Homework 3

Zachary DeStefano, 15247592
CS 273A: Winter 2015
Due: January 27, 2015

Problem 1

Part a

This is the plot of class 0 versus class 1, which is linearly separable.

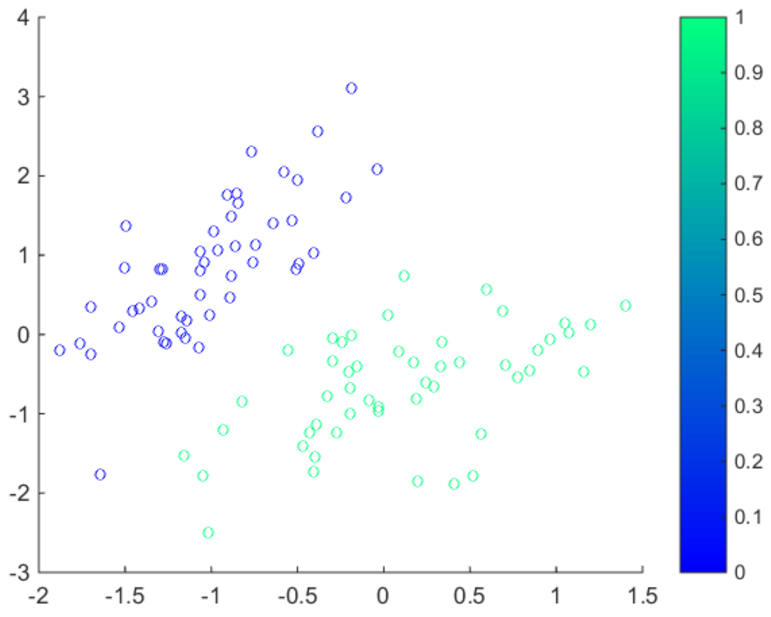

Figure 1: Class 0 and Class 1 Points

This is the plot of class 1 versus class 2, which is not linearly separable.

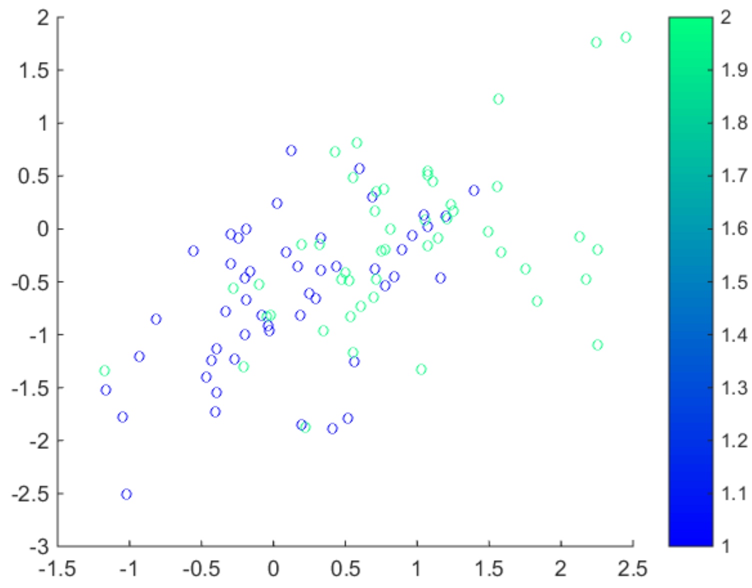

Figure 2: Class 1 and Class 2 Points

Here is the code to complete part a

```
iris=load('data/iris.txt'); % load the text file
X = iris(:,1:2); Y=iris(:,end); % get first two features
[X Y] = shuffleData(X,Y); % reorder randomly
X = rescale(X); % works much better for rescaled data

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

%%
%part A

%plot class 0 and class 1
figure
scatter(XA(:,1),XA(:,2),20,YA); %plot the data points
colormap winter;
colorbar

%plot class 1 and class 2
figure
scatter(XB(:,1),XB(:,2),20,YB); %plot the data points
colormap winter;
colorbar
```

Part b

Here is the plot of class 0 and class 1 along with the specified decision boundary

