Problem Set 3

By Steven Cao *(written using MATLAB R2019b)*

Part 1 - Trees

Part 1, Problem 1 - Preliminary Setup

```
% Set the seed for this session.
s = RandStream('mt19937ar','Seed',234);
RandStream.setGlobalStream(s);

% Load our datasets.
nes_dataset = readtable('nes2008.csv');
oj_dataset = readtable('oj.csv');

% Define the input variables for both datasets.
nes_inputVar = {'female','age','educ','dem','rep'};
oj_inputVar = oj_dataset.Properties.VariableNames(3:end); % exclude first two columns
    % (first column is just the row ID; second column is just the output variable)

% Define the formulae for both datasets.
nes_formula = 'biden ~ female + age + educ + dem + rep';
oj_inputVar_stringified = join(string(oj_inputVar), " + ");
oj_formula = cat(2, 'Purchase ~ ', char(oj_inputVar_stringified));
    % (defining oj_formula is a little bit roundabout, but it's basically "Purchase ~ everythin

% Define "p" as the problem set requests, though I won't be actually using this variable.
p = nes_dataset(:,nes_inputVar);

% Define our "parameter sweep" range for the learning rate hyperparameter (used for boosting-ba
learningRate = 0.0001:0.001:0.0400;

% Define a helper function to calculate MSE.
MSE = @(estimatedValues, actualValues, sampleSize) ...
    sum( (estimatedValues-actualValues).^2 ) / sampleSize;
```

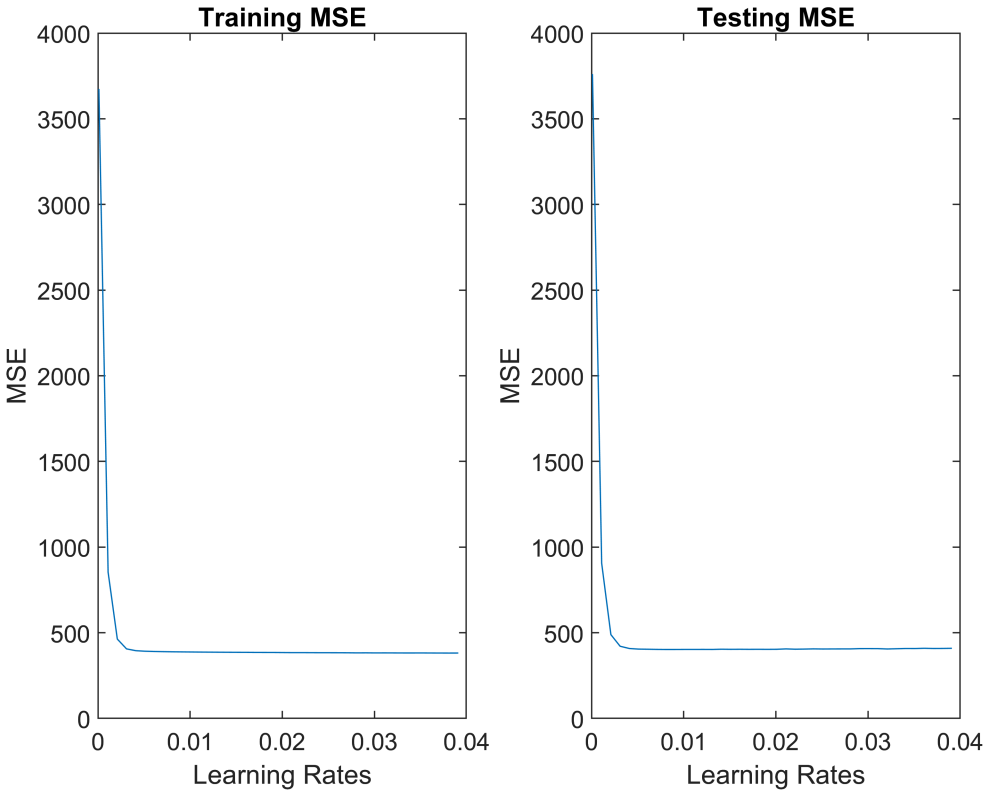
Part 1, Problem 2 - Train-Test Split

Split the indices of the NES dataset; create the appropriate variables.

