273A hw3

Enyu Huang

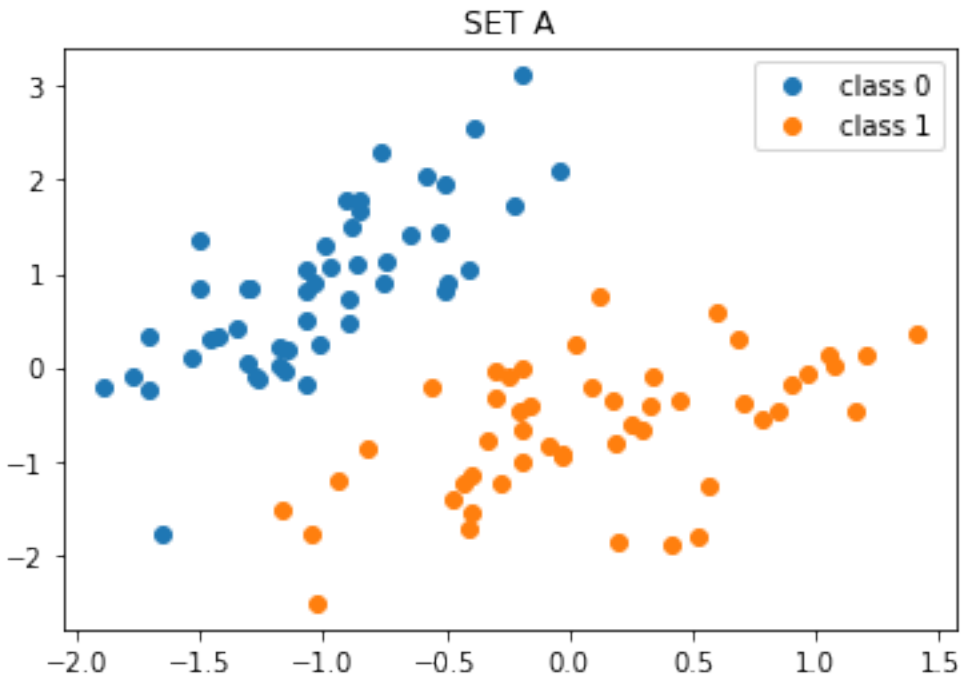
February 5, 2021

1 273A hw 3

2 Problem 1

2.1 problem 1.1

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml

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

plt.plot(XA[YA==0,0],XA[YA==0,1],"o", label = "class 0")
plt.plot(XA[YA==1,0],XA[YA==1,1],"o", label = "class 1")
plt.title("SET A")
plt.legend()
plt.show()
plt.title("SET B")
plt.plot(XB[YB==1,0],XB[YB==1,1],"o", label = "class 1")
plt.plot(XB[YB==2,0],XB[YB==2,1],"o", label = "class 2")
plt.legend()
plt.show()
```

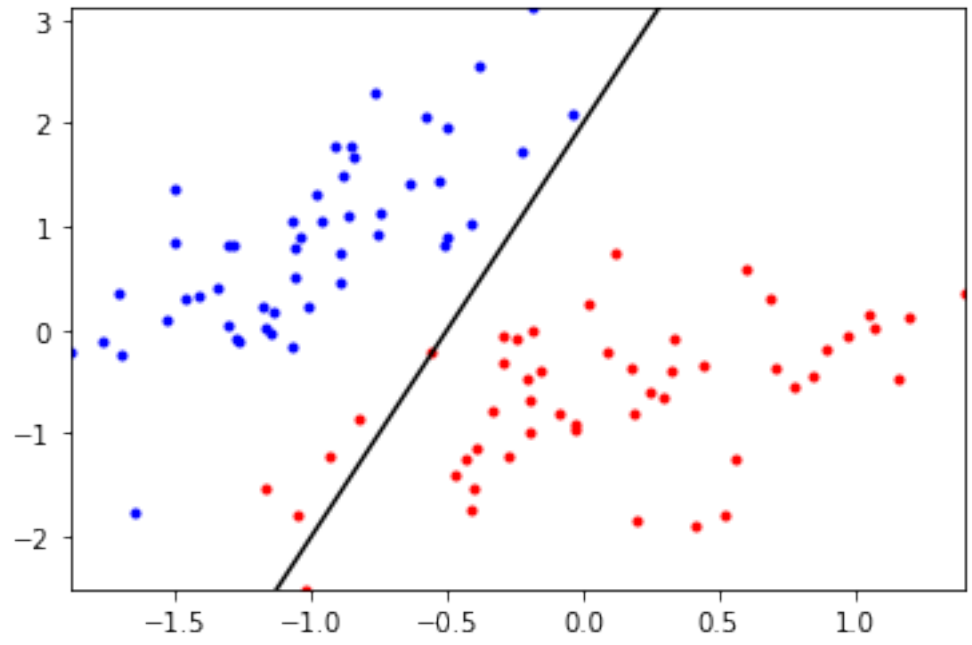

SET A
class 0
class 1
3
2
1
0
−1
−2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

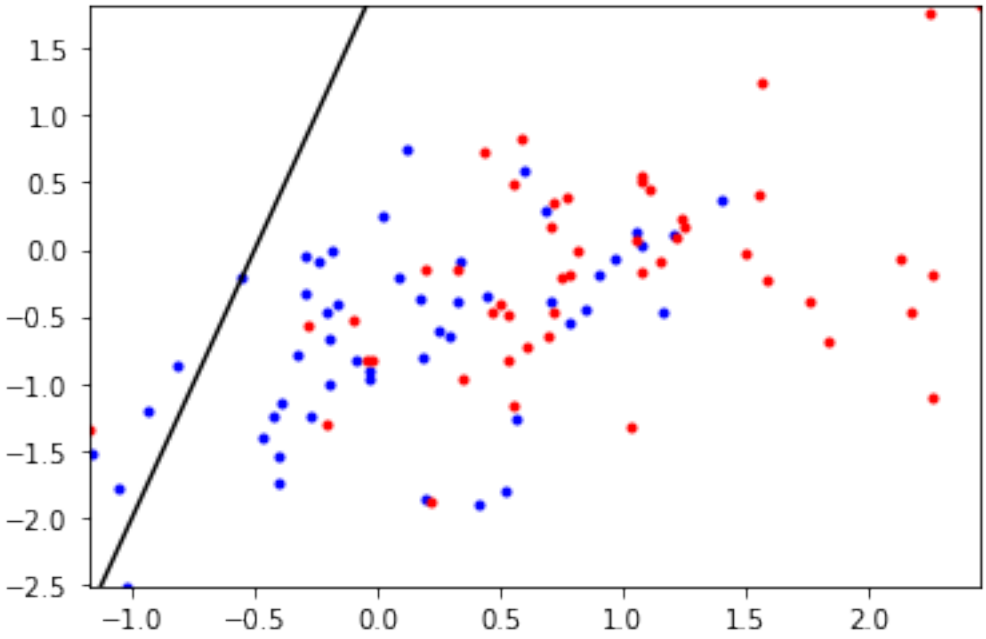

SET B
class 1
class 2
2
1
0
−1
−2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

2.2 problem 1.2

