Week 3 **Varad Meru**
CS 273a - Introduction to Machine Learning (Winter '15)[*] Student # 26648958
Prof. Alex Ihler Due Date: 01/27/2015

Homework 3[†‡]

Problem 1: Perceptron and Logistic Regression

(a) The scatter plot of the data is generated by code written in Listing 1. The scatter plots generated can be seen in Figure 1. From visual inspection, it can be seen that the data is linearly separable.

Listing 1: Plotting the scatter plot

```
iris=load('data/iris.txt');      % load the text file
Y = iris(:,end);                 % target value is last column
X = iris(:,1:end-1);             % features are other columns
features = char('Sepal length','Sepal width','Petal length','Petal ...
    width','Species');
features_short = char('SL','SW','PL','PW','SP');

% Vertically Splitting the dataset. Keeping the first 2 columns for the
% perceptron.
Xs = X(:,1:2);

% Shuffling and Rescaling X
[Xs Y] = shuffleData(Xs,Y);
Xs   = rescale(Xs);

% get class 0 vs 1
XA = Xs(Y<2,:);
YA=Y(Y<2);
% get class 1 vs 2
XB = Xs(Y>0,:);
YB=Y(Y>0);

%% Problem a
h=figure; scatter(XA(:,1), XA(:,2), 50, YA,'filled'); ...
    saveas(h,'scatter-classes1.jpg','jpg'); h=figure; scatter(XB(:,1), ...
    XB(:,2), 50, YB,'filled'); saveas(h,'scatter-classes2.jpg','jpg');
```

(b) The `@logisticClassify2/plot2DLinear` is modified to present the decision boundary and can be seen in Listing 2. The generated scatter plots can be seen in Figure 2.

Listing 2: The `plot2DLinear()`

```
function plot2DLinear(obj, X, Y)
[n,d] = size(X);
if (d≠2) error('Sorry -- plot2DLogistic only works on 2D data...'); end;

%%% TODO: Fill in the rest of this function...
wts = getWeights(obj);
% LaTeX does not display the 'at' symbol
f = (x1, x2) wts(1) + wts(2)*x1 +wts(3)*x2;

scatter(X(:,1),X(:,2),50,Y,'filled');
hold on;
ezplot(f,[-3,3])
hold off;
```

[*] Website: http://sli.ics.uci.edu/Classes/2015W-273a
[†] Questions available at http://sli.ics.uci.edu/Classes/2015W-273a?action=download&upname=HW3.pdf
[‡] All the figures and listing numbers are auto-referred.

Listing 3: plotting using `plot2DLinear()`

```
%% Problem b
learner = logisticClassify2();
learner=setClasses(learner, unique(YA));
wts = [.5 1 -.25];
learner=setWeights(learner, wts);

h=figure;
plot2DLinear(learner,XA,YA);
saveas(h,'log1.jpg','jpg');

h=figure;
plot2DLinear(learner,XB,YB);
saveas(h,'log2.jpg','jpg');
```

(c) The `predict()` function, given in Listing 4, would find the sign of the passed data points based on the function and weights assigned.

Listing 4: `predict()` function

```
function Yte = predict(obj,Xte)
% Yhat = predict(obj, X)  : make predictions on test data X
% (1) make predictions based on the sign of wts(1) + wts(2)*x(:,1) + ...
% (2) convert predictions to saved classes: Yte = obj.classes( [1 or 2] );
wts = getWeights(obj);
yhat = zeros(size(Xte,1),1);

% LaTeX does not display the 'at' symbol
f = (x1, x2) wts(1) + wts(2)*x1 + wts(3)*x2;

for i=1:size(Xte,1);
    x = sign(f(Xte(i,1),Xte(i,2)));
    yhat(i) = obj.classes(ceil((x+3)/2));
end;

Yte = yhat;
```

The `predict()` function is used to predict the possible classes and the error is computed using `errorTrain()`. The code can be seen in Listing 5, along with the errors for the `XA` and `XB` sections of the data.

