CS273a Homework #3 Solution

Intro to Machine Learning: Fall 2012

Problem 1: Decision Trees

You can most easily do this by hand, but since I have to type it I will put it in Matlab format:

```
Xy = [0 0 1 1 0 -1
      1 1 0 1 0 -1
      0 1 1 1 1 -1
      1 1 1 1 0 -1
      0 1 0 0 0 -1
      1 0 1 1 1  1
      0 0 1 0 0  1
      1 0 0 0 0  1
      1 0 1 1 0  1
      1 1 1 1 1 -1 ];
ep=1e-12;            % move values away from log(0)=-infty
X = Xy(:,1:end-1); Y=Xy(:,end);
```

(a) Calculate the entropy of the class variable y

```
p = mean(Y>0);
Hy = -(p*log2(p) + (1-p)*log2(1-p));
% .971 bits
```

(b) Calculate the information gain for each feature x_i. Which feature should I split on first?

```
for i=1:5,
 idx = X(:,i)>0;
 if (sum(idx)==0 || sum(~idx)==0) IG(i)=0; continue; end;
 p1 = mean( Y(idx)>0 )+ep; p0 = mean(Y(~idx)>0)+ep; a = mean(idx)+ep;
 IG(i) = Hy + a*(p1*log2(p1)+(1-p1)*log2(1-p1))+(1-a)*(p0*log2(p0)+(1-p0)*log2(1-p0));
end;
IG,
% IG = 0.0464    0.6100    0.0058    0.0913    0.0058
% Pick feature # 2
```

(c) Draw the complete decision tree that will be learned from these data.

```
% Splitting on feature 2 divides the data:
Xy = [1 1 0 1 0 -1
      0 1 1 1 1 -1
      1 1 1 1 0 -1
      0 1 0 0 0 -1
      1 1 1 1 1 -1];
% On this branch we can always predict "-1"

Xy = [0 0 1 1 0 -1
      1 0 1 1 1  1
      0 0 1 0 0  1
      1 0 0 0 0  1
      1 0 1 1 0  1 ];
% On this branch, we'll need to split; repeating, we find that the next best is feature #1

Xy = [1 0 1 1 1  1
      1 0 0 0 0  1
```

```
      1 0 1 1 0  1 ];
% On this branch we just predict "1"

Xy = [0 0 1 1 0 -1
      0 0 1 0 0  1 ];
% On this branch we'll need to split again; by inspection split on feature #4 and predict 1 or -1

% So the final rule is:
% if (long) discard
% else
%   if (known) read
%   else
%     if (has 'grade') discard
%     else read
```

Problem 2: VC Dimension and Decision Trees

(a) With one feature, every split will be on that feature. For depth $\leq D$, there are 2^D possible regions, which can shatter 2^D points but not $2^D + 1$. (The counterexample is, give the points an alternating pattern of class labels from left to right.)

(b) With two features and D=1, we can split three points but not four. For three points, place them at $a = (0, 0)$, $b = (1, 2)$, and $c = (2, 1)$. Then we can split any pattern by thresholding feature 1 at 0.5 or at 1.5, or feature 2 at 1.5.

For four points, a true counterexample requires more care. Any placement of four points in two dimensions induces an ordering of those points along each feature – for example, in the previous three point case we had ordering $[abc]$ along feature 1, and $[acb]$ along feature 2. Now, order the points according to feature 1 as $[abcd]$, so splits on x_1 can produce groupings $(a)(bcd)$, $(ab)(cd)$, and $(abc)(d)$. Now order by feature 2; we need to produce groupings $(b)(acd)$ and $(c)(abd)$, so use order $[bdac]$ without loss of generality. Now we can also create pattern $(bd)(ac)$, but cannot create pattern $(bc)(ad)$ – so we cannot shatter the data.

(c) Just by our previous construction, make feature three have order $[bcad]$. We can now split on feature 3 to get the remaining pattern. A concrete construction would be:

$$a = (0, 2, 2) \qquad b = (1, 0, 0) \qquad c = (2, 3, 1) \qquad d = (3, 1, 3);$$

Problem 3: Decision trees; bagging

```
rand('state',0); randn('state',0);
data = load('data/spambase.data');
X = data(:,1:57); Y=data(:,end);                  % extract features & class labels
[X,Y] = reorderData(X,Y);                         % permute out of class-based order
[Xt,Xv,Yt,Yv] = splitData(X,Y,.6);                % divide into training & test
[Xt,S] = rescaleData(Xt); Xv = rescaleData(Xv,S); % pre-process data scaling
```

(a) learn a decision tree classifier for the data

```
tc = treeClassify(Xt,Yt, 1,inf,0,100);
err(tc,Xv,Yv),
% Error rate is ~ .0957
```

(b) search over values for the maximum depth cutoff

```
for d=1:20,
  tc = treeClassify(Xt,Yt, 1,d,0,100);
  ed(d) = err(tc,Xv,Yv);
end;
figure; plot(1:20,ed,'b-','linewidth',3);   % min is around d=7 for my data
```