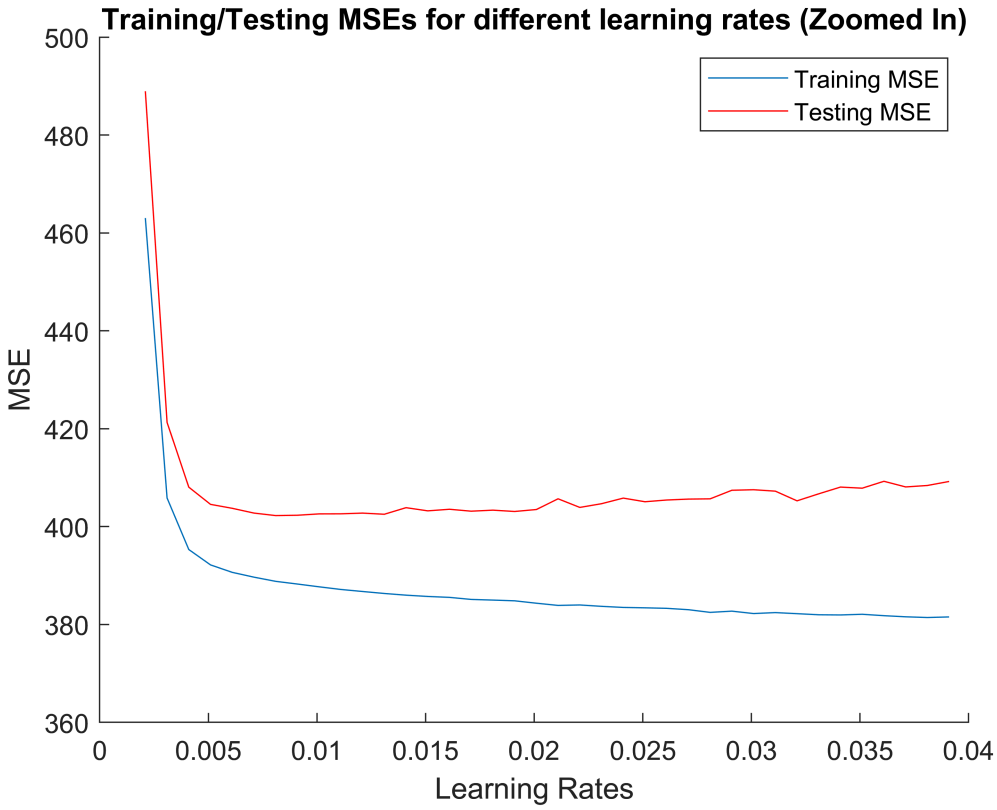
```
nes_sampleSize = height(nes_dataset);

nes_trainingProportion = 0.75;
nes_trainingSize = floor(nes_sampleSize * nes_trainingProportion); % (floor to guarantee an int

nes_trainingIndices = randsample( [1:nes_sampleSize] , nes_trainingSize , false );
                                                                    % false == w/o replacement
nes_testingIndices  = setdiff( [1:nes_sampleSize] , nes_trainingIndices );

nes_testingSize = length(nes_testingIndices);
```

