Table of Contents

```
clear; clc;
```

Part 3

Use make cloud.m to generate training data sampled from two point clouds
****

```
[X, t] = make_cloud();
```

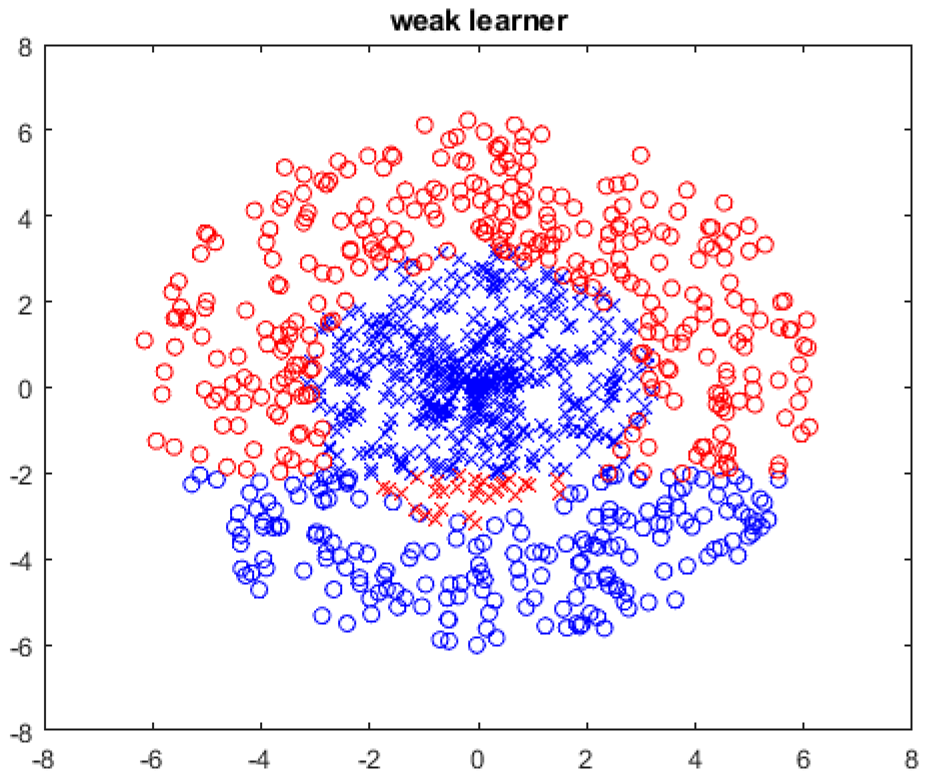
Part 4

First, let’s establish the performance of a single weak learner: use weaklearn.m to classify the data directly (with uniform weights). Plot the data (for example, use scatter) and distinguish between classes, and whether we classify correctly, or not (e.g. ’x’ vs ’o’ for the true class, and blue versus red for correct/wrong classification). How well did this classifier perform? Could a different single linear classifier perform better on this data? If so, what would the best case look like?
****

Honestly the weak learner does pretty well. It gets 64.7% correct which is impressive. I don't think it is possible for a linear classifier to do significantly better on this. The distribution of the 'donut hole' does not appear to be perfectly circular, so perhaps a linear model could do better if its slope is parallel to the 'major axis' of the hole elipse. Or it could do a tiny bit better just based on the random density of points there might be a line that has higher density so it would nominally be better. But in reality, neither of these methods are significantly better.

```
params = weaklearn(X, t);
preds = weakeval(X, params);

figure('Name', 'weak learner' );
plot_preds('weak learner', X, t, preds);  % helper function to do the
 plotting
```

```
A weak learner gets 66 % correct
```



