Kyle Decker
Machine Learning
HW # 3

Problem 1 /

set $\{x_n\} \Rightarrow \vec{x} = \sum_n a_n \vec{x}_n$

where $a_n \geq 0$ and $\sum_n a_n = 1$

Set $\{y_m\} \Rightarrow \vec{y} = \sum_m b_m \vec{y}_m$

where $b_m \geq 0$ and $\sum_m b_m = 1$

$f(\vec{x}) = w^T x_n + w_0 > 0$

$f(\vec{y}) = w^T y_m + w_0 < 0$

at point of intersection: $f(\vec{x} \text{ intersect } \vec{y}) = \sum_n a_n (w^T x_n + w_0) = \sum_m b_m (w^T y_m + w_0)$

a_n and b_m must be non-negative so this is not possible, as $f(\vec{x} \text{ intersect } \vec{y})$ can not be negative and positive.

Problem 2/

"Cost sensitive" Adaboost

$$R^{train} = \sum_i g_i e^{-(M\lambda)_i}$$

Coordinate descent on $R^{train}(\lambda)$

$$j_t \in \arg\max_j \left[- \frac{dR^{train}(\lambda_t + \alpha e_j)}{d\alpha} \Big|_{\alpha=0} \right]$$

$$= \arg\max_j \left[-\frac{\partial}{\partial\alpha} \left[\sum_i g_i e^{-(M(\lambda_t + \alpha e_j))_i} \right] \Big|_{\alpha=0} \right]$$

$$= \arg\max_j \left[-\frac{\partial}{\partial\alpha} \left[\sum_i g_i e^{-(M\lambda_t)_i - \alpha M_{ij}} \right] \Big|_{\alpha=0} \right]$$

$$= \arg\max_j \left[\sum_i g_i M_{ij} e^{-(M\lambda_t)_i} \right]$$

$$d_{t,i} = g_i e^{-(M\lambda_t)_i} / Z_t$$

where $Z_t = \sum_i g_i e^{-(M\lambda_t)_i}$

Line search along direction j:

$$0 = -\sum_{i: M_{ij_t}=1} d_{t,i} e^{-\alpha} - \sum_{i: M_{ij_t}=-1} -d_{t,i} e^{\alpha} \Big/ d_t$$

$$= -e^{-\alpha_t} \underbrace{\sum_{i: M_{ij_t}=1} d_{t,i}}_{d_+} + e^{\alpha} \underbrace{\sum_{i: M_{ij_t}=-1} d_{t,i}}_{d_-}$$

$$= -e^{-\alpha_t} d_+ + e^{\alpha_t} d_-$$

$$d^{\alpha_t} d_+ = e^{\alpha_t} d_-$$

$$\frac{d_+}{d_-} = e^{2\alpha}$$

$$\frac{1}{2} \ln \left(\frac{d_+}{d_-} \right) = \alpha_t \Rightarrow \alpha_t = \frac{1}{2} \ln \left(\frac{1-d_-}{d_-} \right)$$

Finding direction:

$$j_t \in \arg\max_j \sum_i g_i M_{ij} e^{-(M\lambda_t)_i}$$

$$j_t \in \arg\max_j \sum_i M_{ij} d_{t,i}$$

$$j_t \in \arg\max_j (d_t^T M)_j$$

$\downarrow$

replace w/ weak learning algorithm

$$j_t \in \arg\min_j \sum_{i: M_{ij}=-1} d_{t,i}$$

Algorithm:

$$d_{1,i} = \frac{1}{n} g_i \quad \text{for } i = 1 \cdots n$$

$$\bar{\lambda}_1 = \bar{0}$$

for $t = 1 \dots T$

Step 1: $\arg\min_j \left[\sum_{i: M_{ij}=-1} d_{t,i} \right.$

where $d_- = \sum_{M_{ij_t}=-1} d_{t,i}$

Step 2: $\alpha_t = \frac{1}{2} \ln \left(\frac{1-d_-}{d_-} \right)$

Weight update: $d_{t+1,i} = g_i e^{-(M\lambda_t)_i} / Z_{t+1}$

$\propto d_{t,i}\, e^{-\alpha_t}$ if $M_{ij_t} = 1$

$\propto d_{t,i}\, e^{\alpha_t}$ if $M_{ij_t} = -1$

Problem 3/ If weak learning assumption holds, AdaBoost's misclassification error decays exponentially fast:

$$\frac{1}{m}\sum_{i=1}^{m} 1\left[y_i \neq H(x_i)\right] \leq e^{-2\gamma_{WLA}^2 T}$$