Listing 5: Train and Test Error using the `predict()`

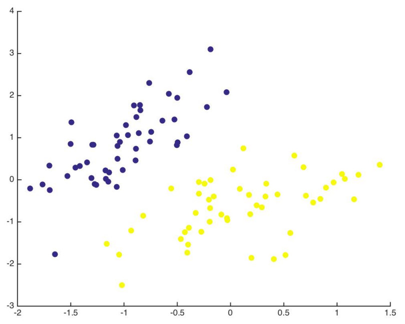
```
%% Problem c
yte = predict(learner,XA);
error = errorTrain(YA,yte);
% error = 0.0505
yte = predict(learner,XB);
error = errorTrain(YB,yte);
% error = 0.5455
```

(d) The derivation of the gradient is done my computing the $\frac{\partial J_j(\theta)}{\partial \theta_i}$ over the surrogate loss for point j given by $x^{(j)}, y^{(j)}$. The derivative is given Equation 1, where $\sigma(z) = (1 + exp(z))^{-1}$ and $z = \theta.x^{(j)T}$

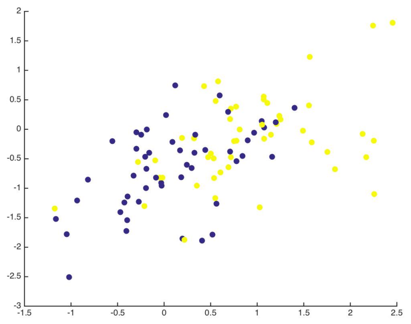
$$\frac{\partial J_j(\theta)}{\partial \theta_i} = x^{(j)}(\sigma(z) - y^{(j)}) + 2.\alpha.\theta_i \tag{1}$$

(e) I completed the `train()` function with the following changes

1. To compute the surrogate loss function at each iteration, I calculate the surrogate losses for each point and then compute the mean. (See Listing 6)

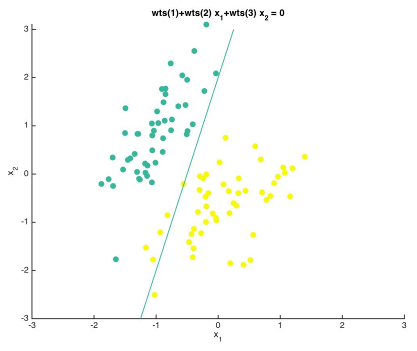


(a) Scatter Plot data of classes 0 and 1.

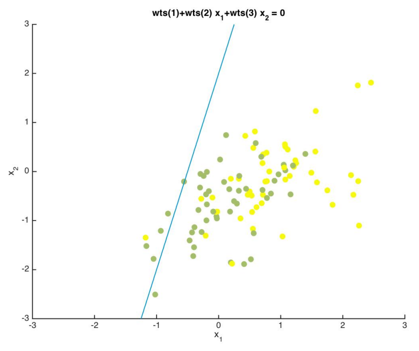


(b) Scatter Plot data of classes 1 and 2.

Figure 1: Scatter Plots of data



(a) Scatter Plot data of classes 0 and 1.



(b) Scatter Plot data of classes 1 and 2.

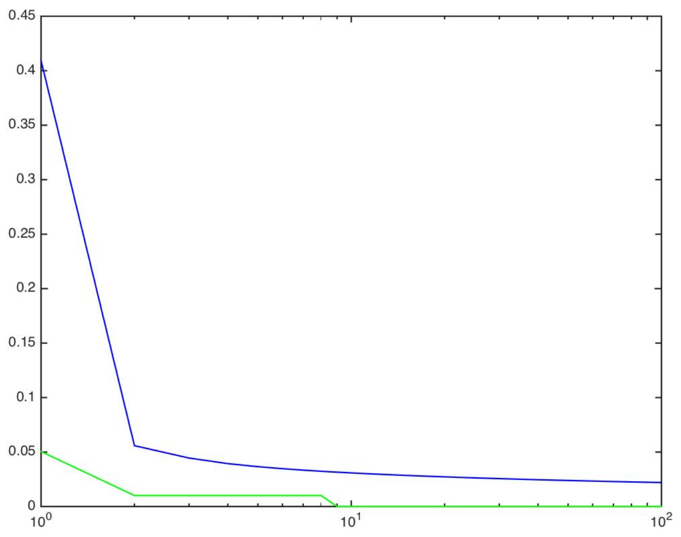
Figure 2: Scatter Plots of data with the Perceptron Decision boundary for weights = [0.5 1 -0.25]

2. Computing the gradient and predictions of data point $x^{(i)}, y^{(i)}$ is done using the `logistic()` function provided (See Listing 6).
3. The gradient step derived in the 1(d) part of this question is computed in this step. This helps us to find the movement of the solution and direction required to reach near the optima (See Listing 6).
4. The stopping condition is calculated and enforced at the end of the loop. It can be seen in Listing 6.