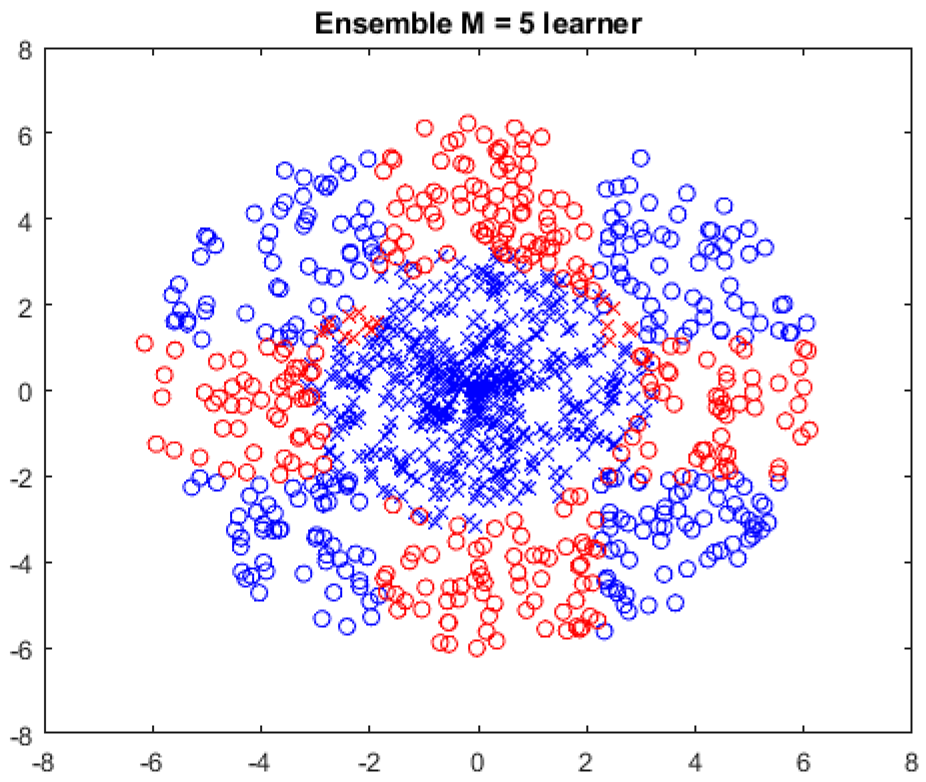
Part 5

Use boostlearn.m to classify the same data as above with M =5 weak classifiers. Generate and turn in a figure showing the classification result.
****

This looks pretty fun. It is a fun shape and not one that I initially expected. But in hindsight, it makes sense given the nature of our weak learners.

```
[params, alpha] = boostlearn(X, t, 5);
preds = boosteval(X, params, alpha);

figure('Name', 'Ensemble M=5' );
plot_preds('Ensemble M = 5 learner', X, t, preds);  % helper function
 to do the plotting
```

```
A Ensemble M = 5 learner gets 72.6 % correct
```



Part 6

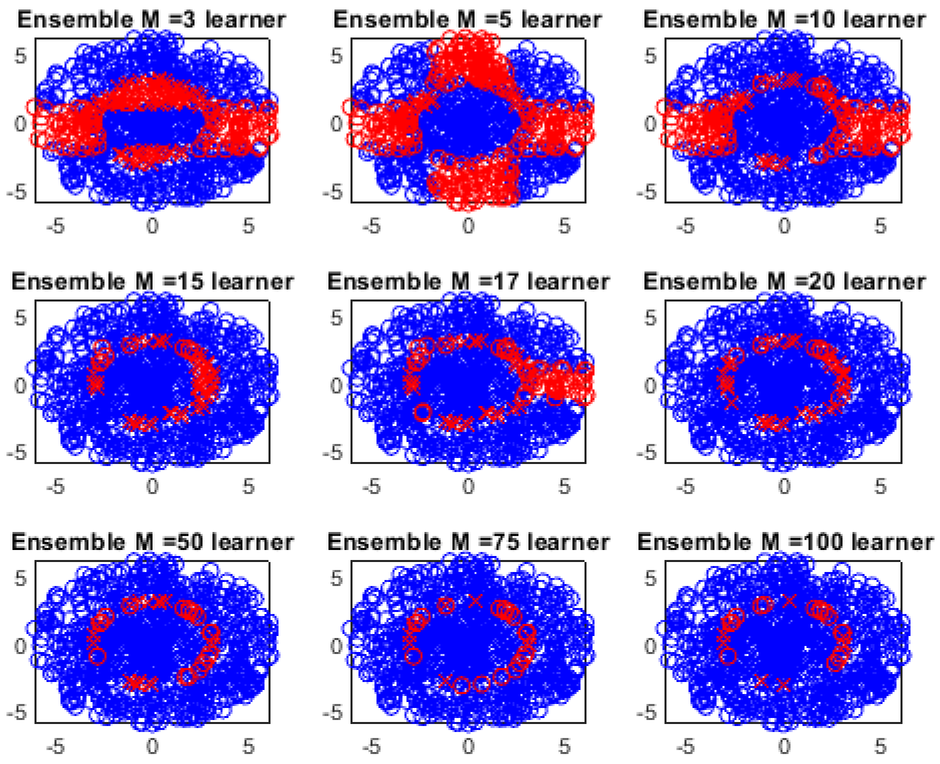
Try larger values of M in the range 1 to 100 and study how the classifier changes. Based on visual inspection of the results: At what point do you believe the classifier is sufficiently reflecting the data? At what point do you believe the classifier might be over-fitting? **