There will be zero training error if:

$$e^{-2\gamma_{WLA}^2 T} \leq \frac{1}{m}$$

$$\ln\left(e^{-2\gamma_{WLA}^2 T}\right) \leq \ln\left(\frac{1}{m}\right)$$

$$-2\gamma_{WLA}^2 T \leq \ln\left(\frac{1}{m}\right)$$

$$\cancel{2\gamma_{WLA}^2 T \leq \ln(m)}$$

$$2\gamma_{WLA}^2 T \geq \ln(m)$$

$$T \geq \frac{\ln(m)}{2\gamma_{WLA}^2}$$

As long as there are at least $\frac{\ln(m)}{2\gamma_{WLA}^2}$ iterations of ~~a~~ updates, Adaboost will find a linear classifier that classifies the dataset perfectly.

Problem 3/

Symmetric data set, twin $-h(x)$ of each classifier $h(x)$ is included. If the weak learning assumption does not hold the AdaBoost can not find a classifier that classifies the symmetric dataset perfectly all of the time.

$$M_{ij} = \left[\begin{array}{cccccc} 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 & -1 & 1 \\ \hline -1 & 1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & -1 & 1 & -1 \end{array}\right]$$

Problem 4/

I decided to use the letter-recognition data from the UCI machine learning repository. I performed classification of B letters vs O letters.

Code: Please see attached matlab code and generated outputs

Preprocessing: Simply reading the data in, mapping the letter B to class 1 and letter O to class 0. Random permutation of examples.

Algorithms:
1) Logistic Regression
2) SVM → optimize Kernel
3) CART
4) Random Forest → optimize # of trees
5) Boosted Trees

Using nested Cross-validation I evaluated the following parameters
for SVM: Kernel = [Linear, Polynomial, Gaussian]

for RF: # trees = [3, 10, 30, 60, 100]

ROC Curves: Please see the generated matlab plots.

AUC values for features and algorithms : please see generated outputs in code.

Discussion:

(i) The algorithms performed similarly in terms of judging their performance from the AUC. SVM, RF, and Boosted trees performed slightly better than logistic regression and CART.

(ii) The classification of B vs. D in the dataset I used was not terribly difficult, i.e. a linear decision boundary performed quite well in the classification. Thus the algorithms all performed vey well in result.

(iii) Combining features and training a model resulted in much better results than the individual features alone.

Table of Contents


```
% Kyle Decker
% Machine Learning
% HW 3

close all
clc
```

