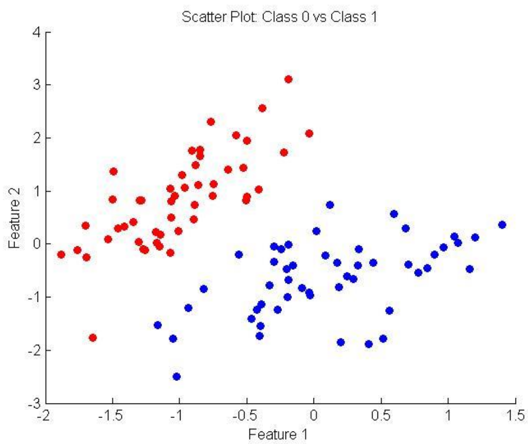
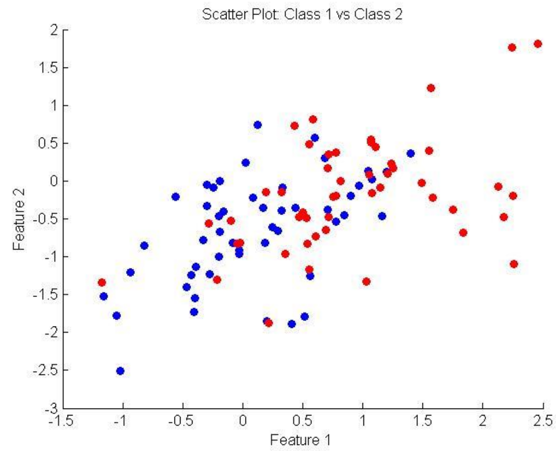
Problem 1

PART A: Show two classes in a scatter plot and verify that one is linearly separable while the other is not.





According to the plots above, “Class 0 vs Class 1” is linearly separable while “Class 1 vs Class 2” is not.

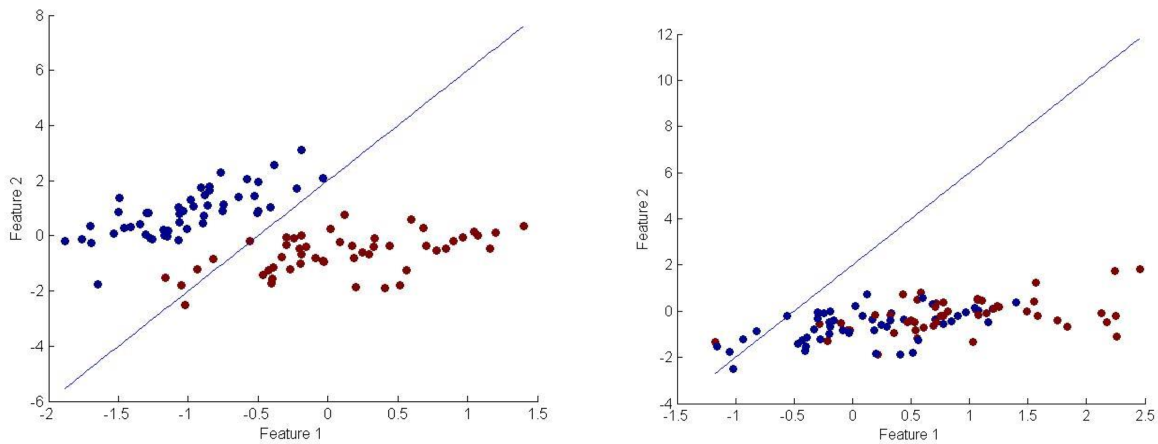
Code:

```
%Class 0 vs 1
figure (1)
XA_1 = X(Y==1,:);
XA_0 = X(Y==0,:);
hold on;
scatter(XA_1(:,1),XA_1(:,2),'b','filled')
scatter(XA_0(:,1),XA_0(:,2),'r','filled')
title('Scatter Plot: Class 0 vs Class 1')
  xlabel('Feature 1')
  ylabel('Feature 2')
hold off;

%Class 1 vs 2
figure (2)
XB_1 = X(Y==1,:);
XB_2 = X(Y==2,:);
hold on;
scatter(XB_1(:,1),XB_1(:,2),'b','filled')
scatter(XB_2(:,1),XB_2(:,2),'r','filled')
title('Scatter Plot: Class 1 vs Class 2')
  xlabel('Feature 1')
  ylabel('Feature 2')
hold off;
```