The ensemble classifier reaches peak performance right around 15-20 (it changes each run). So within that range is a pretty accurate representation, and beyond that it starts to overfit. I would pick my M to be 17 because that seems to have a good balance of accuracy to overfitting based on multiple trials.

```
figure('Name', 'Comparison of different M values' );
tiledlayout(3, 3);

Ms = [3 5 10 15 17 20 50 75 100];
for M=Ms
    [params, alpha] = boostlearn(X, t, M);
    preds = boosteval(X, params, alpha);

    nexttile
    plot_preds(strcat('Ensemble M = ',num2str(M), ' learner'), X, t,
 preds);  % helper function to do the plotting

end
```

```
A Ensemble M =3 learner gets 76.6 % correct
A Ensemble M =5 learner gets 72.6 % correct
```

```
A Ensemble M =10 learner gets 86 % correct
A Ensemble M =15 learner gets 93.6 % correct
A Ensemble M =17 learner gets 91.8 % correct
A Ensemble M =20 learner gets 95.3 % correct
A Ensemble M =50 learner gets 96.5 % correct
A Ensemble M =75 learner gets 97.3 % correct
A Ensemble M =100 learner gets 97.6 % correct
```



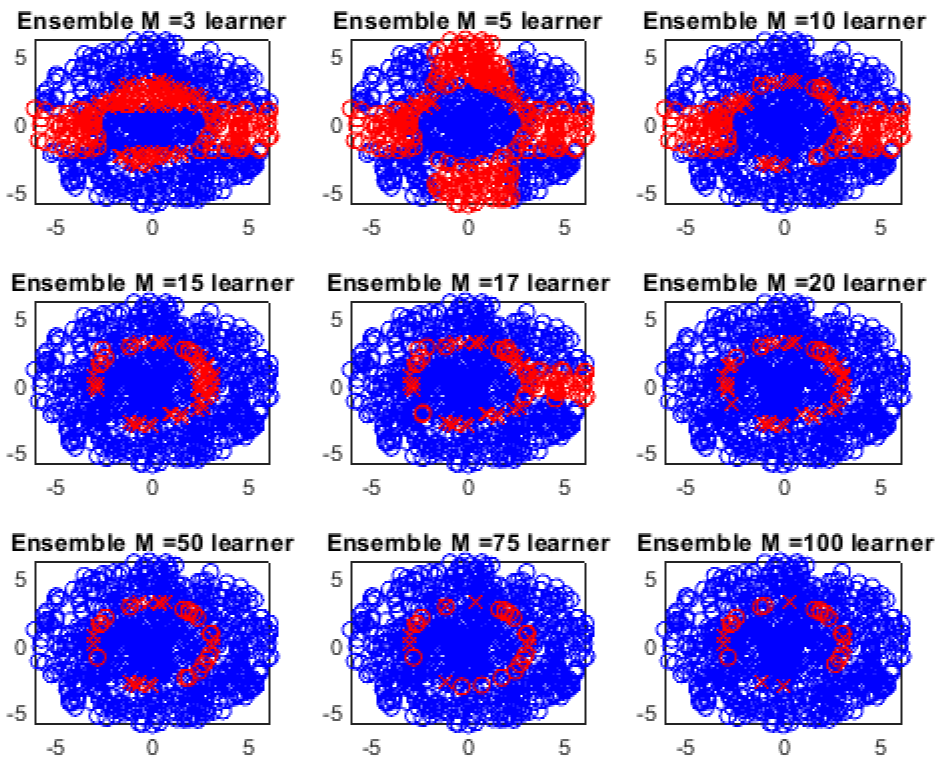
Part 7

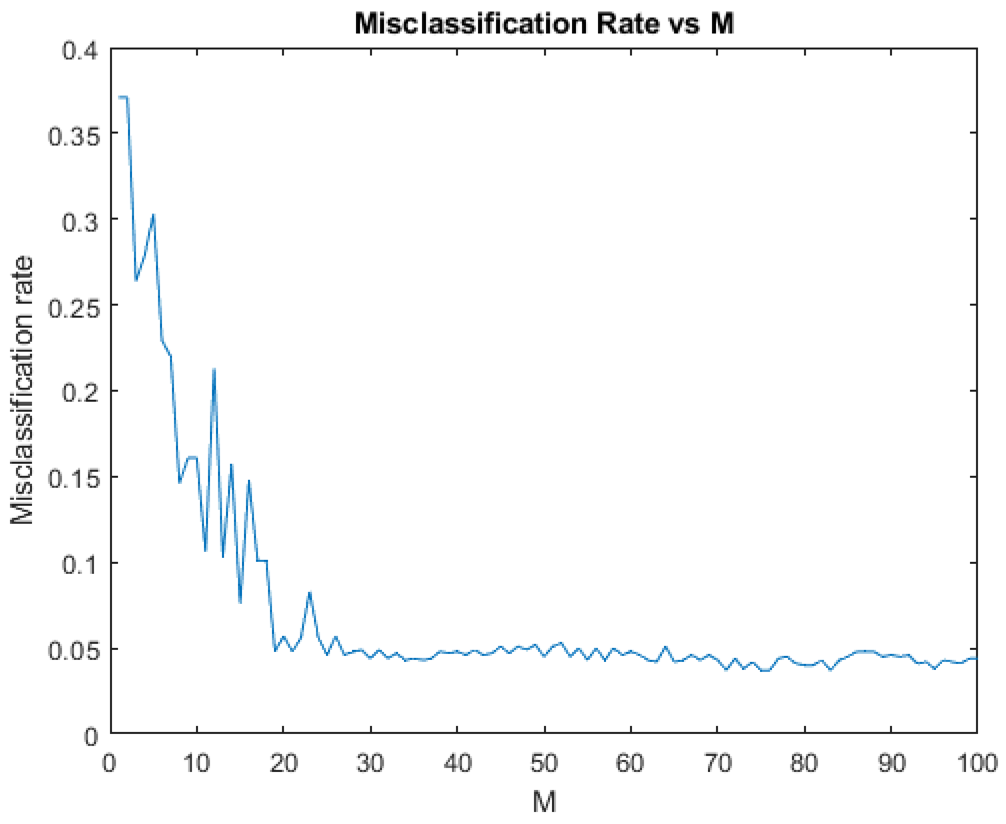
Use make cloud.m to generate a second set of observations (validation data). Test values for M, systematically. That is, for varying values of M, train AdaBoost on the training data, and evaluate its performance on the validation data. Generate and turn in a plot with M on the x-axis and the misclassification rate for validation data on the y-axis. Describe the shape of this graph.