Part 1, Problem 3 - Boosting + Visualising Effects of Hyperparameter-Tuning

```
% Define our trees to be stumps.
t = templateTree('MaxNumSplits',1);

% Preallocate arrays. (size == length of learningRate vector)
mdls_boost              = cell( 1,length(learningRate) );
mdls_boost_training_MSE = nan(  1,length(learningRate) );
mdls_boost_testing_MSE  = nan(  1,length(learningRate) );

% Now generate all of the models and get their respective training/testing MSEs.
for i = 1:length(learningRate)

    % Generate the individual boosting-based model...
    mdls_boost{1,i} = fitrensemble(nes_dataset(nes_trainingIndices,:), ...
        'biden ~ female + age + educ + dem + rep', ...
        'Method', 'LSBoost', ...
        'NumLearningCycles', 1000, ...
        'Learners', t, ...
        'Resample', 'on', 'Replace', 'off', 'FResample', 0.50, ... # to make the gradient boost
        'NumBins', 50, ... % slightly speeds up the computation for the sake of sanity
        'LearnRate', learningRate(i));

    % ...and then calculate training/testing MSE of that individual model.
    predictedValues_training = predict( mdls_boost{1,i}, nes_dataset(nes_trainingIndices,nes_in
    predictedValues_testing  = predict( mdls_boost{1,i}, nes_dataset(nes_testingIndices,nes_inp

    mdls_boost_training_MSE(1,i) = MSE( predictedValues_training, nes_dataset{nes_trainingIndic
    mdls_boost_testing_MSE(1,i)  = MSE( predictedValues_testing , nes_dataset{nes_testingIndice

end

% Plot the MSE across the hyperparameter space...
figure;
subplot(1,2,1);
plot(learningRate, mdls_boost_training_MSE);
title('Training MSE');
xlabel('Learning Rates');
ylabel('MSE');
subplot(1,2,2);
plot(learningRate, mdls_boost_testing_MSE);
title('Testing MSE');
xlabel('Learning Rates');
ylabel('MSE');
```




```
% ...hard to compare those side-by-side plots, so let's cut out the first two models,
% which gives extreme MSE values. This will make it easier to visualise and compare.
figure;
hold on;
plot(learningRate(3:end), mdls_boost_training_MSE(3:end));
title('Training/Testing MSEs for different learning rates (Zoomed In)')
ylim([360, 500]);
yticks([360, 380, 400, 420, 440, 460, 480, 500]);
xlabel('Learning Rates');
ylabel('MSE');
plot(learningRate(3:end), mdls_boost_testing_MSE(3:end), 'r');
ylim([360, 500]);
yticks([360, 380, 400, 420, 440, 460, 480, 500]);
legend('Training MSE', 'Testing MSE');
hold off;
```



Part 1, Problem 4 - Boosting Model (i.e. stump-stacking)

Repeat the boosting problem from before, albeit without doing a parameter sweep (and with some tweaks).

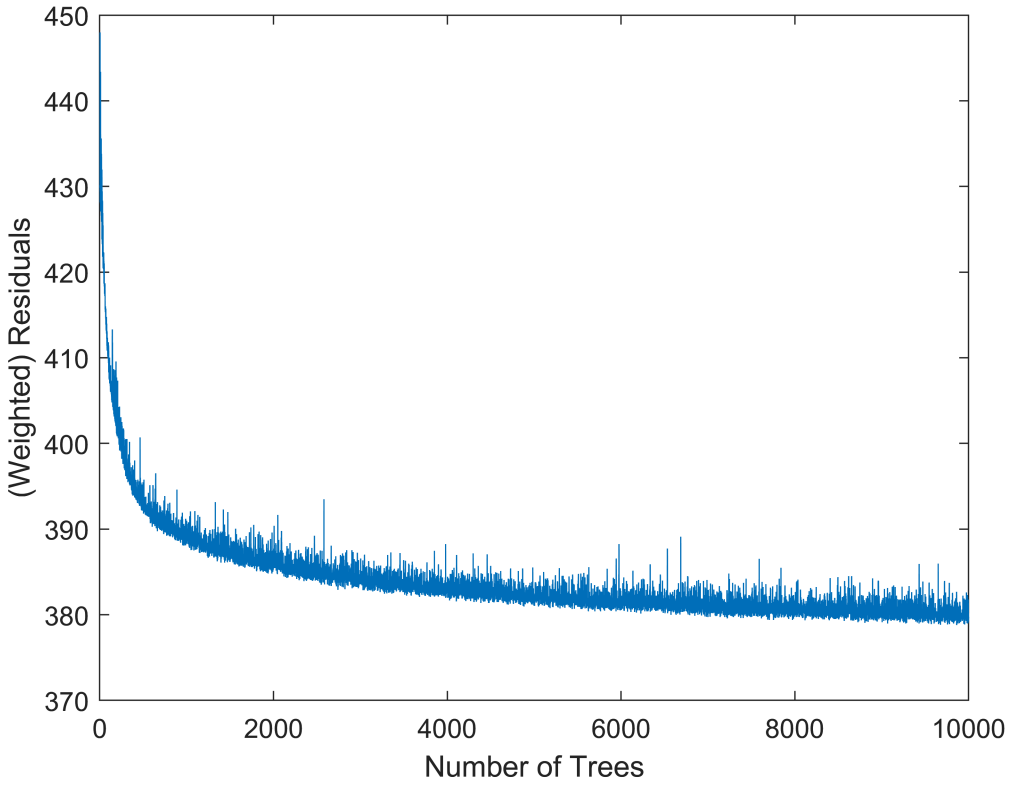
```
% We're using the same t from before; t = templateTree('MaxNumSplits',1);

% Generate the model.
mdl2_boost = fitrensemble(nes_dataset(nes_trainingIndices,:), ...
    'biden ~ female + age + educ + dem + rep', ...
    'Method', 'LSBoost', ...
    'NumLearningCycles', 10000, ... % 10,000 trees since the problem asks for over 1,000
    'Learners', t, ...
    'Resample', 'on', 'Replace', 'off', 'FResample', 0.50, ... # to make the gradient boost int
    'NumBins', 50, ... % slightly speeds up the computation for the sake of sanity
    'LearnRate', 0.01);

% Get its training/testing MSE.
predictedValues_training2 = predict( mdl2_boost, nes_dataset(nes_trainingIndices,nes_inputVar)
predictedValues_testing2 =  predict( mdl2_boost, nes_dataset(nes_testingIndices ,nes_inputVar)
mdl2_boost_training_MSE = MSE( predictedValues_training2, nes_dataset{nes_trainingIndices,'bide
mdl2_boost_testing_MSE  = MSE( predictedValues_testing2 , nes_dataset{nes_testingIndices ,'bide

% Compare this model's test set MSE to model #11's test set MSE (from the previous problem).
% I choose model #11 (with learningRate==0.0101) because it's usually one of the best models
% (out of the 40 models and in terms of test set MSE performance).
mdl2_boost_testing_MSE
```

```
mdl2_boost_testing_MSE = 412.9761
```

```
mdls_boost_testing_MSE(11)
```

```
ans = 402.5990
```

```
% Since I chose a 10,000 tree model instead of a 1,000 tree model, how different is the
% 10,000-tree model as opposed to the 1,000 tree model?
% Not directly related to the problem, but I just wanted to check with an extra plot.
% (NOTE: these are not technically MSEs, but are related)
figure;
plot(mdl2_boost.FitInfo);
xlabel('Number of Trees');
ylabel('(Weighted) Residuals');
```




```
% Now for a more "direct" comparison, using "weighted residuals" again.
mdls_boost{11}.FitInfo(1000)
```

```
ans = 388.6023
```

```
mdl2_boost.FitInfo(1000)
```

```
ans = 390.9824
```