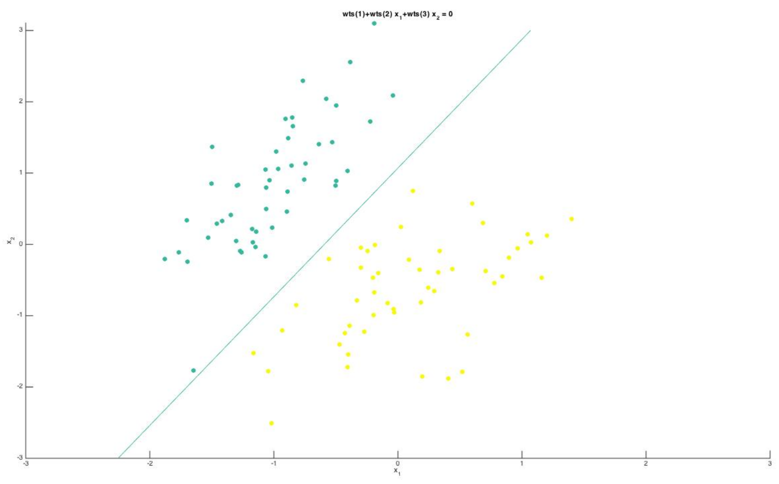
Listing 6: `train()` function snippet

```
while (¬done)
  step = stepsize/iter;                % update step-size and evaluate ...
      current loss values
  %Jsur(iter) = inf;   %%% TODO: compute surrogate (neg log ...
      likelihood) loss
  Jsur(iter) =mean( - Y .* log(logistic(obj, X)) - (1 - Y) .* log(1 - ...
      logistic(obj, X)) + reg * obj.wts * obj.wts');
  J01(iter) = err(obj,X,Yin);

  if (plotFlag), switch d,             % Plots to help with visualization
    case 1, fig(2); plot1DLinear(obj,X,Yin);  %  for 1D data we can ...
        display the data and the function
    case 2, fig(2); plot2DLinear(obj,X,Yin);  %  for 2D data, just the ...
        data and decision boundary
    otherwise, % no plot for higher dimensions... %  higher dimensions ...
        visualization is hard
  end; end;
  fig(1); semilogx(1:iter, Jsur(1:iter),'b-',1:iter,J01(1:iter),'g-'); ...
      drawnow;

  for j=1:n,
    % Compute linear responses and activation for data point j
    y = logistic(obj,X(j,:));
    %%% TODO ^^^

    % Compute gradient:
