
-D-
        1. Yes, having an 'a' value of 0 and any positive 'b' value will make T[z] = 1 for any x1 > 0
        2. Yes, positive values for 'a', 'b', and 'c' will always make T[z] = 1 for any x1 > 0 && x2 > 0
        3. Yes, having any positive value for 'c' will make T[z] = 1 for any x1 and x2, because (x1 - a)^2 and (x2 - b)^2 will always be >= 0. The sum will be positive
        4. Yes, having positive values for 'a', 'b', 'c', and 'd' will make T[z] = 1 for any x1 > 0 && x2 > 0
```

I worked by myself