Data Loading

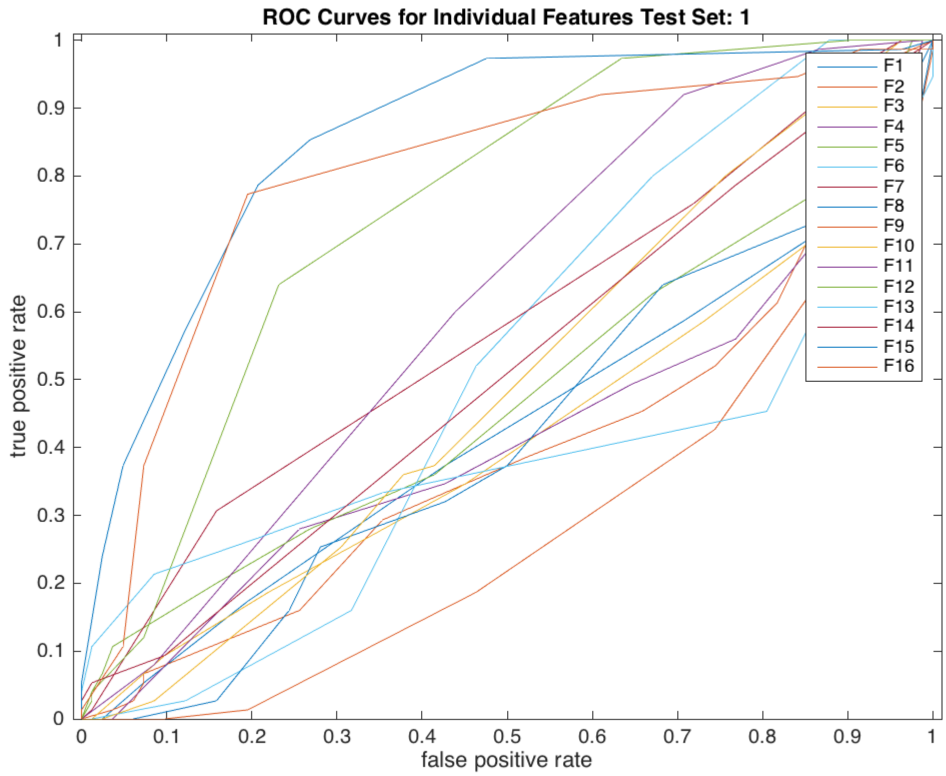
```
f = fopen('letter-recognition.data.txt');
data0 = textscan(f,'%s %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
 %f', 20000, 'Delimiter',',');
fclose(f);
data_all = cell2mat(data0(:,2:end));
letter = cell2mat(data0{1,1});
class_all = zeros(size(letter,1),1);
class_all_B = (letter=='B'); % Classify B vs D
B_ind = find(class_all_B);
class_all_D = (letter=='D');
D_ind = find(class_all_D);

data = cat(1,data_all(B_ind,:),data_all(D_ind,:));
class = zeros(size(data,1),1);
class(1:size(B_ind,1))=1; % B = class 1
class = (class==1); % make logical


% data0 = csvread('wine.data.txt');
% data = data0(:,2:end);
% class = data0(:,1)>2; % Make binary, wine 1 vs wine 2&3

% Random permutation of the dataset
rand_i = randperm(size(class,1));
data_perm = data(rand_i,:);
class_perm = class(rand_i);
```