```
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
x2b = []
for value in x1b:
    x2b.append((self.theta[0] + self.theta[1]*value)/-self.theta[2]) #finding x2

#make x2b consistent with x1b
x2b = np.array(x2b)
## 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();
```

[2]:
```
from logisticClassify2 import *
# Create a learner for SET A only
learnerA = logisticClassify2(); # create "blank" learner
learnerA.classes = np.unique(YA) # define class labels using YA or YB
wts = np.array([0.5,1,-0.25]); # TODO: fill in values
learnerA.theta = wts; # set the learner's parameters
learnerA.plotBoundary(XA,YA)
plt.show()
# Create a learner for SET B only
learnerB = logisticClassify2(); # create "blank" learner
learnerB.classes = np.unique(YB) # define class labels using YA or YB
wts = np.array([0.5,1.,-0.25]); # TODO: fill in values
learnerB.theta = wts; # set the learner's parameters
learnerB.plotBoundary(XB,YB)
plt.show()
```





2.3 Problem 1.3

```
def predict(self, X):
""" Return the predictied class of each data point in X"""
```

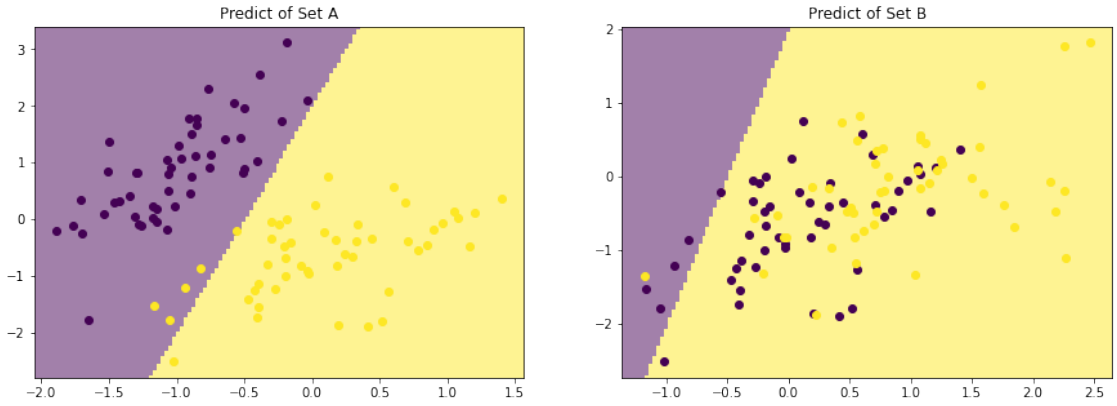
```
    ## TODO: compute linear response r[i] = theta0 + theta1 X[i,1] + theta2 X[i,2] + ... for eac
    ## TODO: if z[i] > 0, predict class 1:  Yhat[i] = self.classes[1]
    ##       else predict class 0:  Yhat[i] = self.classes[0]
    r = np.zeros(X.shape[0]);
    Yhat = np.zeros(X.shape[0]);
    for i in range(len(X)):
        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
```

[3]:
```
err_A = learnerA.err(XA,YA)
print("Error rate of A:",err_A)
err_B = learnerB.err(XB,YB)
print("Error rate of B:",err_B)
```

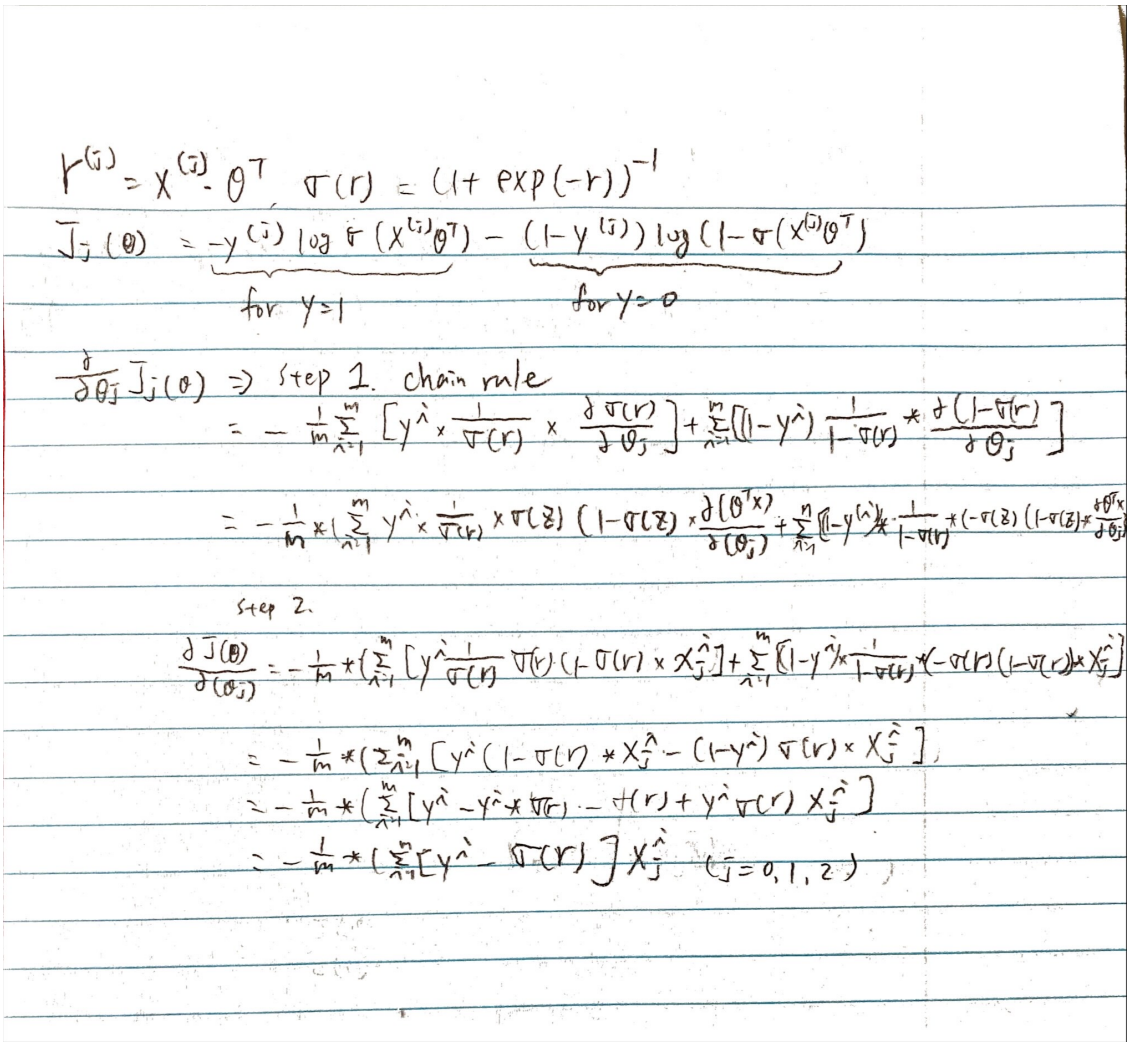
```
Error rate of A: 0.050505050505050504
Error rate of B: 0.46464646464646464
```

2.4 Problem 1.4

[4]:
```
fig, ax = plt.subplots(1, 2, figsize = (15,5));