    grad = X1(j,:) * (y - Y(j)) + 2 * reg * obj.wts;
    %%% TODO ^^

    obj.wts = obj.wts - step * grad;
    % take a step down the gradient

  end;

  JDelta =mean( - Y .* log(logistic(obj, X)) - (1 - Y) .* log(1 - ...
      logistic(obj, X)) + reg * obj.wts * obj.wts');
  if iter == stopIter || abs(JDelta - Jsur(iter)) < stopTol;
    done = true;
  end;
  %%% TODO: Check for stopping conditions

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

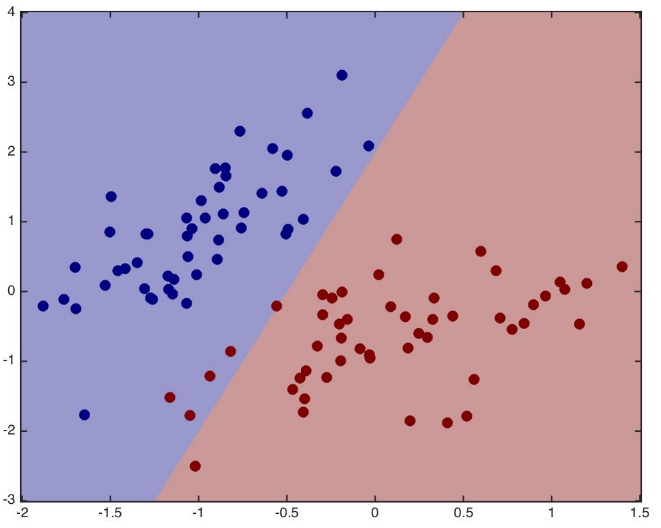
(f) With α set to zero, and the number of iterations = 100. The current step size is 1. The plots can be seen for two datasets `XA` and `XB` at Figure 3 and Figure 4



(a) Plot of convergence of the surrogate loss and the error rate

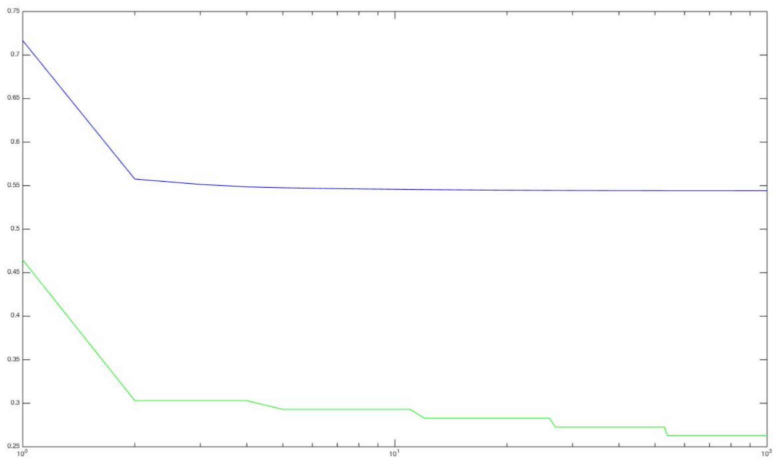


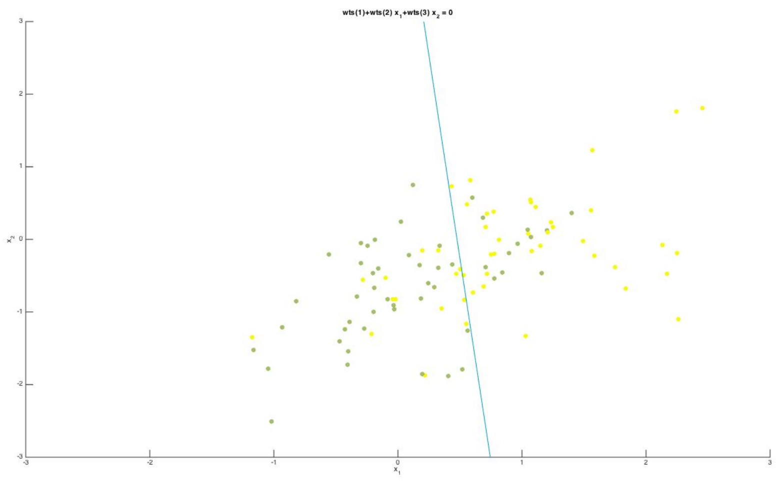
(b) Plot of the final converged classifier



(c) final converged classifier decision boundaries

Figure 3: For data XA, the error, surrogate loss plots and the classes.


(a) Plot of convergence of the surrogate loss and the error rate



(b) Plot of the final converged classifier