PART B: Write the function @logisticClassify2/plot2DLinear.m so that it plots the two classes of data in different colors, along with the decision boundary (a line). Boundary is defined as sign($0.5 + 1x_1 - 0.25x_2$).

For testing purposes, the following code is written to set up the testing.

```
learner = logisticClassify2();  % Create "blank" learner
learner = setClasses(learner, unique(YA)); % Define class labels using YA or
YB
wts = [0.5 1 -0.25]; % [theta0 theta1 theta2];
learner = setWeights(learner,wts); % Set the learner's parameters

plot2DLinear(learner, XA, YA)
plot2DLinear(learner, XB, YB)
```



The image on the left is the output plot when data set A is passed into “plot2DLinear” function while the image on the right shows the output plot when data set B is passed into the function.

Code:

```
  %%% TODO: Fill in the rest of this function...
  figure
  hold on;
  scatter(X(:,1),X(:,2), [], Y, 'filled');
  xlabel('Feature 1')
  ylabel('Feature 2')

  % Let's plot the decision boundry
  % Want to plot wts(0) + wts(1)*x(1) + wts(2)*x(2) = 0
  % wts(2)*x(2) = -(wts(0) + wts(1)*x(1))
  % x(2) = (-1/wts(2))*(wts(0) + wts(1)*x(1))

 x1 = min(X):0.025:max(X);
 x2 = (-1/obj.wts(3))*(obj.wts(1) + obj.wts(2)*x1);
 plot(x1,x2)
 hold off;
```

PART C: Complete the predict.m function to make predictions for your linear classifier. Verify that the function works by computing & reporting the error rate of the classifer in the previous part on both data sets A and B. (The error rate on data set A should be around 0.0505.)

Code:

```
% STEP (1) make predictions based on the sign of wts(1) + wts(2)*x(:,1)
+ ...
predict_sign = sign(obj.wts(1) + obj.wts(2)*Xte(:,1) + obj.wts(3)*Xte(:,2));
% STEP (2) convert predictions to saved classes: Yte = obj.classes( [1 or
2] );
```

```
Y_positive = find(predict_sign == 1);
Y_negative = find(predict_sign == -1);

[n,d]=size(Xte);
Yte = zeros (n,1);

Yte(Y_positive) = obj.classes(2);
Yte(Y_negative) = obj.classes(1);

Yte;
```

"PlotClassify2D" processing Data Set A and Data Set B:

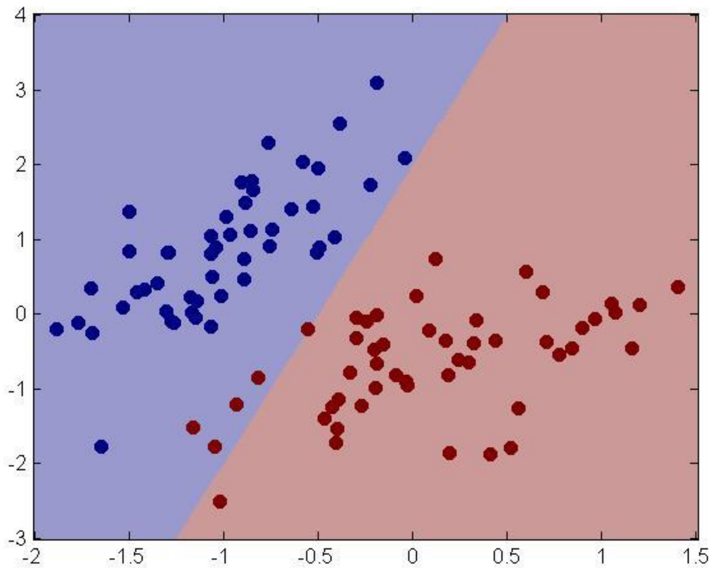


Figure 1 PlotClassify2D(XA) Data Set A

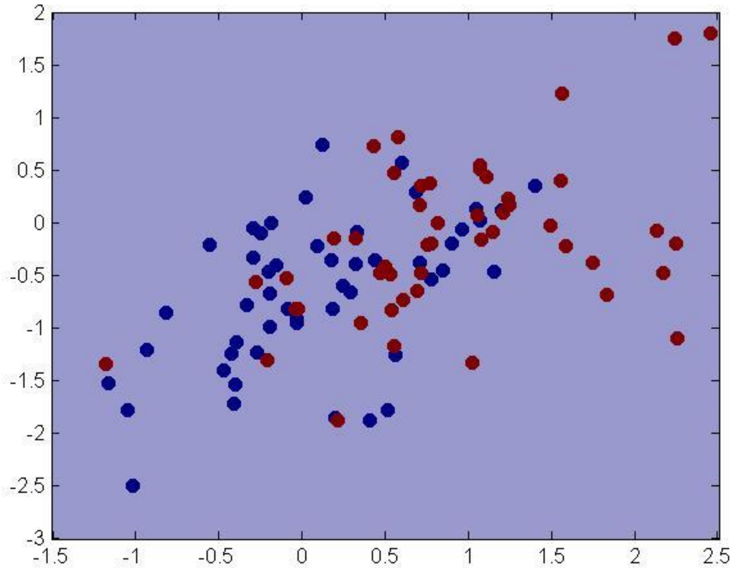


Figure 2 PlotClassify2D(XB) Data Set B

Error Calculation:

error_A =

0.0505

error_B =

0.5455

The calculated error of data set A matches with the error given in the problem; therefore, I believe the predict.m function is working properly.

PART D:

Linear Response of Perceptron : $Z = \theta x^{(i)}$

$\sigma \Rightarrow$ Standard Logistic function.

$$\sigma(z) = (1 + \exp(-z))^{-1}$$

↳ Regularized Logistic Negative log likelihood loss for a single data point j

$$\Rightarrow J_j(\theta) = -y^{(j)} \log \sigma(\theta x^{(j)T}) - (1 - y^{(j)}) \log(1 - \sigma(\theta x^{(j)T})) + \alpha \sum_i \theta_i^2$$

$y^{(j)} \Rightarrow$ either 0 or 1.

→ Derive the gradient of the regularized negative log likelihood J_j for logistic regression.

$$\frac{\partial J_j(\theta)}{\partial \underline{\theta}} = \frac{\partial}{\partial \underline{\theta}}\left[-y^{(j)} \log \sigma(\underline{\theta} x^{(j)T})\right] - (1-y^{(j)}) \frac{\partial}{\partial \underline{\theta}}\left[\log\left(1 - \sigma(\underline{\theta} x^{(j)T})\right)\right] + \alpha \frac{\partial}{\partial \underline{\theta}}\left[\sum_i \theta_i^2\right]$$

$$= -y^{(j)}\left[\frac{1}{\sigma(\theta x^{(j)T})} \frac{\partial}{\partial \theta}\left(\sigma(\underline{\theta} \cdot x^{(j)T})\right) x^{(j)}\right]$$

$$-(1-y^{(j)})\left[\frac{1}{1-\sigma(\underline{\theta} x^{(j)T})} \frac{\partial}{\partial \underline{\theta}}\left(1 - \sigma(\underline{\theta} x^{(j)T}\right) x^{(j)}\right] + 2\alpha\theta$$

