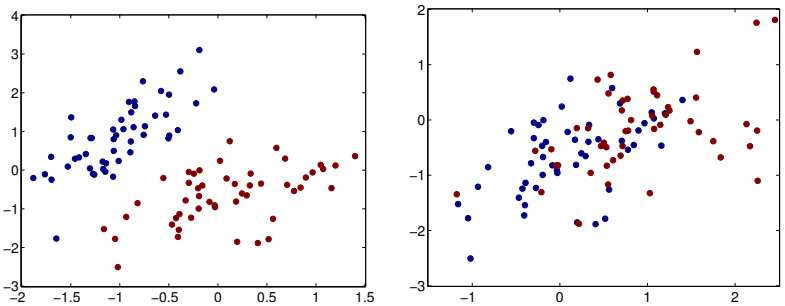
CS178 Homework #3 Solution

Machine Learning & Data Mining: Winter 2015

Problem 1: Logistic Regression

(a) Plotting the data:

```
% Run the code provided to extract the data into XA, XB
figure(1); plotClassify2D([],XA,YA);  % plot XA
figure(2); plotClassify2D([],XB,YB);  % plot XA
```



Data A Data B

Figure 1: The two binary data sets.

(b) Plotting the decision boundary:

```
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(0)+wts(2)*X(1)+wts(3)*X(2)
%
  [n,d] = size(X);
  if (d~=2) error('Sorry -- plot2DLinear only works on 2D data'); end;
  classes=unique(Y);
  if (length(classes)~=2) error('Sorry -- plot2DLinear only works on binary data'); end;
  c0 = find(Y==classes(1)); c1=find(Y==classes(2));
  Xplt = linspace(min(Xtrain(:,1)),max(Xtrain(:,1)),200);
  % decision boundary is:
  %  logistic( w2 x2 + w1 x1 + w0 ) =.5  <=> w2 x2 + w1 x1 + w0 = 0  <=> x2 = -w0 -w1/w2 x1;
  plot(X(c0,1),X(c0,2),'bo',...       % class zero training data
       X(c1,1),X(c1,2),'rs',...       % class one training data
       Xplt,-obj.wts(1)/obj.wts(3) - obj.wts(2)/obj.wts(3).*Xplt,'k-');  % decision boundary
  drawnow;
```

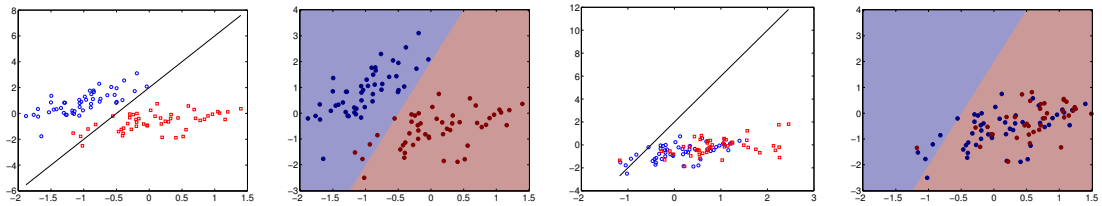
(See next part for plots.)

(c) Making predictions:

```
function Yte = predict(obj,X)
% Yhat = predict(obj, X)  : make predictions on test data X

  Yte = obj.classes( 1 + round(logistic(obj,X)) );              % can just use logistic & round
  %Yte = obj.classes( 1 + ((obj.wts(1)+X*obj.wts(2:end)')>0) );  % equivalent "sign" computation
```

We can compute the error rate, and also verify that our decision boundary from part (b) agrees:

```
 learner = logisticClassify2();
 learner = setWeights(learner, [.5 , 1 , -.25]);
 learner = setClasses(learner, unique(YA));
 err(learner, XA,YA),
% ans = 0.0505
 figure(1); plot2DLinear(learner,XA,YA);
 figure(2); plotClassify2D(learner,XA,YA);

 learner = setClasses(learner, unique(YB));
 err(learner, XB,YB),
% ans = 0.4646
 figure(3); plot2DLinear(learner,XB,YB);
 figure(4); plotClassify2D(learner,XB,YB);
```



Data A (plot) Data A (predict) Data B (plot) Data B (predict)

Figure 2: Manual classifier, decision boundaries.