ax[0].set_title("Predict of Set A")
ml.plotClassify2D(learnerA,XA,YA,axis = ax[0])
ax[1].set_title("Predict of Set B")
ml.plotClassify2D(learnerB,XB,YB,axis = ax[1])
```



2.5 problem 1.5



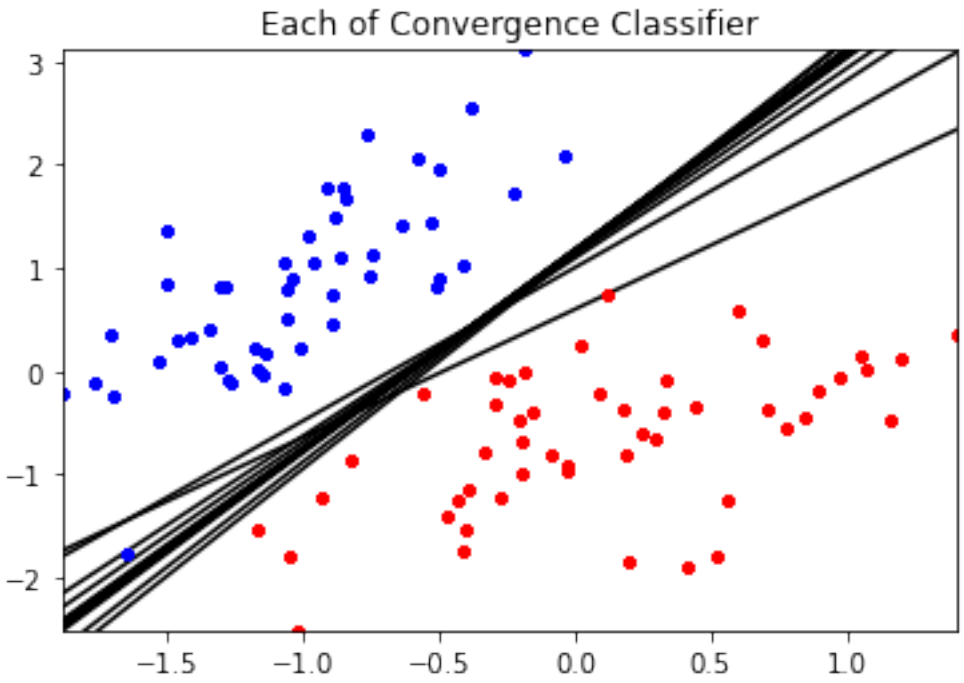
2.6 problem 1.6

```
def train(self, X, Y,alpha,initStep=0.2, stopTol=1e-4, stopEpochs=5000, plot=None):