Let $\underline{\theta} = [\theta_0 \ \theta_1 \ \theta_2 \cdots \theta_K]$ $\quad x^{(j)T} = \begin{bmatrix} x_0^{(j)} \\ x_1^{(j)} \\ \vdots \end{bmatrix}$

$$\underline{\theta} \cdot x^{(j)T} = \theta_0 x_0^{(j)} + \theta_1 x_1^{(j)} + \cdots$$

$$\frac{\partial}{\partial\theta}\left[\sigma(\underline{\theta}\cdot x^{(j)T})\right]$$

$$\to \quad \sigma(\underline{\theta}\cdot x^{(j)T}) = \frac{1}{1+\exp(-\underline{\theta}x^{(j)T})}$$

$$\frac{\partial}{\partial\theta}\left(1+\exp(-\underline{\theta}x^{(j)T})\right)$$

$$= -\left(1+\exp(-\underline{\theta}x^{(j)T})\right)^{-2}\cdot\exp(-\underline{\theta}x^{(j)T})\cdot\frac{\partial}{\partial\theta}\left(-\underline{\theta}x^{(j)T}\right)$$

$$= \left(1+\exp(-\underline{\theta}x^{(j)T})\right)^{-2}\cdot\exp(-\underline{\theta}x^{(j)T})\cdot x^{(j)}$$

$$\frac{\partial}{\partial\theta}\left[\left(1-\sigma(\underline{\theta}x^{(j)T})\right)\right] = \frac{\partial}{\partial\theta} - \frac{\partial}{\partial\theta}\left[\sigma(\underline{\theta}x^{(j)T})\right]$$

$$= -\left[\left(1+\exp(-\underline{\theta}x^{(j)T})\right)^{-2}\cdot\exp(-\underline{\theta}x^{(j)T})\,x^{(j)}\right]$$

$$\Rightarrow \frac{\partial J_j(\theta)}{\partial\theta} = -y^{(j)}\left(1+\exp(-\underline{\theta}x^{(j)T})\right)\left[\left(1+\exp(-\underline{\theta}x^{(j)T})\right)^{-2}\exp(-\underline{\theta}x^{(j)T})\cdot x^{(j)}\right]$$

$$+(1-y^{(j)})\underbrace{\frac{1}{1-\frac{1}{1+\exp(-\underline{\theta}x^{(j)T})}}}\left[\left(1+\exp(-\underline{\theta}x^{(j)T})\right)^{-2}\exp(-\underline{\theta}x^{(j)T})\cdot x^{(j)}\right]$$

$$+2\alpha\theta \qquad \hookrightarrow \frac{1+\exp(-\underline{\theta}x^{(j)T})}{\cancel{1}+\exp(-\underline{\theta}x^{(j)T})\cancel{-1}}$$

$$\Rightarrow \frac{\partial J_j(\theta)}{\partial\theta} = -y^{(j)}\frac{\exp(-\underline{\theta}x^{(j)T})}{1+\exp(-\underline{\theta}x^{(j)T})}x^{(j)} + (1-y^{(j)})\frac{1}{1+\exp(-\underline{\theta}x^{(j)T})}x^{(j)}$$