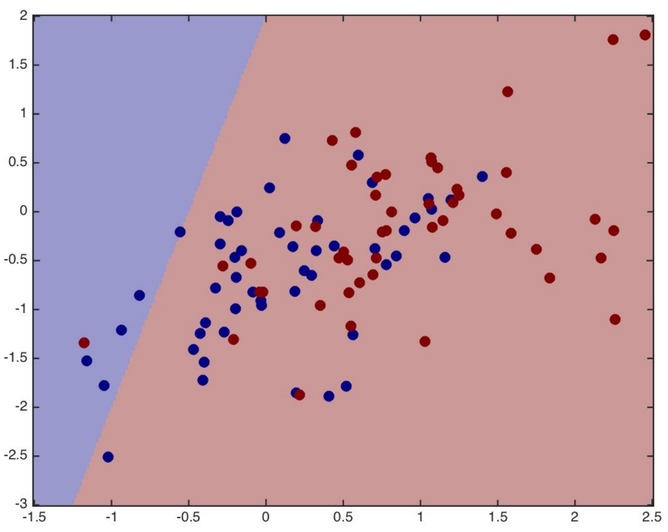

(c) final converged classifier decision boundaries

Figure 4: For data XB, the error, surrogate loss plots and the classes.

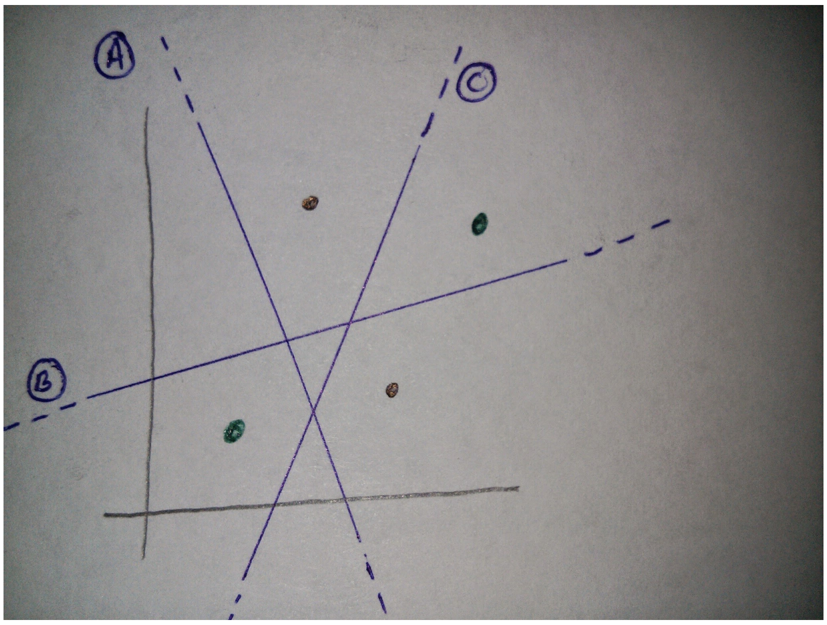


Figure 5: A, B, and C lines (representing the decision boundaries) trying to shatter 4 points with diagonal points with the same classes.

Problem 2: Shattering and VC Dimensions

(a) $T(a + bx_1)$: This function presents a line with the slope to be b and interception to be a. It would be able to shatter the cases (a), (b), and (c).

- It would be able to shatter the case (a) as a line can be on either sides of the single point and classify it in distinct classes.
- It would be able to shatter the case (b) as the line can be on either sides of the two points and in the middle to classify both the points as +1, or -1 or into two separate classes.
- It would be able to shatter the case (c) as the line can be on either sides of the three points. The advantage of interceptions and slope is that the line representing the decision boundary could be placed anywhere.

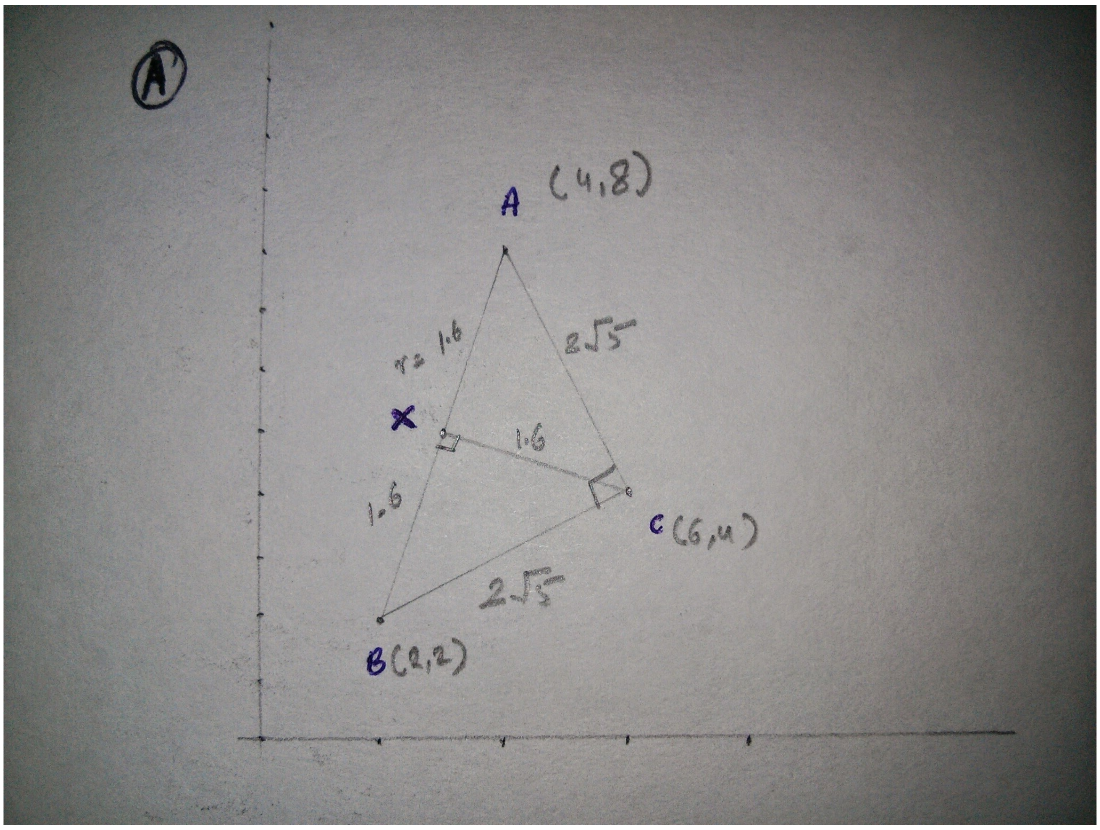