Overall, these two models (with similar hyperparameters) seem to be fairly similar to one another, albeit the one with 10,000 trees seems to have overfit just slightly (hence the slightly higher test MSE).

Part 1, Problem 5 - Bagging Model (i.e. Random Forest minus the random)

```
% t_bag defaults to non-stump, random-forest behaviour.
% The non-stumpness is fine, but we don't want random-forest behaviour in our bagging, so
% we set 'NumVariablesTosample' to 'all'.
t_bag = templateTree('NumVariablesToSample', 'all');

% Generate a bagging model.
mdl_bag = fitrensemble(nes_dataset(nes_trainingIndices,:), ...
    'biden ~ female + age + educ + dem + rep', ...
    'Method', 'bag', ...
    'NumLearningCycles', 1000, ...
    'Learners', t_bag, ...
    'NumBins', 50); % slightly speeds up the computation for the sake of sanity

% Get its training/testing MSE.
predictedValues_training_bag = predict( mdl_bag, nes_dataset(nes_trainingIndices,nes_inputVar)
predictedValues_testing_bag  = predict( mdl_bag, nes_dataset(nes_testingIndices ,nes_inputVar)
mdl_bag_training_MSE = MSE( predictedValues_training_bag, nes_dataset{nes_trainingIndices,'bide
mdl_bag_testing_MSE  = MSE( predictedValues_testing_bag , nes_dataset{nes_testingIndices ,'bide

% Compare to one of the boosting models from before
mdl_bag_testing_MSE
```

```
mdl_bag_testing_MSE = 455.6252
```

```
mdls_boost_testing_MSE(11) % Chose model #11 from the parameter sweep because it's one of the b
```

```
ans = 402.5990
```

So far, bagging seems to be worse than gradient boosting.