""" Train the logistic regression using stochastic gradient descent """
M,N = X.shape;
self.classes = np.unique(Y);
XX = np.hstack((np.ones((M,1)),X))
YY = ml.toIndex(Y,self.classes);
if len(self.theta)!=N+1:
    self.theta=np.random.rand(N+1);
#(r) = 1 / (1 + exp(r))
def sigma(r):
    Sigma_r = (1+np.exp(-r)) ** -1
    return Sigma_r
```

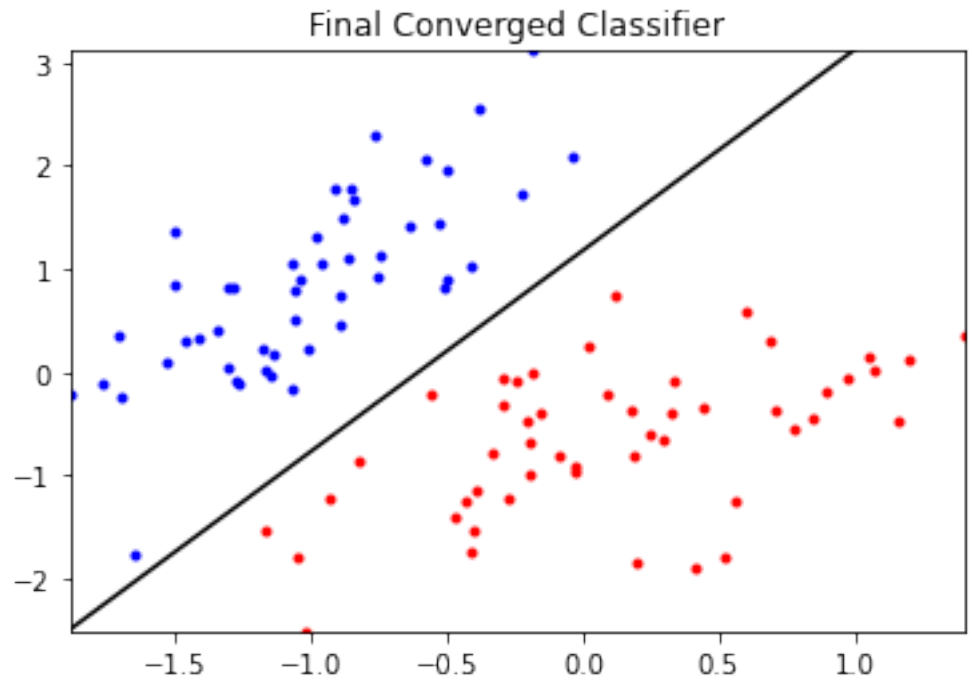
```
# init loop variables:
epoch=0; done=False; Jnll=[np.inf]; J01=[np.inf];
while not done:
    stepsize, epoch = initStep*2.0/(2.0+epoch),epoch+1;
     # Do an SGD pass through the entire data set:
    for i in np.random.permutation(M):
        # TODO: compute linear response r(x)
        # r = x
        ri = self.theta[0]*XX[i,0] + self.theta[1]*XX[i,1] + self.theta[2]*XX[i,2];
        # TODO: compute gradient of NLL loss
        # (x, y) = 2(y  (x))* (x) x
        gradi = 2*(-YY[i] + sigma(ri))*XX[i,:] ;
        self.theta -= stepsize * gradi;
    J01.append( self.err(X,Y) )
    ## TODO: compute surrogate loss
    Jsurrogate = 0;
    for i in np.random.permutation(M):
        #(x, y) =  log p(y | x) =  y log (x)  (1  y)log(1  (x))
        Jsurrogate += -YY[i]*np.log(sigma(np.dot(self.theta, XX[i,:])))-(1-YY[i])*np.log(1-s
        # Add regularization term into surrogate loss function for problem 1.8, set alpha =
    Jsur = (Jsurrogate + alpha*(self.theta[0]**2+self.theta[1]**2+self.theta[2]**2))/M
    ##  Jsur = sum_i [ (log si) if yi==1 else (log(1-si)) ]
    Jnll.append( Jsur ) # TODO evaluate the current NLL loss
    if N==2:
        plt.figure(1);
        self.plotBoundary(X,Y);
        plt.title("Each of Convergence Classifier")
        #plt.draw(); # & predictor if 2D
    ## For debugging:
    ## print self.theta, ' => ', Jnll[-1], ' / ', J01[-1]
    ## raw_input()   # pause for keystroke
    # TODO check stopping criteria:
    if epoch > stopEpochs or np.abs(Jnll[-2] - Jnll[-1]) < stopTol:
        done = True;
plt.show()  # show the overall convergence line in a graph
plt.figure(2);
self.plotBoundary(X,Y);
plt.title("Final Converged Classifier")
plt.draw();
plt.figure(3);
plt.plot(Jnll,'-',J01,'-');
plt.xlabel("epoch")
plt.title("Convergence of Surrogate Loss and Error Rate")
```

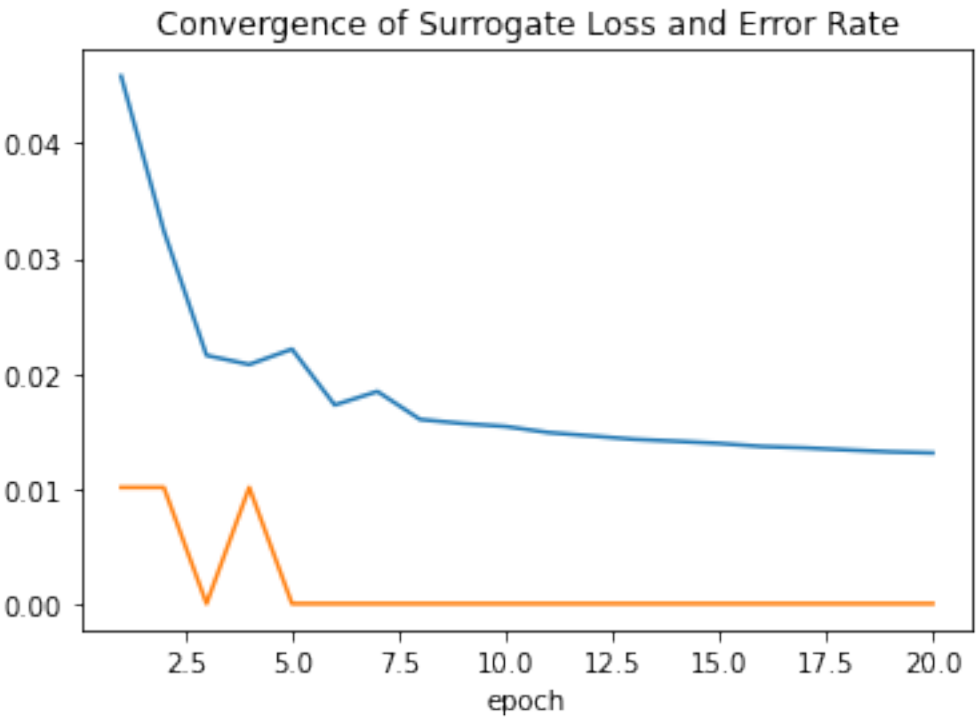
2.7 Problem 1.7

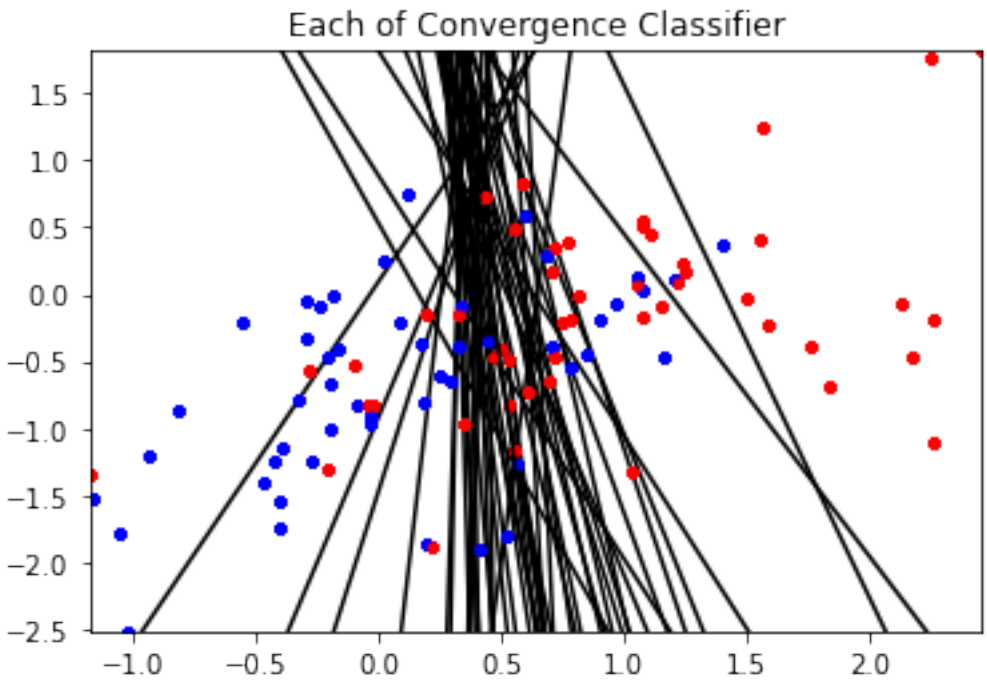
[5]:
```
wts = np.array([0.5,1.,-0.25])
learner = logisticClassify2()
learner.theta = wts; # set the learner's parameters