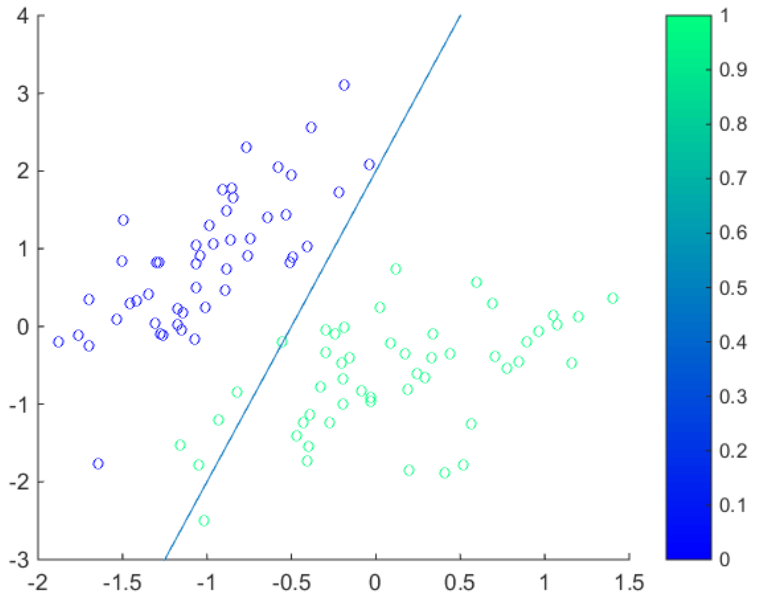

Figure 3: Class 0 and Class 1 points along with the sample decision boundary

Here is the plot of class 1 and class 2 along with that decision boundary

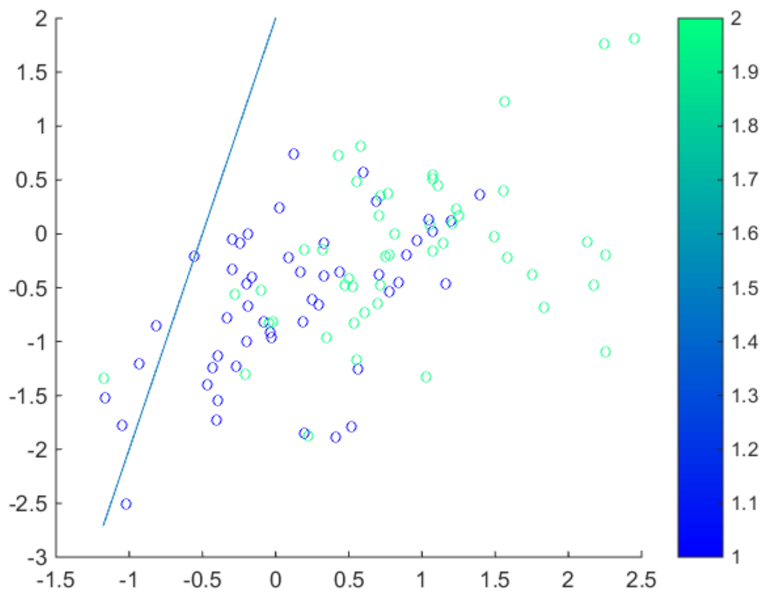

Figure 4: Class 1 and Class 2 points along with the sample decision boundary

Here is the code for plot2DLinear

```
function plot2DLinear(obj, X, Y)
% plot2DLinear(obj, X,Y)
%   plot a linear classifier (data and decision boundary) when features X are 2-dim
%   wts are 1x3,  wts(1)+wts(2)*X(1)+wts(3)*X(2)
%
  [n,d] = size(X);
  if (d~=2) error('Sorry -- plot2DLogistic only works on 2D data...'); end;

  weights = obj.wts;
  xs = min(X):0.05:max(X);
  ys = -(xs.*weights(2) + weights(1))/(weights(3));

  %figure
  scatter(X(:,1),X(:,2),20,Y); %plot the data points
  colormap winter;
  colorbar
  ax = axis;
  hold on
  plot(xs,ys); %plot the decision boundary line
  axis(ax) %make sure the axis still is just the points
  hold off
```