Part 1, Problem 6 - Random Forest Model (i.e. Bagging plus the random)

```
% t_rf will use the default template (non-stumped and randomly selecting a subset of the variab
t_rf = templateTree();

% Generate a random forest model.
mdl_rf = fitrensemble(nes_dataset(nes_trainingIndices,:), ...
    'biden ~ female + age + educ + dem + rep', ...
    'Method', 'bag', ...
    'NumLearningCycles', 1000, ...
    'Learners', t_rf, ...
    'NumBins', 50); % slightly speeds up the computation for the sake of sanity

% Get its training/testing MSE.
predictedValues_training_rf = predict( mdl_rf, nes_dataset(nes_trainingIndices,nes_inputVar) );
predictedValues_testing_rf  = predict( mdl_rf, nes_dataset(nes_testingIndices ,nes_inputVar) );
mdl_rf_training_MSE = MSE( predictedValues_training_rf, nes_dataset{nes_trainingIndices,'biden'
mdl_rf_testing_MSE  = MSE( predictedValues_testing_rf,  nes_dataset{nes_testingIndices ,'biden'

% More comparing:
mdl_rf_testing_MSE
```

```
mdl_rf_testing_MSE = 408.0701
```

```
mdl_bag_testing_MSE
```

```
mdl_bag_testing_MSE = 455.6252
```

```
mdls_boost_testing_MSE(11)
```

```
ans = 402.5990
```

According to this, random forest is just barely better than one of the best boosting models. That means random forest was also way better than bagging (because boosting was also way better than bagging). That makes sense because random forest is less prone to overfitting. More specifically, random forest allows more variation between trees, because they don't all consider the same feature space.

Part 1, Problem 7 - Linear Model

```
% Generate linear model.
mdl_linear = fitlm(nes_dataset, nes_formula);

% Get MSEs.
predictedValues_training_linear = predict(mdl_linear, nes_dataset(nes_trainingIndices,nes_input
predictedValues_testing_linear  = predict(mdl_linear, nes_dataset(nes_testingIndices ,nes_input
mdl_linear_training_MSE = MSE( predictedValues_training_linear, nes_dataset{nes_trainingIndices
mdl_linear_testing_MSE  = MSE( predictedValues_testing_linear , nes_dataset{nes_testingIndices,

% Get model's performance.
mdl_linear_testing_MSE
```

```
mdl_linear_testing_MSE = 394.1601
```

Yields the lowest value among all of our models so far.

Part 1, Problem 8 - And the winner is...

The linear model is the best, because it has the lowest MSE out of all the models. This makes sense, given that the linear model is also the only parametric approach, meaning that it leverages more information (in the form of assumptions) in order to yield a better fit (but of course, only if the assumptions are on point).

Part 2 - SVMs

Part 2, Problem 1 - Train-Test Split

```
oj_sampleSize = height(oj_dataset);
oj_trainingSize = 800;

oj_trainingIndices = randsample( [1:oj_sampleSize] , oj_trainingSize , false );
oj_testingIndices  = setdiff( [1:oj_sampleSize] , oj_trainingIndices );
```

Part 2, Problem 2 - Support Vector Classifier

```
% Specify the cost hyperparameter.
% (MATLAB calls this hyperparameter the 'box constraint', just FYI)
bc = 0.01;

% Create an SVM.
% (Uses linear kernel function by default.)
mdl_svm = fitcsvm(oj_dataset(oj_trainingIndices,:), oj_formula, 'BoxConstraint', bc);

% Generate training/testing confusion matrices (to gauge performance).
predictedValues_training_svm = predict( mdl_svm, oj_dataset(oj_trainingIndices,oj_inputVar) );
predictedValues_testing_svm  = predict( mdl_svm, oj_dataset(oj_testingIndices ,oj_inputVar) );
trainingConfusion = confusionmat( oj_dataset{oj_trainingIndices,'Purchase'} , predictedValues_t
testingConfusion  = confusionmat( oj_dataset{oj_testingIndices ,'Purchase'} , predictedValues_t
```

As seen from the results returned below, the total error rate on the test set seems to be higher than the error rate on the training set, which is generally as expected.

Part 2, Problem 3 - Performance

How confused are we? Show confusion matrices and total error rates for both training and testing sets.

```
trainingConfusion
```

```
trainingConfusion = 2×2
   386    93
   104   217
```

```
trainingConfusion_totalErrorRate = ...
    ( trainingConfusion(1,2)+trainingConfusion(2,1) ) / ( sum(trainingConfusion,'all') )
```

```
trainingConfusion_totalErrorRate = 0.2463
```

```
testingConfusion
```

```
testingConfusion = 2×2
   138    36
    40    56
```