#I changed the output of train function so that I am able to put the convergence␣
 ↪line on the same plot
learner.train(XA,YA,alpha=0,initStep=1)
print ("Final Theta of dataset A:", learner.theta)
learner.train(XB,YB,alpha=0,initStep=1)
print ("Final Theta of dataset B:", learner.theta)
```



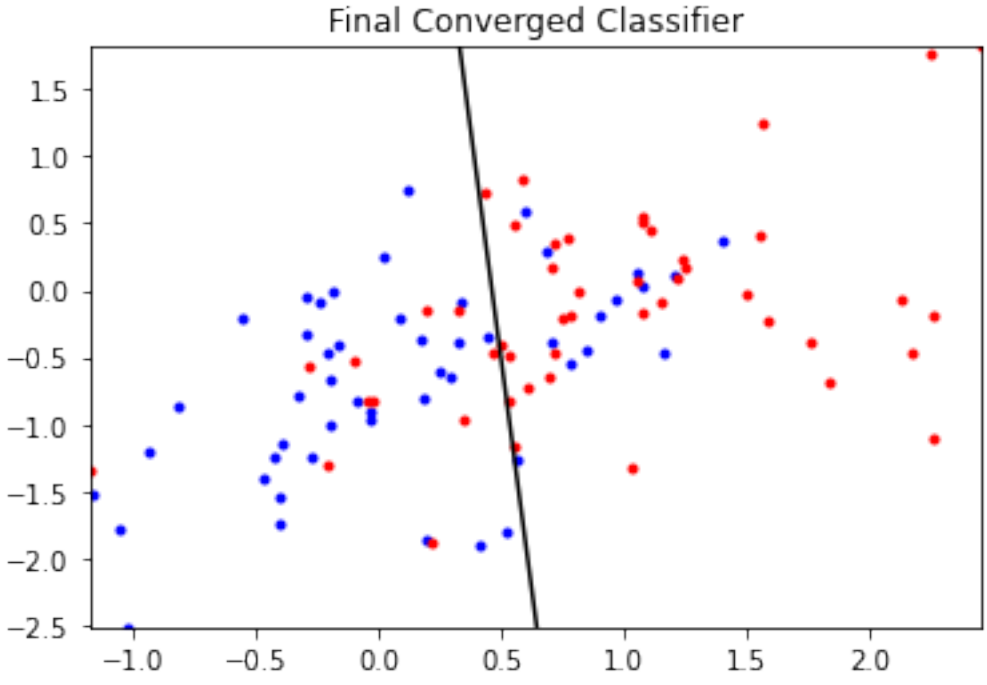