Here is the code to complete part b. It does rely on some of the variables from the part a code.

```
%%
%Part B

%weights to demo for part b
wts = [0.5 1 -0.25];

%define the two learners we will use
learnerA=logisticClassify2();
learnerB=logisticClassify2();

%set the class labels for our learners
learnerA=setClasses(learnerA, unique(YA));
learnerB=setClasses(learnerB, unique(YB));

%sets the weights for both learners
learnerA=setWeights(learnerA, wts);
learnerB=setWeights(learnerB, wts);

%plot the data and the decision boundary
figure
plot2DLinear(learnerA,XA,YA);
figure
plot2DLinear(learnerB,XB,YB);
```

Part c

I got an error rate of 0.0505 for data set A
and an error rate of 0.4646 for data set B.

Here is the code for predict.m

```
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) + ...
weights = obj.wts;
yhat = weights(1);
for i = 2:length(weights)
    yhat = yhat + Xte(:,i-1).*weights(i);
end
yhat = sign(yhat);

% (2) convert predictions to saved classes: Yte = obj.classes( [1 or 2] );
Yte = ones(length(yhat),1);
for i = 1:length(Yte)
   if(yhat(i) == -1)
      Yte(i) = obj.classes(1);
   else
       Yte(i) = obj.classes(2);
   end
end
```

Here is the code to compute the error rates for part C. It relies on the code for Part B.

```
plot2DLinear(learnerB,XB,YB);

%%
%part C

%get the predicted class labels for A and B
YhatA = predict(learnerA,XA);
YhatB = predict(learnerB,XB);

%get the error rate
errorA = length(find(YhatA~=YA))/length(YA);
errorB = length(find(YhatB~=YB))/length(YB);
```

Part d

We will use the following identity:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

When we first take the derivative, we end up with

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

We know that $z' = \frac{\partial z}{\partial \theta_i} = x_i^{(j)}$ thus expanding terms we get

$$\frac{\partial J_j}{\partial \theta_i} = -y^{(j)}x_i^{(j)} + x_i^{(j)}\sigma(z)y^{(j)} + \sigma(z)x_i^{(j)} - y^{(j)}\sigma(z)x_i^{(j)} + 2\alpha\theta_i$$

Cancelling out terms and then putting the $x_i^{(j)}$ factors together, we get

$$\frac{\partial J_j}{\partial \theta_i} = x_i^{(j)}(\sigma(z) - y^{(j)}) + 2\alpha\theta_i$$

The gradient vector $\nabla J_j(\theta)$ will be as follows

$$[\frac{\partial J_j}{\partial \theta_1}\frac{\partial J_j}{\partial \theta_2}...\frac{\partial J_j}{\partial \theta_d}]$$