The (d) case would not be shattered by the line as it would not be able to divide the points where the classes of the diagonal points are same. It can be seen in Figure 5.

(b) $T((x_1 - a)^2 + (x_2 - b)^2 + c)$: This function presents a circle with coordinates of the center (a, b). It would be able to shatter the cases (a), (b).

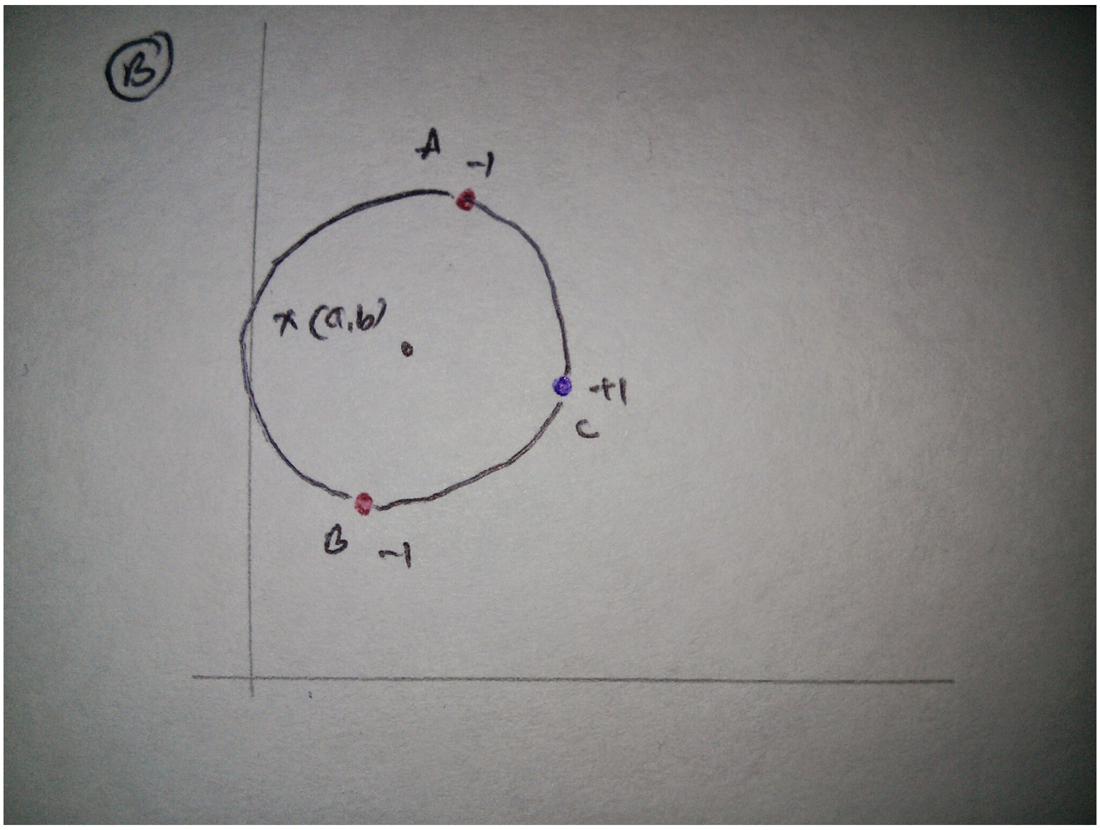
- It would be able to shatter the case (a) as either the lone point can be inside the circle or outside it, representing two different classes.
- It would be able to shatter the case (b) as the line as either the two points could be inside the circle, or outside the circle. Small circle can also be drawn to only contain individual points.

The (c) case would not be shattered by the circle as the placement of the three points would not allow a circle to be formed in a way that only two points come A and B come inside the circle and not C as can be seen in 6(b). The (d) case have similar problem, with two points.

(c) $T((a * b)x_1 + (c/a)x_2)$: This function shows lines passing through the origin. It would be able to shatter the cases (a), (b).