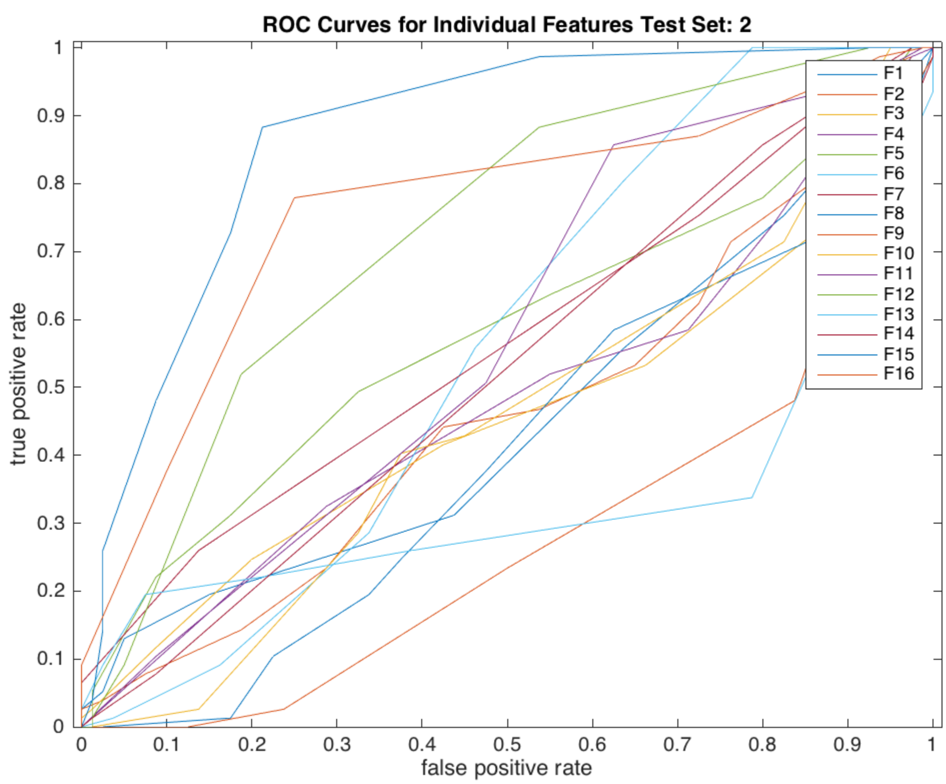
Using Features themselves for ROC

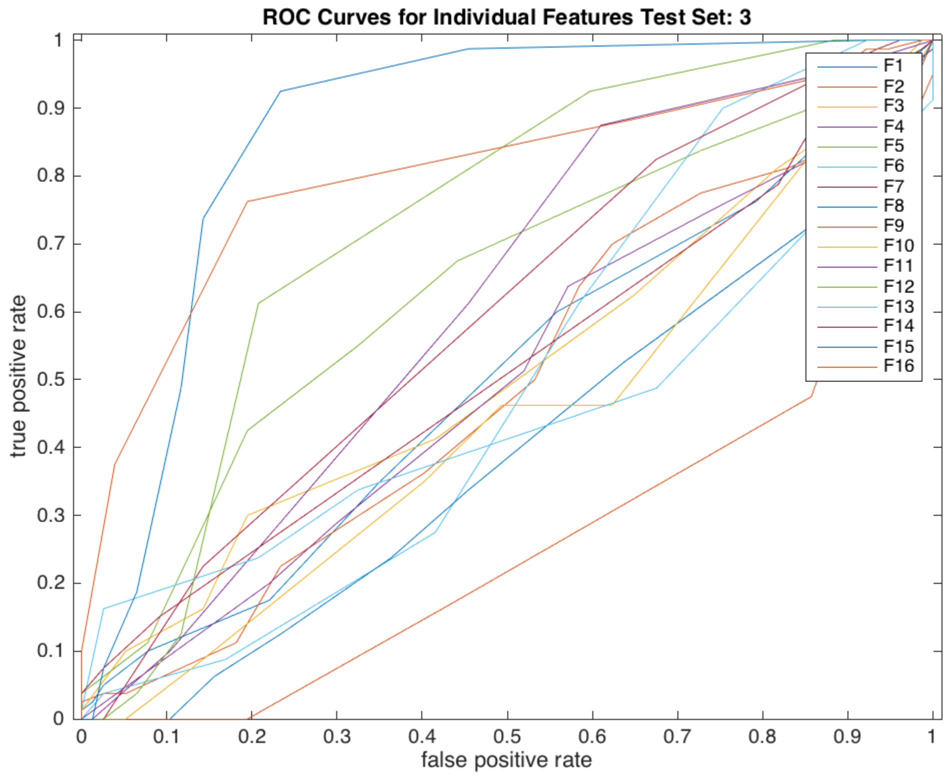
```
AUROC_all = zeros(10,size(data_perm,2));
```