****

The shape is interesting and quite unexpected. I expected the error rate to go back up, as it overfits the data. But it just levels off at around M=20. This means that it is learning most of the shape of the underlying function with about M=20, and the extra weak learners do not really effect the performance of the model at all. For our weak learner and this example dataset, it implies that too many weak learners does not really worsen the performance of the ensemble at all, as the extra ones are pretty much ignored. This is not true in all cases, but for our very specific application in this lab, this seems to hold.

```
[X_test, t_test] = make_cloud();

misclassification_rate = zeros(1, 100);
```

```
for M = 1:100
    [params, alpha] = boostlearn(X, t, M);
    preds = boosteval(X_test, params, alpha);

    correct = preds == t_test;
    wrong = ~correct;

    misclassification_rate(M) = sum(wrong) / numel(t_test);
end

figure('Name', 'Misclassification Rate vs M')
plot(1:100, misclassification_rate);
title('Misclassification Rate vs M');
ylabel('Misclassification rate');
xlabel('M');
```





boostlearn.m

```
%% adaBOOST model LEARNer
%  Uses the AdaBoost algorithm to train a classifier on data.
% Inputs
%  X - D x N : Observations
%  t - N x 1 : class labels
%  M - The number of weak learners to include in the ensemble.
% Outputs
%  params - A matrix containing the parameters for the M weak
 learners.
%  alpha - A vector of weights used to combine the results of the
%    M weak learners.

function [params, alpha] = boostlearn(X, t, M)
    [D, N] = size(X);
    weights = (1 / N) * ones(N, 1);

    alpha = zeros(M, 1);
    params = zeros(3, M);

    for k = 1:M
        params(:, k) = weaklearn(X, t, weights);
        preds = weakeval(X, params(:,k));  % get model predictions
        correct = preds == t;  % get logical vector of ones we got
 right
        wrong = ~correct;

        eps = sum(weights(wrong)) / sum (weights);
        a = log((1-eps)/eps);
        alpha(k) = a;

        weights = weights .* exp(a * wrong);
    end

end
```