(a) Calculations of the distances computed amongst the points



(b) Trying to shatter placing a circle with A and B being two-ends of the diameter

Figure 6: Trying to shatter with circle. The calculations for the distances are shown in 6(a).

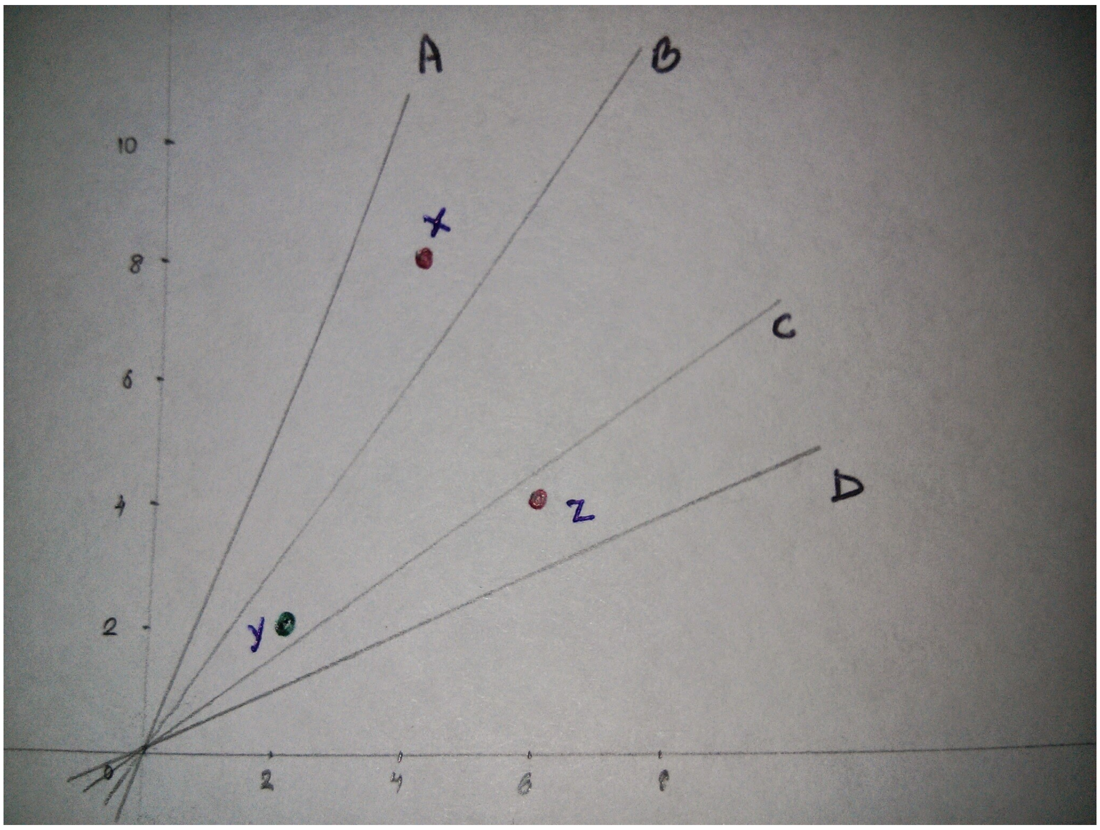


Figure 7: The lines A, B, C and D represent various function values.

- It would be able to shatter the case (a) as a line can be on either sides of the single point and classify it in distinct classes.
- It would be able to shatter the case (b) as the line can be on either sides of the two points and in the middle to classify both the points as +1, or -1 or into two separate classes.

The (c) case would not be shattered by the line passing through the origin as the lines would not be able to separate points with same classes on one side, such as X, and Z in the Figure 7. The configurations A, B, C and D show the various situations where the function fails to shatter the points. The same is true for the case (d) where the fourth point would not be shattered.