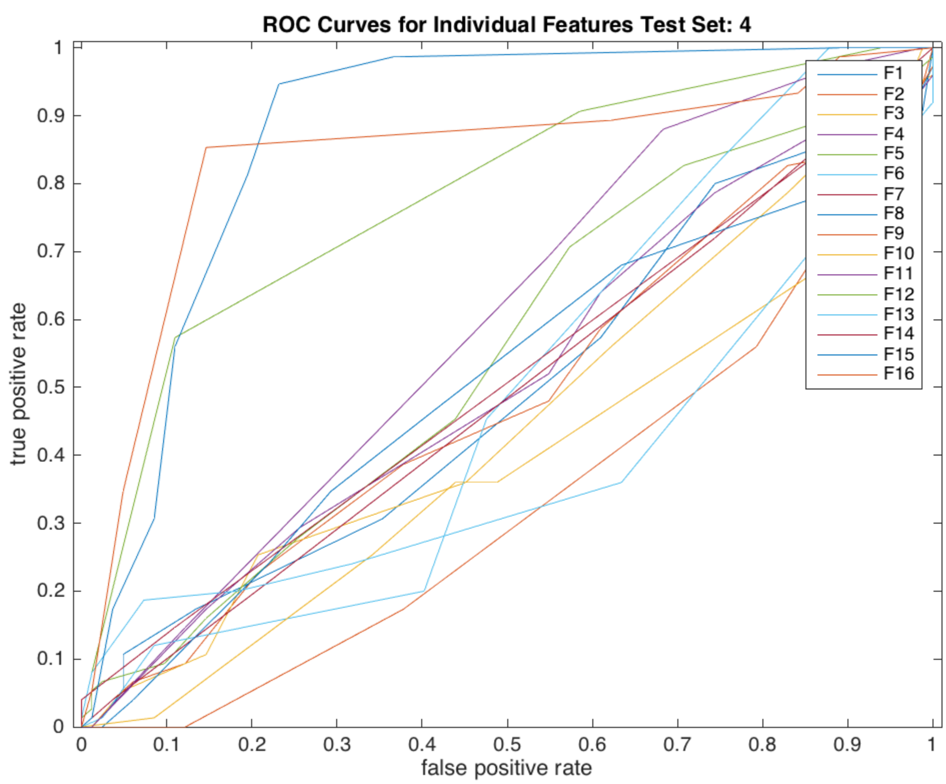
```
for test_ind = 1:10
    p = 0.9; % proportion of the dataset for training
    m = size(data,1);

    % Circ shift to create new test and training set
    data_perm = circshift(data_perm,round((1-p)*m),1);
    class_perm = circshift(class_perm,round((1-p)*m),1);

    X = data_perm(1:round(p*m),:);
    Y = class_perm(1:round(p*m));
    Xtest = data_perm(round(p*m)+1:end,:);
    Ytest = class_perm(round(p*m)+1:end);

    figure;
    Yscores = Xtest(:,1);
    [Xfeat,Yfeat,Tfeat,AUCfeat] = perfcurve(Ytest,Yscores,'true');
    AUROC_all(test_ind,1) = AUCfeat;
    plot(Xfeat, Yfeat)
    hold on
    for feat_ind = 2:size(Xtest,2)
        %feat_ind = 1;
        Yscores = Xtest(:,feat_ind);
        [Xfeat,Yfeat,Tfeat,AUCfeat] = perfcurve(Ytest,Yscores,'true');
        AUROC_all(test_ind,feat_ind) = AUCfeat;
        plot(Xfeat, Yfeat)
    end
    xlabel('false positive rate');
    ylabel('true positive rate');
    title_str = ['ROC Curves for Individual Features Test Set:
 ',num2str(test_ind)];
    title(title_str)
    axis([-0.01,1.01,0,1.01])
    legend('F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', ...
        'F9', 'F10', 'F11', 'F12', 'F13', 'F14', 'F15', 'F16')
    hold off

end

fprintf('AUROC (mean +/- standard deviation) for\n');
fprintf('Feature 1: %f +/- %f\n',
 mean(AUROC_all(:,1)),std(AUROC_all(:,1)) );
fprintf('Feature 2: %f +/- %f\n',
 mean(AUROC_all(:,2)),std(AUROC_all(:,2)) );
fprintf('Feature 3: %f +/- %f\n',
 mean(AUROC_all(:,3)),std(AUROC_all(:,3)) );
fprintf('Feature 4: %f +/- %f\n',
 mean(AUROC_all(:,4)),std(AUROC_all(:,4)) );
fprintf('Feature 5: %f +/- %f\n',
 mean(AUROC_all(:,5)),std(AUROC_all(:,5)) );
fprintf('Feature 6: %f +/- %f\n',
 mean(AUROC_all(:,6)),std(AUROC_all(:,6)) );
fprintf('Feature 7: %f +/- %f\n',
 mean(AUROC_all(:,7)),std(AUROC_all(:,7)) );
```