Part e

Here is the code for train.m

```
function obj = train(obj, X, Y, varargin)
% obj = train(obj, Xtrain, Ytrain [, option,val, ...])  : train logistic classifier
%     Xtrain = [n x d] training data features (constant feature not included)
%     Ytrain = [n x 1] training data classes
%     'stepsize', val  => step size for gradient descent [default 1]
%     'stopTol',  val  => tolerance for stopping criterion [0.0]
%     'stopIter', val  => maximum number of iterations through data before stopping [1000]
%     'reg', val       => L2 regularization value [0.0]
%     'init', method   => 0: init to all zeros;  1: init to random weights;
% Output:
%   obj.wts = [1 x d+1] vector of weights; wts(1) + wts(2)*X(:,1) + wts(3)*X(:,2) + ...

  [n,d] = size(X);            % d = dimension of data; n = number of training data

  % default options:
  plotFlag = true;
  init     = [];
  stopIter = 1000;
  stopTol  = -1;
  reg      = 0.0;
  stepsize = 1;

  i=1;                                       % parse through various options
  while (i<=length(varargin)),
    switch(lower(varargin{i}))
    case 'plot',      plotFlag = varargin{i+1}; i=i+1;   % plots on (true/false)
    case 'init',      init     = varargin{i+1}; i=i+1;   % init method
    case 'stopiter',  stopIter = varargin{i+1}; i=i+1;   % max # of iterations
    case 'stoptol',   stopTol  = varargin{i+1}; i=i+1;   % stopping tolerance on surrogate loss
    case 'reg',       reg      = varargin{i+1}; i=i+1;   % L2 regularization
    case 'stepsize',  stepsize = varargin{i+1}; i=i+1;   % initial stepsize
    end;
    i=i+1;
  end;

  X1    = [ones(n,1), X];     % make a version of training data with the constant feature

  Yin = Y;                              % save original Y in case needed later
  obj.classes = unique(Yin);
  if (length(obj.classes) ~= 2) error('This logistic classifier requires a binary classification problem
  Y(Yin==obj.classes(1)) = 0;
  Y(Yin==obj.classes(2)) = 1;           % convert to classic binary labels (0/1)

  if (~isempty(init) || isempty(obj.wts))   % initialize weights and check for correct size