$$+2\alpha\theta.$$

PART E:

```
% Training loop (SGD):
iter=1; Jsur=zeros(1,stopIter); J01=zeros(1,stopIter); done=0;
while (~done)
  step = stepsize/iter;               % update step-size and evaluate current loss values
  %%% TODO: compute surrogate (neg log likelihood) loss
  for i=1:n
    z = obj.wts*X1(i,:)';
```

```
    sigma = 1/(1+exp(-z));
    Jsur(iter) = Jsur(iter)-Y(i,:)*log(sigma)-(1-Y(i,:))*(log(1-sigma))+
reg*(sum((obj.wts).^2));   %%% TODO: compute surrogate (neg log likelihood)
loss
  end
  J01(iter) = err(obj,X,Yin);



for j=1:n,
    % Compute linear responses and activation for data point j
    %%% TODO ^^^
       z = obj.wts*X1(j,:)';
      sigma = 1/(1+exp(-z));
    % Compute gradient:
    %%% TODO ^^^
    grad = -Y(j,:)*(1-sigma)*X1(j,:)+(1-Y(j,:))*sigma*X1(j,:)+ 2*reg*obj.wts
    %grad = -Y(j,:)*sigma*exp(-z)*X1(j,:)+(1-Y(j,:))*sigma*X1(j,:)+
2*reg*obj.wts;
    obj.wts = obj.wts - step * grad;      % take a step down the gradient
  end;

  done = false;

  %%% TODO: Check for stopping conditions
  norm(obj.wts-wtsold)
    if (stopIter == iter || norm(obj.wts-wtsold)<stopTol)
      done = true;
  end;

  wtsold = obj.wts;
  iter = iter + 1;
end;
```

PART F:

I run the train function with Data Set A, train function allows it to update the weight for better classification. Numbers shows the gradient function value at a specific data point and iteration and the changes of the gradient function. The train function stop when the changes of the weight function drop less than 0.02.

Testing Code:

```
% % Part E & F: Train Function
    % Please comment or uncomment either one for data set A and data set B
    train(learner, XA, YA, 'stopiter', 200, 'stoptol', 0.02);
    %train(learner, XB, YB, 'stopiter', 200, 'stoptol', 0.02);
```

Result:

grad =

0.0661 -0.0328 0.0590

ans =

0.0196

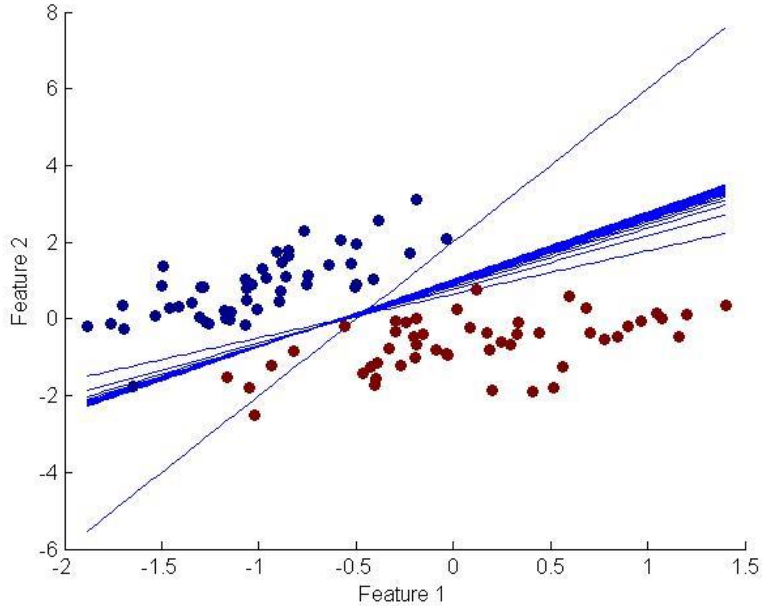


Figure 3 TrainFucntion (XA)