boosteval.m

```
%% adaBOOST model EVALuator
%  Uses a trained AdaBoost algorithm to classify data.
% Inputs
%  X - Matrix with observations (in columns) to classify. (D x N)
%  params - Output of boostlearn.m (weak learner parameters).
%  alpha - Output of boostlearn.m (weak learner mixing coefficients).
% Outputs
%  C - A matrix with predicted class labels (-1 or 1) for the input
%    observations in X.

function [C] = boosteval(X, params, alpha)
   % who cares about memory, lets just store NxM matrix of all our
 predictions! lol
   [D, N] = size(X);
   [p, M] = size(params);

   preds = zeros(N, M);
   for k = 1:M
      preds(:,k) = weakeval(X, params(:,k));;
   end

   scaled = preds * alpha;  % scales predictions by alpha, and sums
 them (dot product)
   C = 2 * (scaled > 0) - 1;
end
```

plot_preds.m

```
function [acc] = plot_preds(tit,X, t, preds)
%PLOT_PREDS Summary of this function goes here
%   Detailed explanation goes here
```

```
idx = (t > 0) & (t == preds); % true L1
plot( X(1,idx), X(2,idx), 'bx' ); hold on;
idx = (t < 0) & (t == preds); % true L2
plot( X(1,idx), X(2,idx), 'bo' );
idx = (t > 0) & (t.*preds < 0); % false L1
plot( X(1,idx), X(2,idx), 'rx' );
idx = (t < 0) & (t.*preds < 0); % false L2
plot( X(1,idx), X(2,idx), 'ro' );
title(tit);

acc = 100*sum(preds==t) / numel(t);

disp(['A ' tit ' gets ' num2str(acc) ' % correct']);
end
```

weaklearn.m

```
%% WEAK LEARNer
%  Trains a simple classifier which achieves at least 50 percent
 accuracy.
% Inputs
%  X - Matrix with observations (in columns) to train. (D x N)
%  t - a vector (N x 1) with a label (-1 or +1) for each of the N data
 points
%  v - (Optional) Column vector with data weight for each data point.
 (N x 1) Defaults to uniform weights.
% Outputs
%  params - Parameters for the weak trained model (column vector).


function [params] = weaklearn(X, t, v)

    X0 = X(:,t==1);
    X1 = X(:,t==-1);

    if nargin == 2
        W0 = ones(size(X0,2),1);
        W1 = ones(size(X1,2),1);
    else
        W0 = v(t==+1);
        W1 = v(t==-1);
    end

    best_d = 1;
    best_x = 0;
    best_err = inf;
    is_01 = 1;

    X = [X0, X1];
    W = [-W0; W1];
```

```
        for d = 1:size(X0,1)
            [~,IX] = sort(X(d,:));

            err = cumsum(W(IX));
            [min_cum,min_k] = min(err);
            best_01 = sum(W0) + min_cum;
            best_01_x = X(d,IX(min_k));

            err = cumsum(-W(IX));
            [min_cum,min_k] = min(err);
            best_10 = sum(W1) + min_cum;
            best_10_x = X(d,IX(min_k));

            if best_01 < best_err
                best_d = d;
                best_x = best_01_x;
                best_err = best_01;
                is_01 = 1;
            end

            if best_10 < best_err
                best_d = d;
                best_x = best_10_x;
                best_err = best_10;
                is_01 = 0;
            end
        end

        beta = zeros(size(X0,1),1);
        beta(best_d) = 1;
        params = [beta;-best_x];
        if is_01
            params = -params;
        end
    end
```

weakeval.m

```
%% WEAK learner EVALuate
%  Uses a simple classifier with given parameters to classify data.
% Inputs
%  X - Matrix with, in each column, an observation to assign labels.
 (D x N)
%  params - Parameters from weaklearn.m for a weak classifier.
% Outputs
%  C - Vector of class labels (1 or -1) for each input observation.

function [C] = weakeval(X, params)
    C = ((X'*params(1:(end-1)) + params(end)) > 0)*2 - 1;
end
```

make_cloud.m

```
%% MAKE a CLOUD of points
%%  Samples 1000 observations in 2D from two classes.
%% Inputs
%%  None
%% Outputs
%%  X - 500 observations from each class (in columns).
%%  t - labels

function [X,t] = make_cloud()
    N = 500;
    X0 = 6.5*rand(2,N);
    X1 = 6.5*rand(2,N);

    X0 = (0.5).*[X0(2,:);X0(2,:)].*[sin(X0(1,:)); cos(X0(1,:))];
    X1 = (0.5).*(6 + [X1(2,:);X1(2,:)]).*[sin(X1(1,:)); cos(X1(1,:))];

    X = [X0,X1];
    t = [ones(N,1); -ones(N,1)];
end
```

Published with MATLAB® R2021a