    obj.wts = randn(1,d+1);
  end;
  if (any( size(obj.wts) ~= [1 d+1]) ) error('Weights are not sized correctly for these data'); end;
  wtsold = 0*obj.wts+inf;

% 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

  %compute surrogate loss

  %computes the function given in 1d on each data point and adds it to the
  %     total loss for this iteration
  for k = 1:length(Y)
        zValueK = dot(obj.wts,X1(k,:));
        sigmaZk = 1/(1+exp(-zValueK));
        Jsur(iter) = Jsur(iter) + -Y(k)*log(sigmaZk) + (1-Y(k))*log(1-sigmaZk) ...
            + reg*sum((obj.wts).^2);
  end

  %divides by number of data points to get average loss
  %     this gives us the final surrogate loss for this iteration
  Jsur(iter) = Jsur(iter)/length(Y);

  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-');
  legend('Surrogate Loss','Error Rate');drawnow;

  for j=1:n,

      %gets the linear response of the data point with the current weights
    zValue = dot(obj.wts,X1(j,:));

    %calculate J' vector using formula derived for it in part 1d
    sigmaZ = 1/(1+exp(-zValue));
    grad = zeros(1,length(obj.wts));
    for i = 1:length(obj.wts)
        grad(i) = X1(j,i) * (sigmaZ - Y(j)) + 2*obj.wts(i)*reg;
    end

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

  done = false;

  %{
  Here if either the stop iteration criteria is met or the stop tolerance
        criteria is met, then this loops stops
  %}

  %sees if number of iterations is equal to stopIter
  if(iter >= stopIter)
     done = true;
  end
  if(iter > 1)
      %sees if change in error is less than stopTol
     if(abs(Jsur(iter-1)-Jsur(iter))<stopTol)
        done = true;
     end
  end

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

Part f

For Data Set A, the convergence seemed to work pretty well with the step size already given that starts at 1 and then decreases with each iteration, so I did not touch the step size. I decided to look at number of iterations and stop tolerance. After about 30-40 iterations, it seemed to be converging. When I zoomed in, however, I noticed that the error rate was still decreasing. I decided then to do 100 iterations total and set the stop tolerance really low at 0.0000001 in order to get the error rate as low as possible. In the end, all the points seemed to be classified perfectly which is what we wanted here since the data was linearly separable.