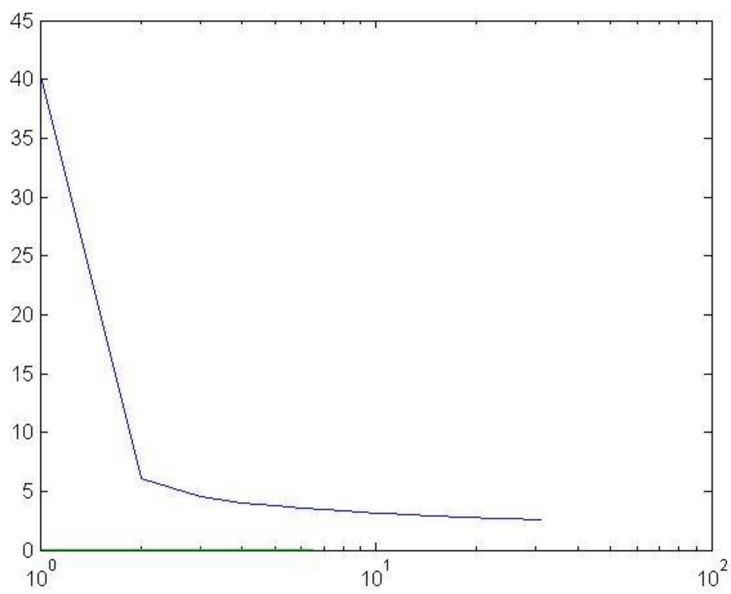


Figure 4 TrainFunction_Change of Gradient Function (XA)

For Data Set B, it is not a linearly separable data set. Therefore, the train function can not calculate a classifier that separate Data Set B no matter how small does "`norm(obj.wts-wtsold)`" (i.e. Change of weight function). In this testing set up, the stoptol is set to 0.001 in the test case with 200 iteration.

Test Code:

```
train(learner, XB, YB, 'stopiter', 200, 'stoptol', 0.001);
```

Result:

grad =

-0.5066 -0.2383 0.2389

ans =

9.8418e-04

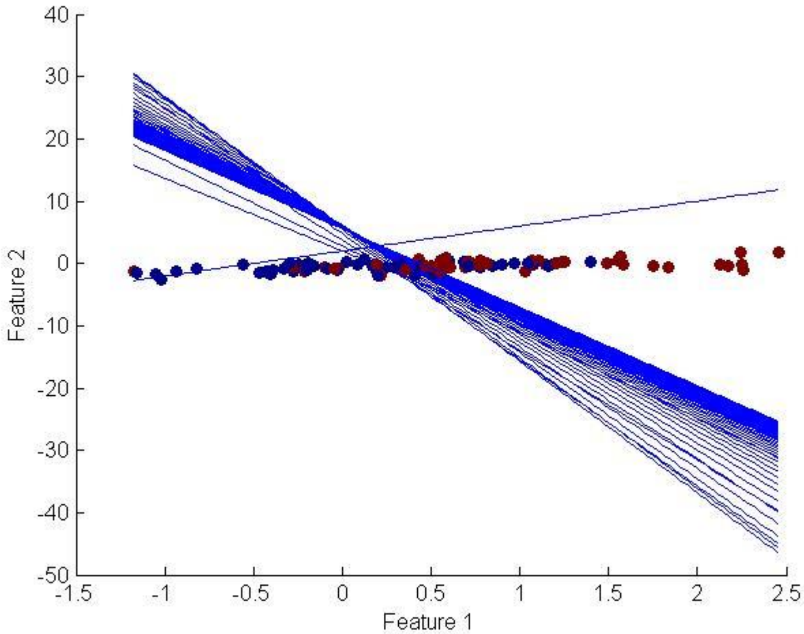


Figure 5 TestFunction(XB)