```
Final Theta of dataset A: [ 5.37920701  8.86590312 -4.55706668]
```

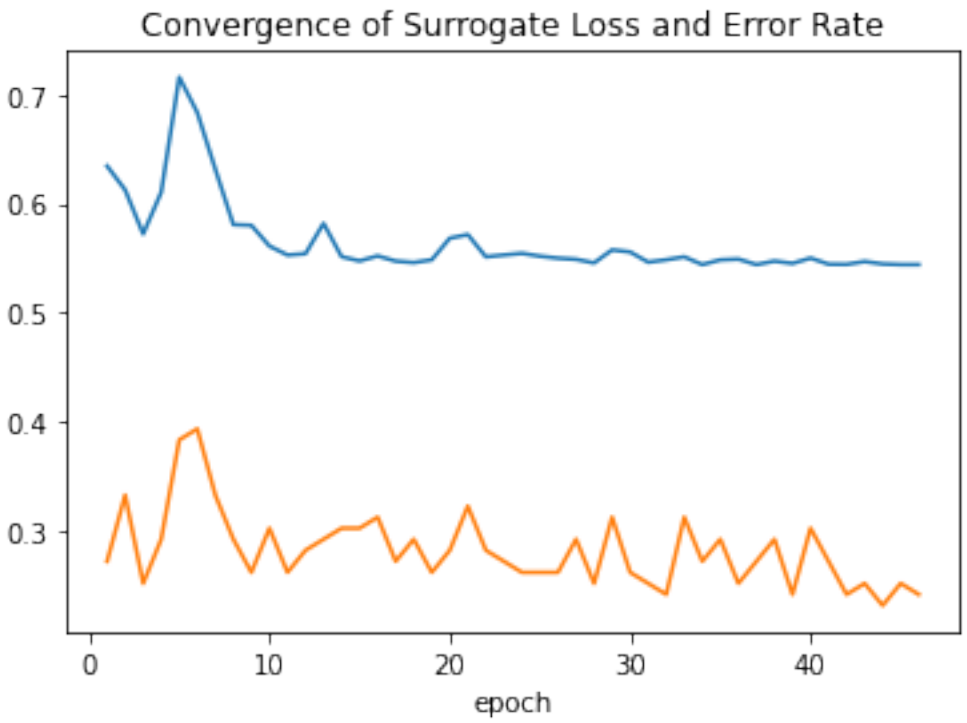

Final Converged Classifier
3
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0


Convergence of Surrogate Loss and Error Rate
0.04
0.03
0.02
0.01
0.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch



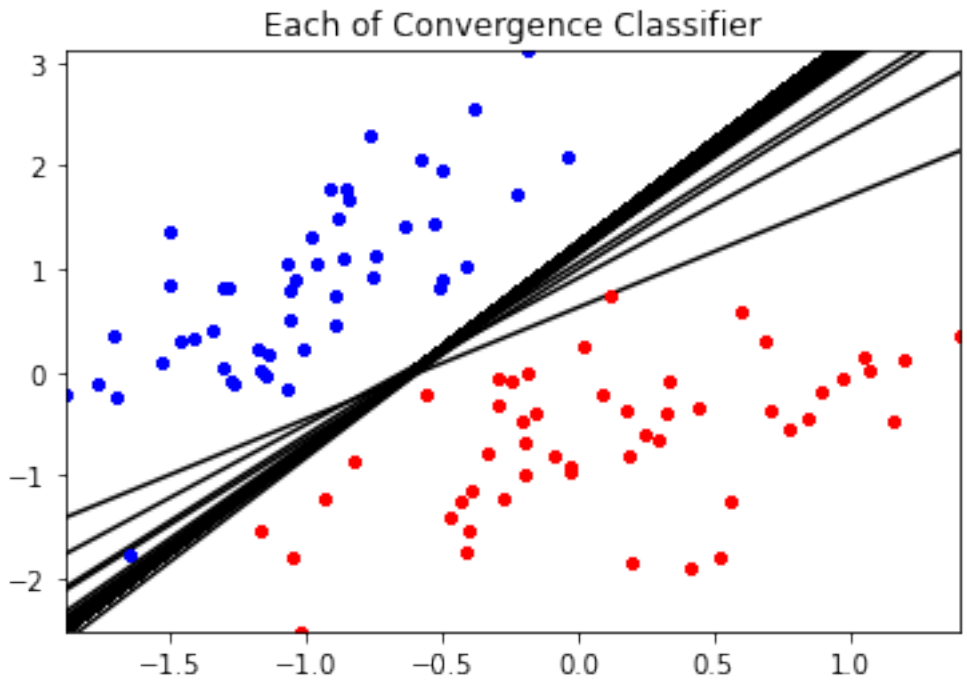