Here is the surrogate loss and error rate plot for Data Set A

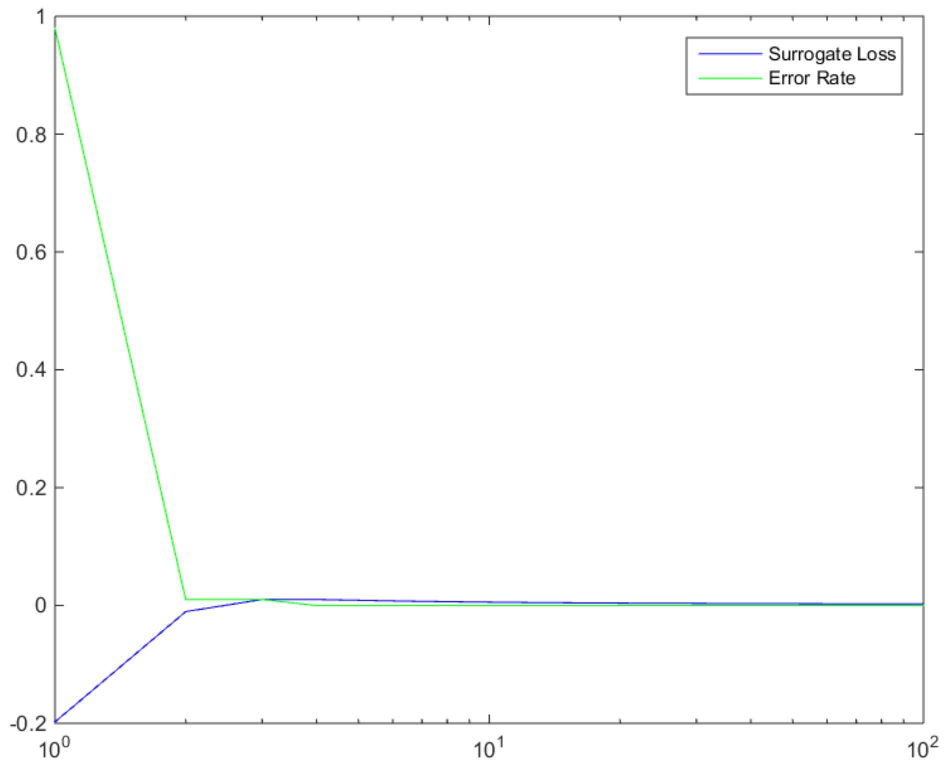

Figure 5: The surrogate loss and error rate as function of number of iterations

Here is the classification plot using the final weights computed to attempt to separate Data Set A.

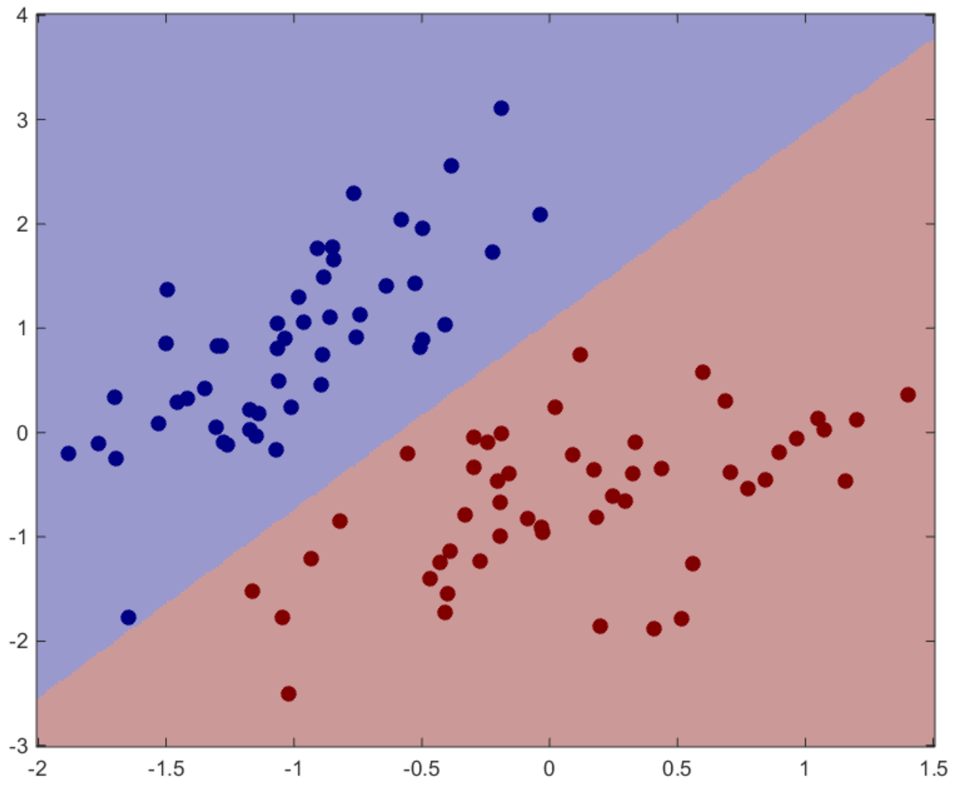

Figure 6: The predicted and actual classification of points. In this case, they were all predicted correctly!

For Data Set B, the convergence seemed to work well also so I left the step size as it was for data set A. It did take longer for the surrogate loss to converge, so I set the max number of iterations to 1000 and it was near converenge around 400 iterations. I decided to set the stop tolerance to the same value at 0.0000001 in order to get the error rate as low as possible. In this case, it did not execute all iterations before the stop tolerance was reached. There were around 800 iterations and then the stop tolerance criteria was met. The error rate however was still somewhat high. This is likely due to the fact that the data is not linearly separable so we reached a point where the error rate was as low as we could possibly make it.

Here is the surrogate loss and error rate plot for Data Set B

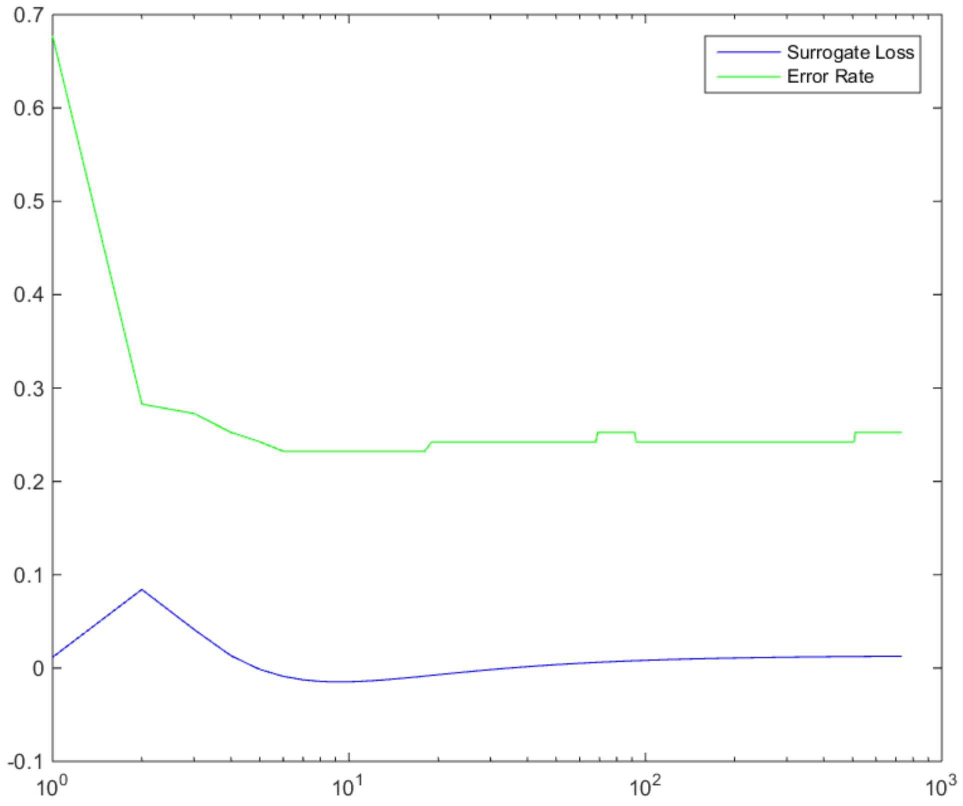

Figure 7: The surrogate loss and error rate as function of number of iterations

Here is the classification plot using the final weights computed to attempt to separate Data Set B

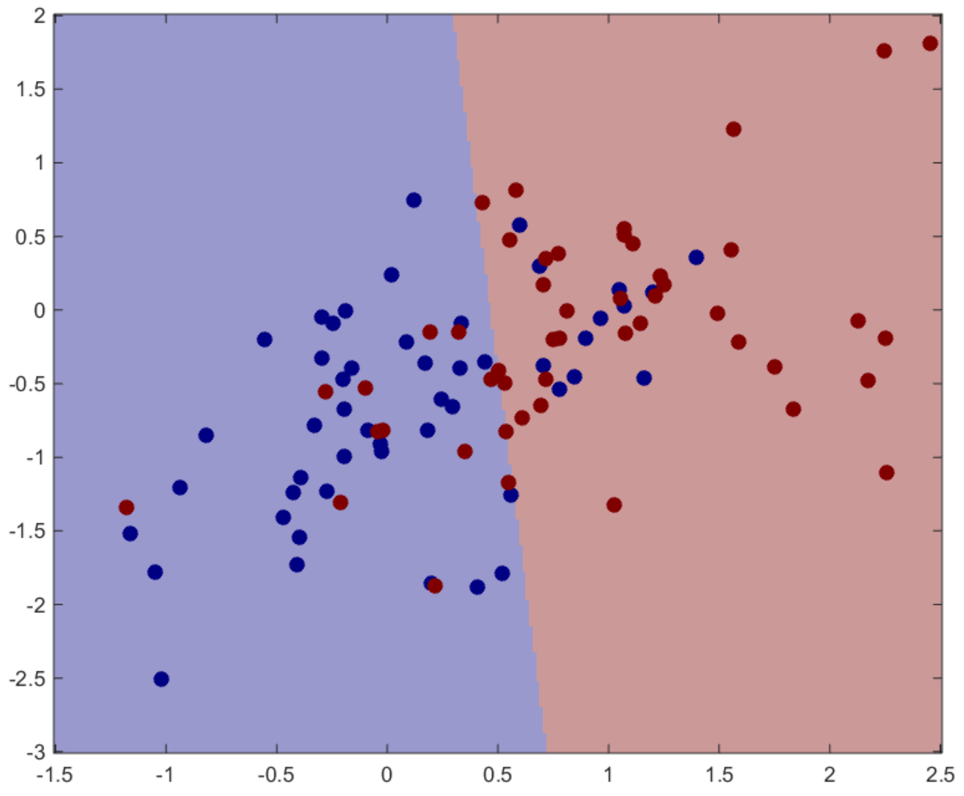

Figure 8: The predicted and actual classification of points

As part as code is concered, in addition to the train.m function that I listed in part e as well as the code from part a to initialize the variables, I also used the following script to call the train functions and generate the classification plot.