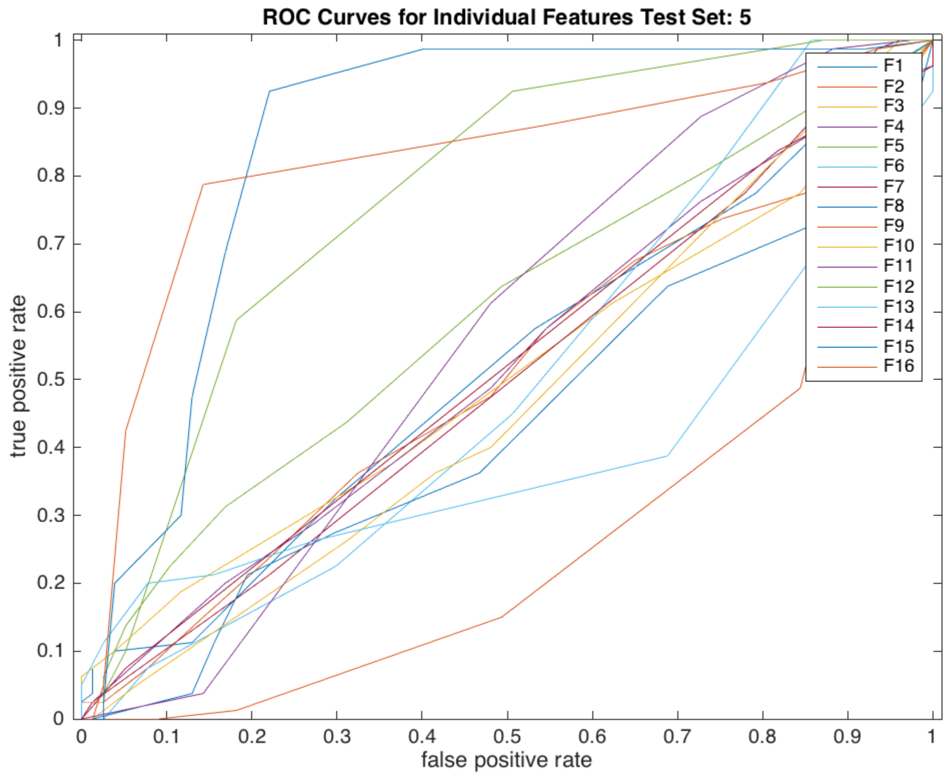
```
fprintf('Feature 8: %f +/- %f\n',
 mean(AUROC_all(:,8)),std(AUROC_all(:,8)) );
fprintf('Feature 9: %f +/- %f\n',
 mean(AUROC_all(:,9)),std(AUROC_all(:,9)) );
fprintf('Feature 10: %f +/- %f\n',
 mean(AUROC_all(:,10)),std(AUROC_all(:,10)) );
fprintf('Feature 11: %f +/- %f\n',
 mean(AUROC_all(:,11)),std(AUROC_all(:,11)) );
fprintf('Feature 12: %f +/- %f\n',
 mean(AUROC_all(:,12)),std(AUROC_all(:,12)) );
fprintf('Feature 13: %f +/- %f\n',
 mean(AUROC_all(:,13)),std(AUROC_all(:,13)) );
fprintf('Feature 14: %f +/- %f\n',
 mean(AUROC_all(:,14)),std(AUROC_all(:,14)) );
fprintf('Feature 15: %f +/- %f\n',
 mean(AUROC_all(:,15)),std(AUROC_all(:,15)) );
fprintf('Feature 16: %f +/- %f\n',
 mean(AUROC_all(:,16)),std(AUROC_all(:,16)) );

AUROC (mean +/- standard deviation) for
Feature 1: 0.487335 +/- 0.052990
Feature 2: 0.477438 +/- 0.040073
Feature 3: 0.481606 +/- 0.051506
Feature 4: 0.488458 +/- 0.044363
Feature 5: 0.571973 +/- 0.048383
Feature 6: 0.522496 +/- 0.045899
Feature 7: 0.563577 +/- 0.043376
Feature 8: 0.436689 +/- 0.048794
Feature 9: 0.299221 +/- 0.032979
Feature 10: 0.466499 +/- 0.048161
Feature 11: 0.587595 +/- 0.035790
Feature 12: 0.748445 +/- 0.035816
Feature 13: 0.382636 +/- 0.040675
Feature 14: 0.531480 +/- 0.025521
Feature 15: 0.872255 +/- 0.040407
Feature 16: 0.804822 +/- 0.027788
```