(c) use the `minParent` option to control complexity

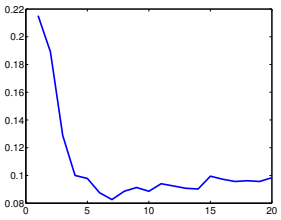
```
for p=1:50,
  tc = treeClassify(Xt,Yt, p,inf,0,100);
  ep(p) = err(tc,Xv,Yv);
end;
figure; plot(1:50,ep,'r-','linewidth',3);   % min is around p=20 for my data
```

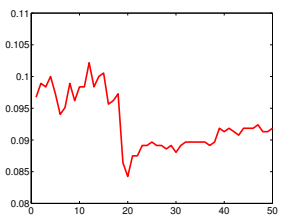
(d) learn a bagged ensemble of decision trees

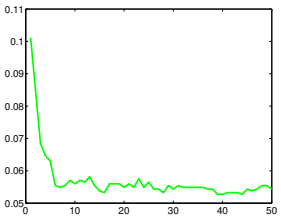
```
yh=zeros(size(Yv,1),50);
for b=1:50,
  [Xb,Yb] = bootstrapData(Xt,Yt,size(Xt,1));
  boot{b} = treeClassify(Xb,Yb, 1,inf,0,100);
  yh(:,b) = predict(boot{b},Xv);
  eb(b) = mean( (mean(yh(:,1:b),2)>.5)~= Yv);
end;
figure; plot(1:50,eb,'g-','linewidth',3);  % pretty good after b > 8 or so
```

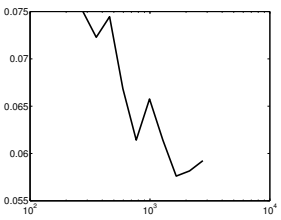
(e) re-run your bagged ensemble learning algorithm, but with fewer data per bootstrap

```
ns = size(Xt,1)*logspace(-1,0,10);
for i=1:length(ns),
  n=ceil(ns(i));
  yh=zeros(size(Yv,1),50);
  for b=1:25,
    [Xb,Yb] = bootstrapData(Xt,Yt,n);
    boot{b} = treeClassify(Xb,Yb, 1,inf,0,100);
    yh(:,b) = predict(boot{b},Xv);
  end;
  en(i) = mean( (mean(yh(:,1:b),2)>.5)~= Yv);
end;
figure; semilogx(ns,en,'k-','linewidth',3);  % smaller n increases underfitting behavior
```


(b) depth


(c) minParent


(d) # bagged


(e) # data in bootstrap

Problem 4: Boosting

I'll order the data 1–7, starting at the upper left ("point 1") and going down, then moving to the upper right ("point 5") and downward.

First, take all the data points with equal weight. The first classifier picks $x_1 \leq .5$, so it gets points 4 and 5 wrong. To use Matlab (since I'm typing anyway):

```
w   = [1  1  1  1  1  1  1]; w=w/sum(w);    % initialize weights
y   = [1  1  1 -1  1 -1 -1];                % correct class values
yh1 = [1  1  1  1 -1 -1 -1];                % predictions
e = (y~=yh1)*w', a = .5*log((1-e)/e),       % compute weighted error and coefficient
%e =
%    0.2857
%a =
%    0.4581
w = w.*exp(-a*(y.*yh));             % signed errors: +1 if right, -1 if wrong
w=w/sum(w),
%w =
%    0.1000    0.1000    0.1000    0.2500    0.2500    0.1000    0.1000
```

The second weighted classifier picks $x_2 > 2.5$ (actually, there are several equivalent classifiers; pick one), getting points 3 and 6 wrong.

```
yh2 = [1  1 -1 -1  1  1 -1];                % our predictions for this classifier
e = (y~=yh2)*w', a = .5*log((1-e)/e),       % compute weighted error and coefficient
%e =
%    0.2000
%a =
%    0.6931
w = w.*exp(-a*(y.*yh));
w=w/sum(w),
%w =
%    0.0625    0.0625    0.2500    0.1563    0.1563    0.2500    0.0625
```

Note that the data we got right twice in a row are now quite small; data we got wrong at least once are larger. At this point, the "weighted average" of only two classifiers will just pick whichever one has higher weight, so our overall prediction is just the second predictor ($a_2 > a_1$), and we still get two data points wrong.

Finally, split at $x_2 > 1.5$, getting pts 6 and 7 wrong (again, there are some equivalent splits):

```
yh3 = [1  1  1 -1  1  1  1];                % our predictions for this classifier
e = (y~=yh3)*w', a = .5*log((1-e)/e),       % compute weighted error and coefficient
%e =
%    0.3125
%a =
%    0.3942
w = w.*exp(-a*(1-2*neq));
w=w/sum(w),
%w =
%    0.0455    0.0455    0.1818    0.1136    0.1136    0.4000    0.1000
```

With three classifiers, it is possible to get a non-trivial voted majority. Our overall predictions are:

```
sign( 0.4581*yh1 + 0.6931*yh2 + 0.3942*yh3 ),
%ans =
%     1     1     1    -1     1     1    -1
```

which gets only one training data point wrong.