```
%Part F
%Data set A
learnerA=logisticClassify2();
learnerA=train(learnerA,XA,YA,'stopIter',100,'stopTol',0.0000001);

fig(3)
plotClassify2D(learnerA,XA,YA)

%%
%Data set B
learnerB=logisticClassify2();
learnerB=train(learnerB,XB,YB,'stopIter',1000,'stopTol',0.000001);

fig(3)
plotClassify2D(learnerB,XB,YB)
```

Problem 2

Part a

This equation describes a vertical line through the points.
It will thus easily separate the points in (a) and (b).

For the points in (c), if the points (2,2) and (6,4) had one class and the point (4,8) was of a different class, then a vertical line could not separate them.

Thus the most points that this one could shatter is 2
The VC dimension is thus estimated at 2

Part b

This equation describes a circle with an arbitrary center and radius
It will thus easily separate points in (a) and (b)

With (c), if all points have the same class then of course this equation will separate them.
The other case to consider is one point has one class and the other two points have another class.
In this case, you can draw a small circle around the one point and now you have successfully separated the classes.

With (d), this equation can technically separate them.
If zero points are in a class, then this is trivial.
If 1 point is in a class, so 3 in another class, then we can draw a circle around the point.
If there are 2 points in a class, then no matter which two you pick for a class, you can actually draw a circle around those two points or the other two points and you have a successful classification.
This only happened due to these particular points and how they were spaced. If I shifted one of the points closer to the other points, then there would be no way to separate them.

Even though we could shatter 4 points, it only seems like we could reliably shatter 3 points in general.
The VC dimension is thus estimated at 3.

Part c

This is a line that is centered at the origin and not horizontal since then $a \neq 0$.

It can easily separate points in (a) and (b)

With (c), if the points (6,4) and (4,8) had one class and the point (2,2) was part of another class, then there would be no way to separate them using a line at the origin. If it were possible to separate them, then we would be able to split the number line in two and put $\frac{2}{3}$ and 2 into one part while putting 1 into the other part.

Thus, the most number of points this equation can shatter is 2
The VC dimension is thus 2