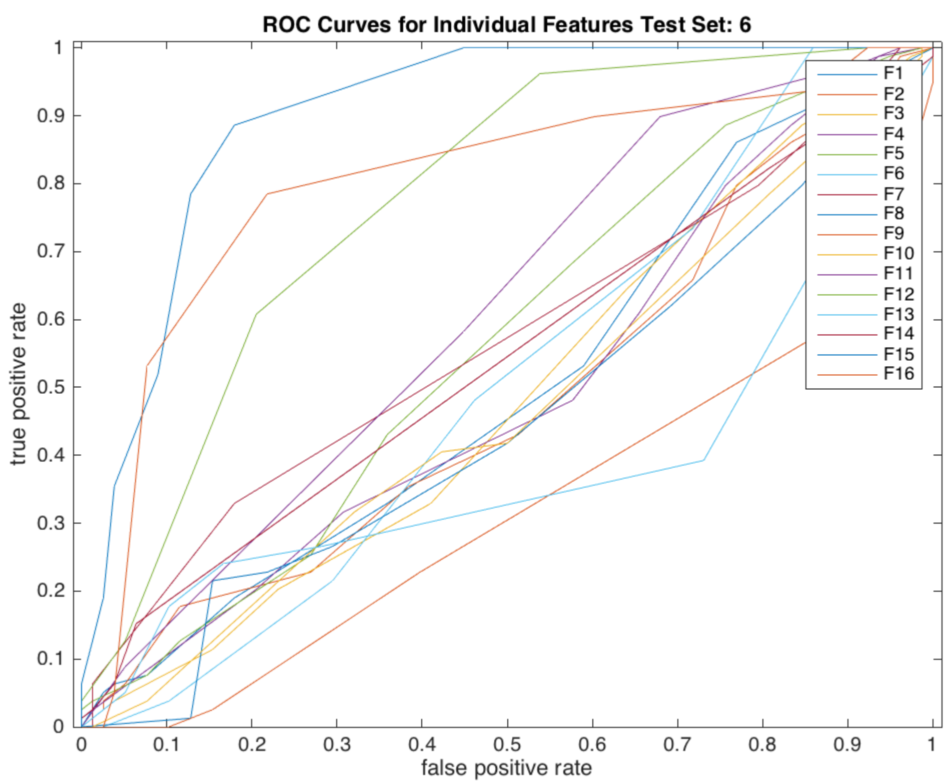

ROC Curves for Individual Features Test Set: 1
true positive rate
false positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16

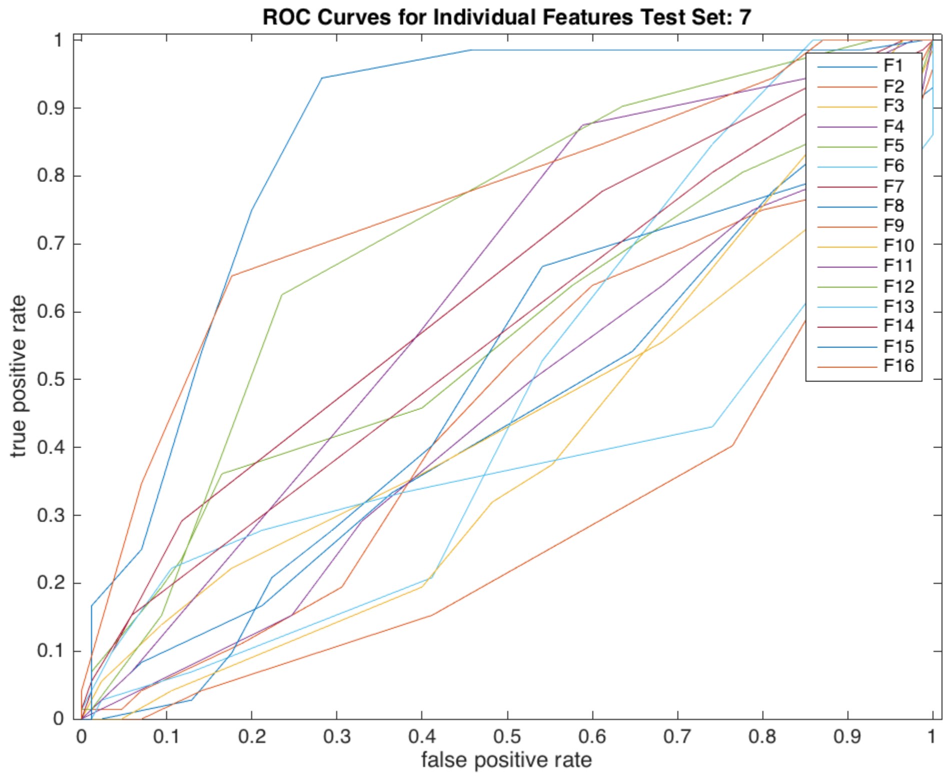

ROC Curves for Individual Features Test Set: 2
true positive rate
false positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16