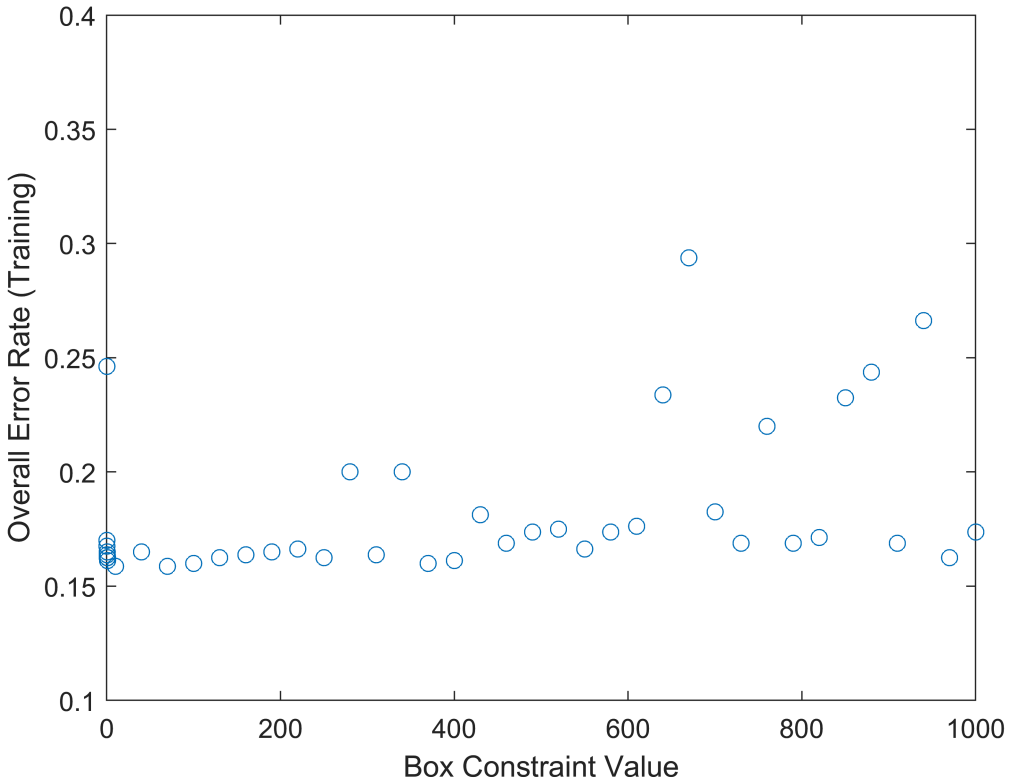
```
testingConfusion_totalErrorRate = ...
    ( testingConfusion(1,2)+testingConfusion(2,1) ) / ( sum(testingConfusion,'all') )
```

```
testingConfusion_totalErrorRate = 0.2815
```