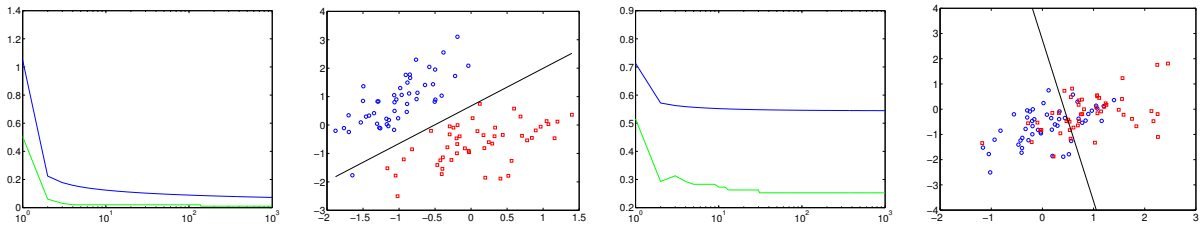
(d) Computing the gradient of the negative log-likelihood.

(e) Training using SGD:

```
% In train.m -- just the primary training loop:
while (~done)
  step = stepsize/iter;               % update step-size
  sig  = logistic(obj, X);            % compute outputs: sigma( Xtrain * wts' )
  Jsur(iter) = mean( [log(sig(Y==1)) ; log(1-sig(Y==0))] ) + reg*sum(obj.wts.^2);
  J01(iter) = err(obj,X,Yin);

  if (plotFlag), switch d,            % Plots to help with visualization
    case 1, fig(2); plot1DLinear(obj,X,Y);    %  for 1D data display data and the function
    case 2, fig(2); plot2DLinear(obj,X,Y);    %  for 2D data, 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 i=1:n,
    sigx = logistic(obj,X(i,:));      % compute output for example i

    % Compute gradient for loss term of example i:
    grad = -Y(i)*(1-sigx)*X1(i,:) + (1-Y(i))*sigx*X1(i,:) + reg*obj.wts;

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

  done = (iter>stopIter) || ((iter>1) && abs(Jsur(iter)-Jsur(iter-1))<stopTol);
  iter = iter + 1;
end;
```

(f) Optimized learners on the two data sets:

```
lrA = logisticClassify2(XA,YA, 'stepsize',.1, 'stopIter',1000, 'reg',0.0);
err(lrA,XA,YA),
% ans = 0.0101

lrB = logisticClassify2(XB,YB, 'stepsize',.1, 'stopIter',1000, 'reg',0.0);
err(lrB,XB,YB),
% ans = 0.2525
```



Losses (A) Boundary (A) Losses (B) Boundary (B)

Figure 3: Trained classifiers on the two binary data sets.

Problem 2: Shattering and VC Dimension

(a) This learner is a standard peceptron on one feature (so, two parameters). It has VC dimension 2; its decision boundaries are vertical lines in the two-dimensional feature space (e.g., a solution to $a + bx_1 = 0$), with either side able to take either class value. Hence, it can shatter either (a) or (b) but not (c) or (d).

(b) This learner has three parameters, so you should probably *guess* shattering (a)–(c), but let's work it out. Its decision boundaries are circles centered at an arbitrary point given by the parameters (a, b); since the value increases with distance from (a, b), for $c < 0$ we'll get class negative inside a circle, and positive outside, with radius determined by c (there's no way to get the reverse). So, to shatter (b), for example, if only one point were negative we would just center the circle on that point. Similarly, for (c), whichever two points are negative, we center and scale the circle so those two are inside, and the other is not. (The other cases are easy.) But, for (d), we can't do it – if the points at $(2, 2)$ and $(8, 6)$ are negative, any circle that contains both with also contain the others.

(c) This learner has three parameters, but you can see that they're not working independently. If I replace $\alpha = a * b$ and $\beta = c/a$, there are really only two independently set-able parameters, so you should guess VC dimension 2. To reason further, this is a linear classifier with no constant (bias) term; so its decision boundary will be a line passing through $(0, 0)$, and (by changing the sign of a) it can predict either class on either side. Shattering (b) is straightforward; if the data signs differ, the line will pass between them. But (c) is not shattered; if $(4, 8)$ and $(6, 4)$ are both negative but $(2, 2)$ is positive, no line passing through the origin can predict them correctly.