```
Final Theta of dataset B: [-0.75622342  1.63242059  0.11871738]
```



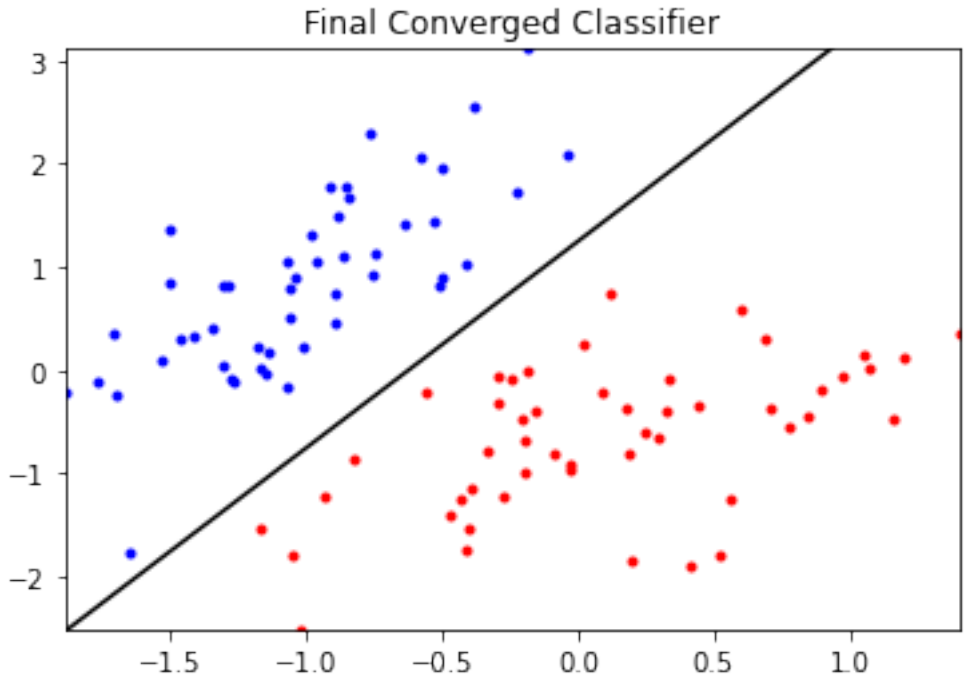


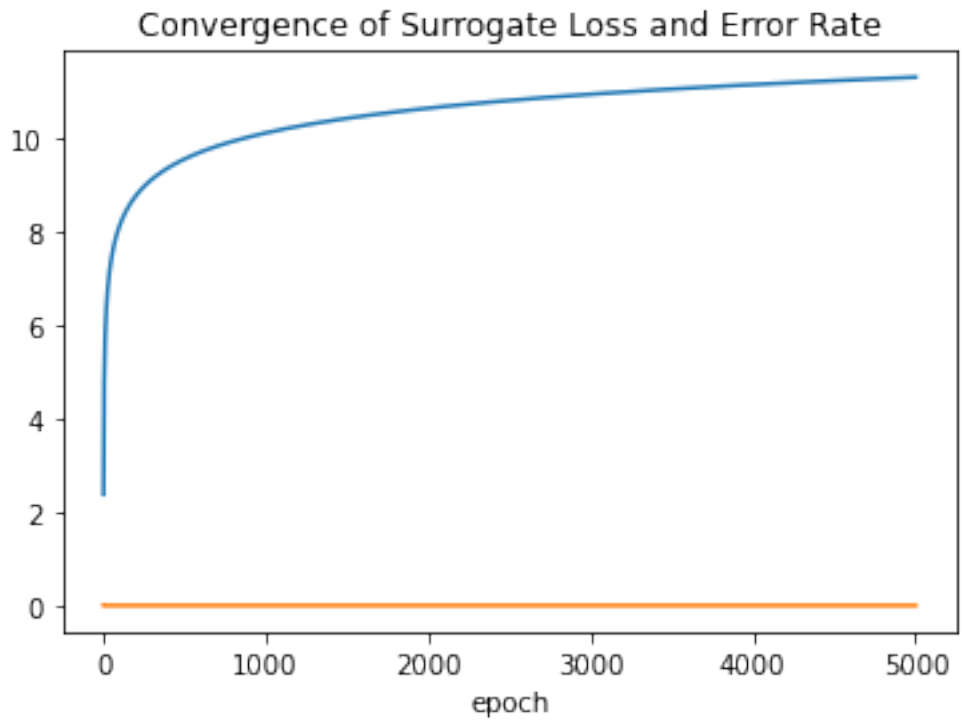
2.8 Problem 1.8

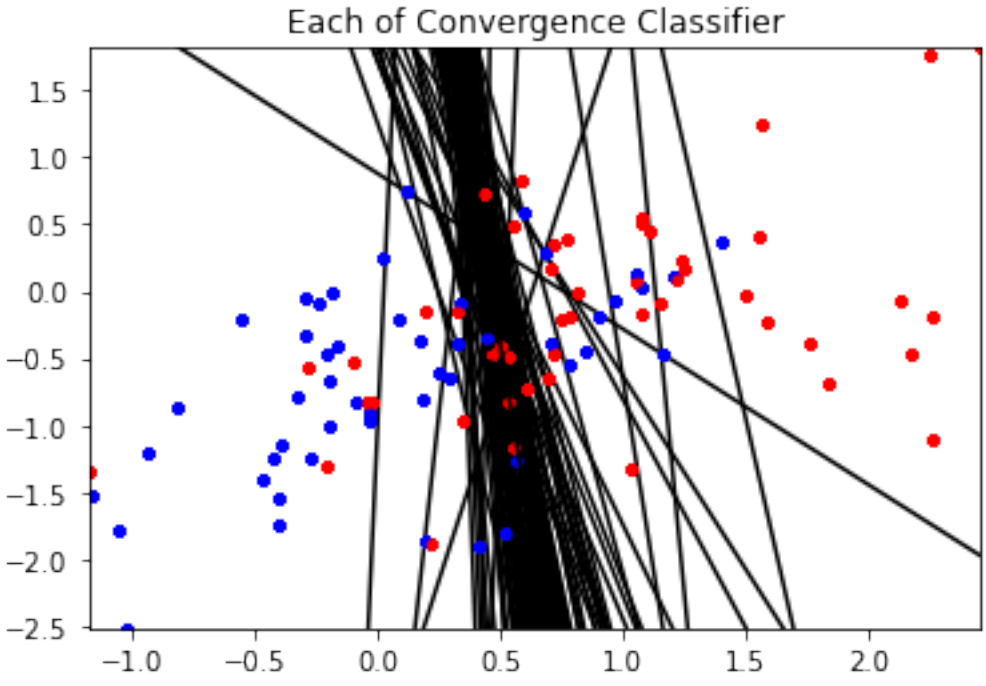
[10]:
```
wts = np.array([0.5,1.,-0.25])
learner = logisticClassify2()
learner.theta = wts; # set the learner's parameters

learner.train(XA,YA,alpha=5,initStep=1)
print ("Final Theta of dataset A after adjusting alpha:", learner.theta)
learner.train(XB,YB,alpha=5,initStep=1)
print ("Final Theta of dataset B after adjusting alpha:", learner.theta)
```