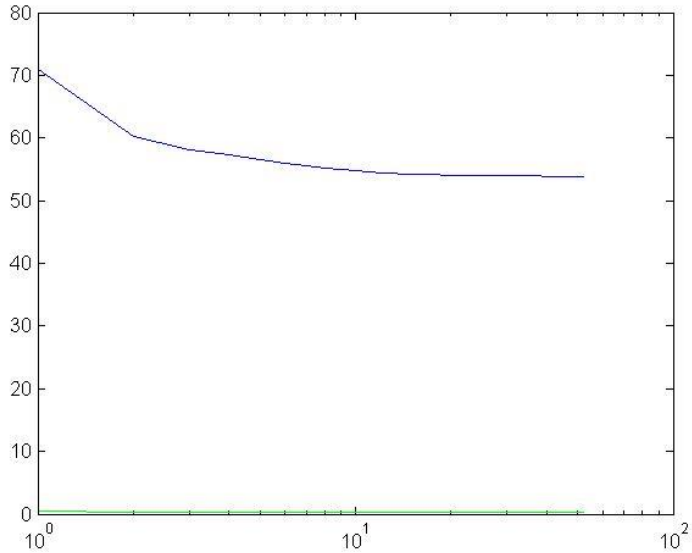


Figure 6 TestFunction Error (XB)

Problem 2

Determine which of the four set of Data points can be shatter by each learner. Explanation / Justification and use the result to guess the VC dimension of the classifier

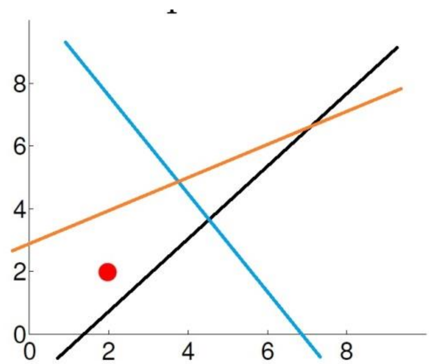


Figure 7: Data Set 1

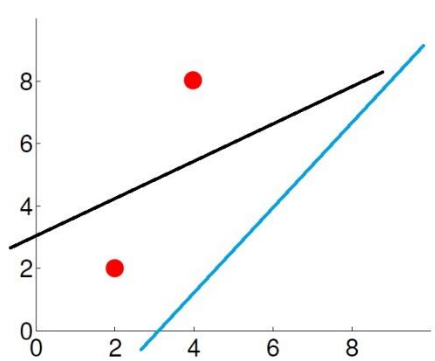


Figure 8: Data Set 2

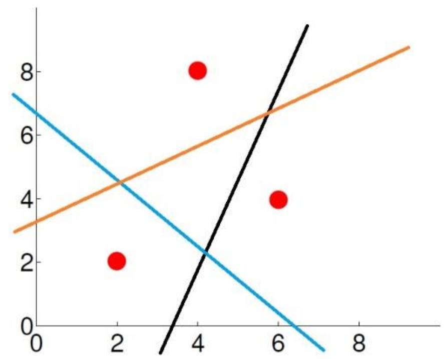


Figure 9: Data Set 3

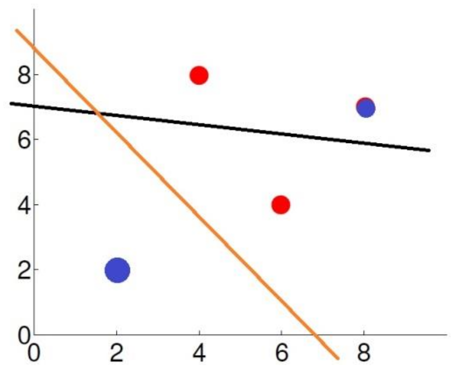


Figure 10: Data Set 4

As shown above, Data set 1 to 3 can be shattered but not Data Set 4.

a) T(a+bx1)
   Function: a+bx1 is a linear function; therefore, it can shatter Data set 1 to 3 as shown above. In other words, VC dim = 3 as it can at most shatter 3 data points.
b) T((x1-a)^2 + (x2-b)^2 + c)
   Function: (x1-a)^2 + (x2-b)^2 + c is a circular function center at point (a,b). The VC dim for this function is 3. It is because if the circles locates at (4,8) with increasing radius, it cannot includes at (4,8) and (2,2) while not including (4,6).
c) T((a*b)x1 + (c/a)x2)
   Function: (a*b)x1 + (c/a)x2 is also a linear function; therefore, the VC dim is the same as the function in part A. VC dim = 3