Part 2, Problem 4 - Find the Best Cost Hyperparameter

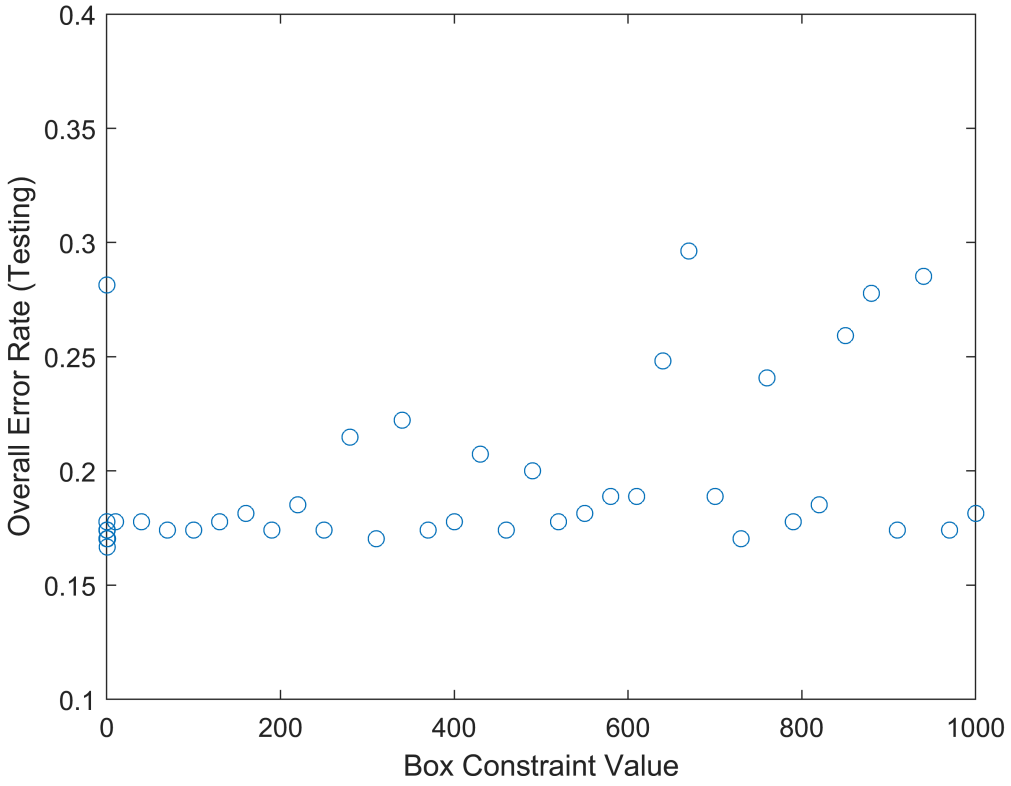
```
% Define the parameter sweep.
bc = [0.01:0.09:1, 10:30:1000]; % length == 46

% Preallocate arrays for our many models.
many_svms               = cell(1,length(bc));
training_confusion_svm  = cell(1,length(bc));
testing_confusion_svm   = cell(1,length(bc));
training_ERR_svm        = nan(1,length(bc));
testing_ERR_svm         = nan(1,length(bc));
```

```
% Generate SVMs for our parameter sweep and get performance for each SVM.
for i = 1:length(bc)

    many_svms{1,i} = fitcsvm(oj_dataset(oj_trainingIndices,:), oj_formula, 'BoxConstraint', bc(

    predictedValues_training_svm = predict( many_svms{1,i}, oj_dataset(oj_trainingIndices,oj_ir
    predictedValues_testing_svm  = predict( many_svms{1,i}, oj_dataset(oj_testingIndices ,oj_ir
    training_confusion_svm{1,i}  = confusionmat( oj_dataset{oj_trainingIndices,'Purchase'} , pr
    testing_confusion_svm{1,i}   = confusionmat( oj_dataset{oj_testingIndices ,'Purchase'} , pr

    trainConfuse = training_confusion_svm{1,i};
    testConfuse  = testing_confusion_svm{1,i};
    training_ERR_svm(i) = ( trainConfuse(1,2)+trainConfuse(2,1) ) / ( sum(trainConfuse,'all') )
    testing_ERR_svm(i)  = ( testConfuse(1,2) +testConfuse(2,1)  ) / ( sum(testConfuse ,'all') )
end

% Plot the performance of our many SVMs.
figure;
plot(bc,training_ERR_svm, 'o');
xlabel('Box Constraint Value');
ylabel('Overall Error Rate (Training)');
ylim([0.10, 0.40]);
```




```
figure;
plot(bc,testing_ERR_svm, 'o');
xlabel('Box Constraint Value');
ylabel('Overall Error Rate (Testing)');
```

```
ylim([0.10, 0.40]);
```



Part 2, Problem 5 - Comparing error rates between the two models

```
% best testing error rate? check the first 10 values
testing_ERR_svm(1:10)
```

```
ans = 1×10
    0.2815    0.1778    0.1741    0.1704    0.1741    0.1704    0.1667    0.1704 ···
```

```
% occurs at i==7; what is the hyperparameter of the 7th SVM model?
bc(7)
```

```
ans = 0.5500
```

```
% get other numbers
training_confusion_svm{1,7}
```

```
ans = 2×2
   425    54
    77   244
```

```
testing_confusion_svm{1,7}
```

```
ans = 2×2
   153    21
    24    72
```

```
training_ERR_svm(7)
```

```
ans = 0.1638
```

```
testing_ERR_svm(7)
```

```
ans = 0.1667
```

The original SVM has a cost of 0.01, and our best SVM also has a cost of around 0.55. The original SVM's error rates are overall much higher (on training and testing sets) than the "best" SVM's error rates (~25% error rates versus ~16.5% error rates). That means our optimally-tuned SVM made only 60% of the mistakes (when classifying which brand of OJ customers likely purchased) that our original SVM made, which can be a fairly significant gain given larger scales of data.