```
Final Theta of dataset A after adjusting alpha: [ 7.28691427 11.67676128
-5.83346328]
```

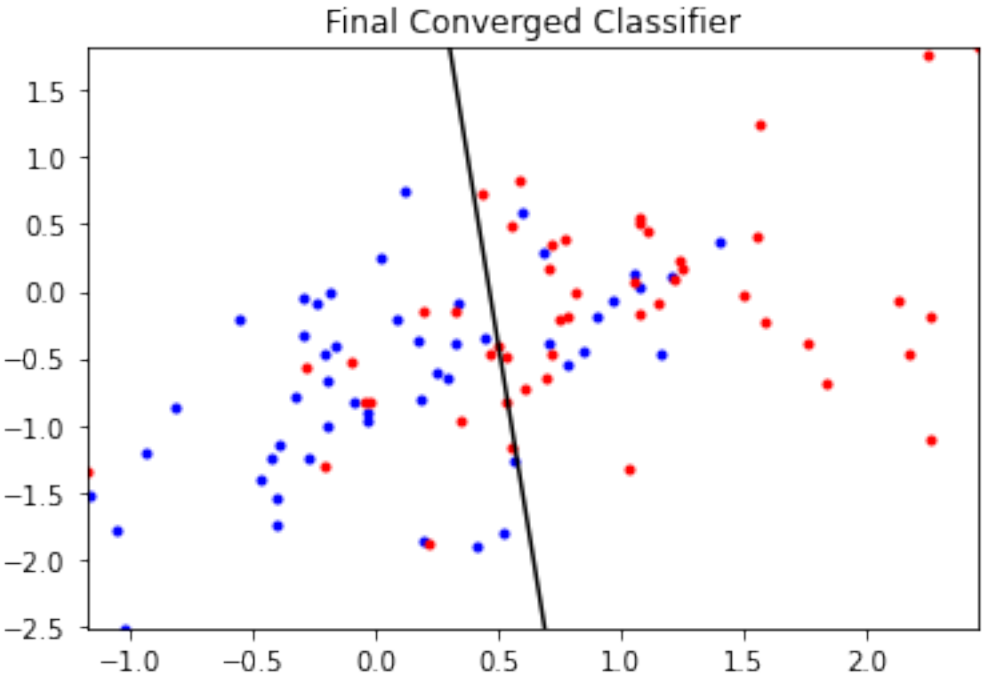


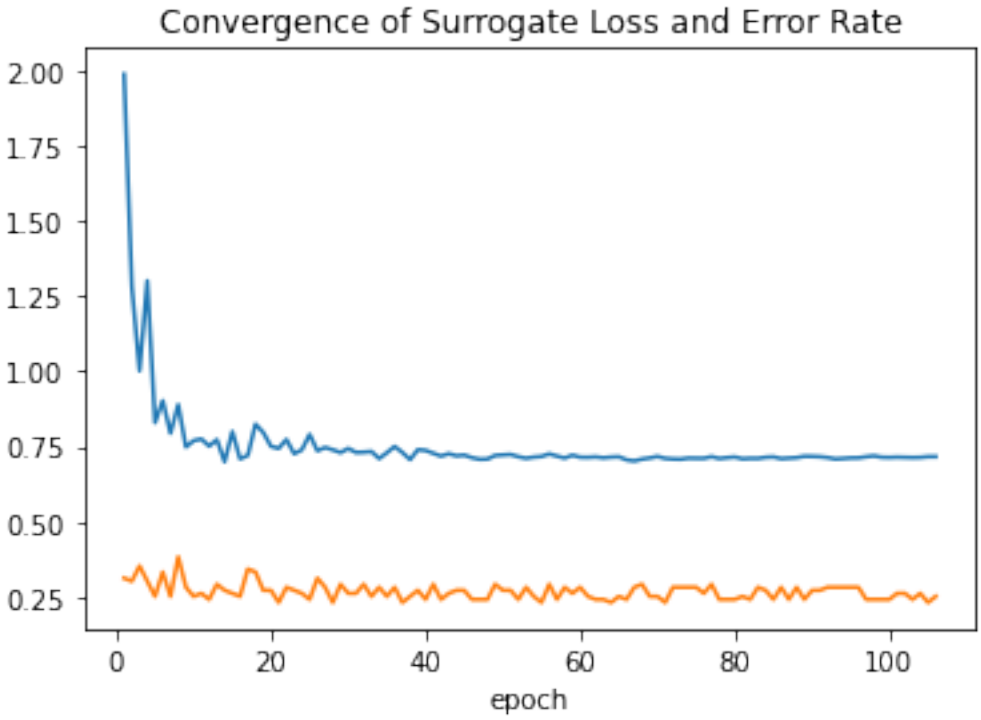





```
Final Theta of dataset B after adjusting alpha: [-0.77718122  1.66930632
```

0.14952328]


Final Converged Classifier
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Convergence of Surrogate Loss and Error Rate
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
20
40
60
80
100
epoch

By adding the regularization term to surrogate loss function, the gradient logistic regression will be controlled by alpha if it's relative large like alpha = 5 comparing to the normal error rate. In this case, we can observe from the plot that for dataset (XA,YA), the error rate starts at the high point and keep increasing along with the gradient changing, this is because the alpha term is dominating the whole gradient process, and due to the distribution of set A, which the severe change will cause the greater error rate since alpha makes model can not be trained too much. For dataset(XB,YB), the error rate begins with high value but then drop and converge rapidly, this is because the distribution of set B is more random, the term alpha will not cause too many difference here. To conclude, L2 regularization can prevent overfitting problem for over training on training data, however if the value of alpha is too large, it will cause underfitting problem.

2.9 Problem 2

1. T(a + bx1) : (a)(b) can be shattered by this learner since the learner is a vertical line which can only classify 0 and 1, (c)(d) can not be shattered since this learner is not able to separate three points into all of the possible 8 conditions.
2. T((a * b)x1 + (c / a)x2) : (a)(b) can also be shattered by this learner, the reasons are just like learner 1, it's a linear line control by factor (a * b) and (c / b), (c)(d) can not be shattered by learner 2 because the lack of constant term in this learner, so that the slope will remain the same all the time, there is no chance for it to shatter (c)(d).
3. T((x1 - a)^2 + (x2 - b)^2 + c) : This learner can shatter (a)(b)(c), for (a)(b), the circle learner can separate each points from another, for (c), it can change the central point of circle and change radius by factor c to classify each group. However in (d), points (4,8) and (6,4) might be hard for circle to classify them together from other two points, it should be done by oval rather than circle learner.
4. T(a + bx1 + cx2) * T(d + bx1 + cx2) : This learner can shatter(a)(b)(c)(d), it's not an issue for a complex model to shatter (a)(b)(c) in this case, for(d), the high dimension of this learner makes it possible to shatter all of the four points.

3 Statement of Collaboration

I finished this assignment by myself, only discussing the concept of train function with Josh Ho, and VC Dimension definition with Will Schallock.