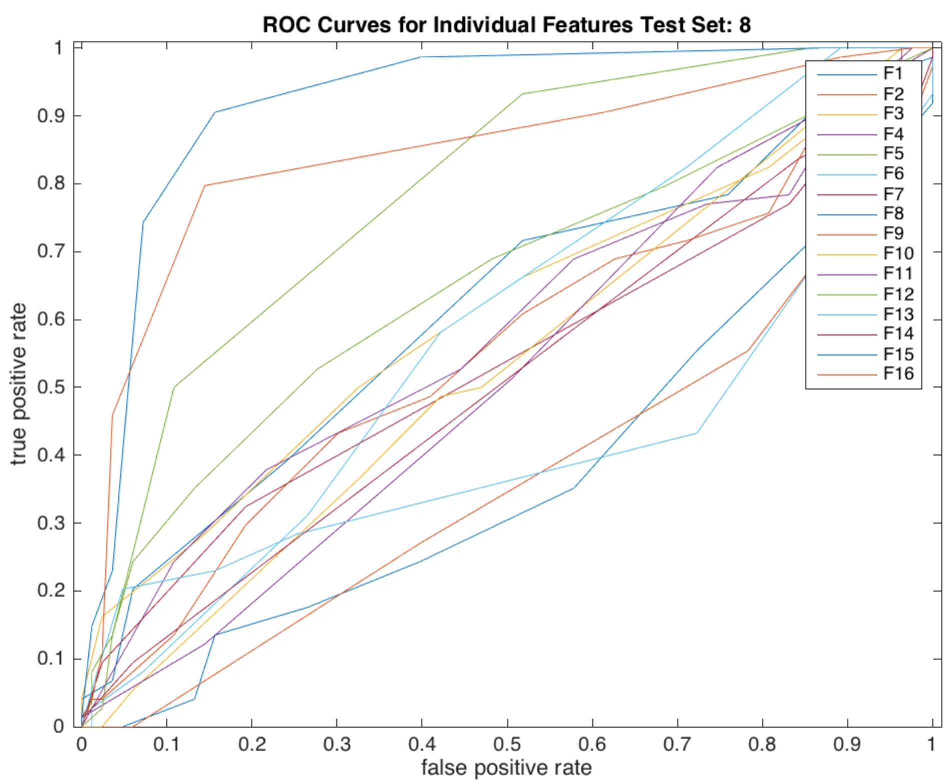

ROC Curves for Individual Features Test Set: 3
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16

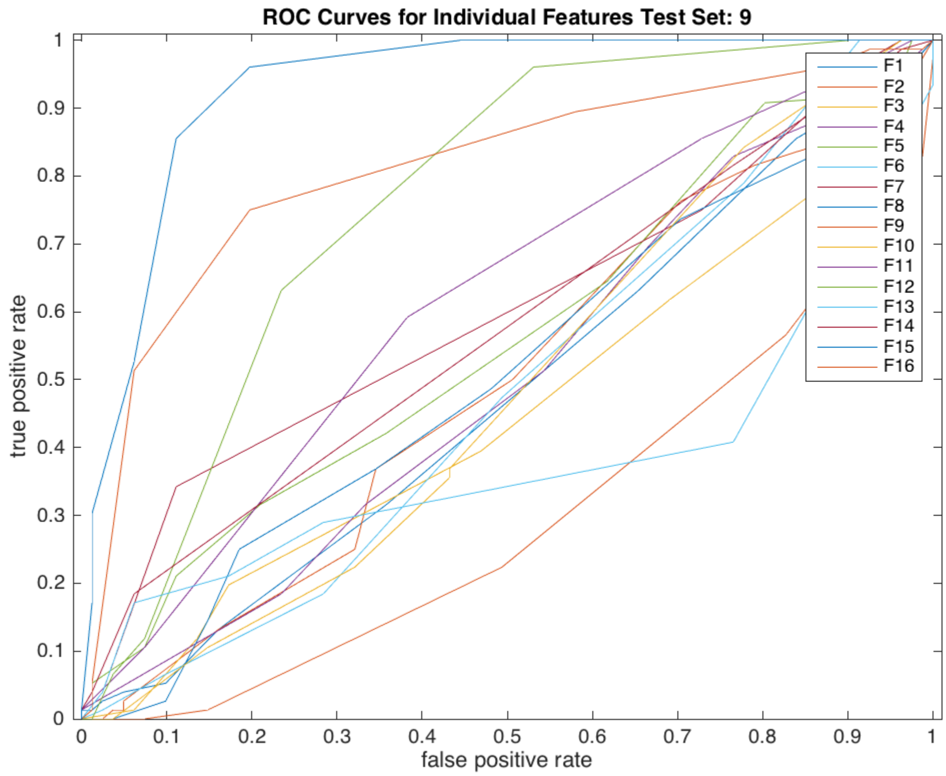

ROC Curves for Individual Features Test Set: 4
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16

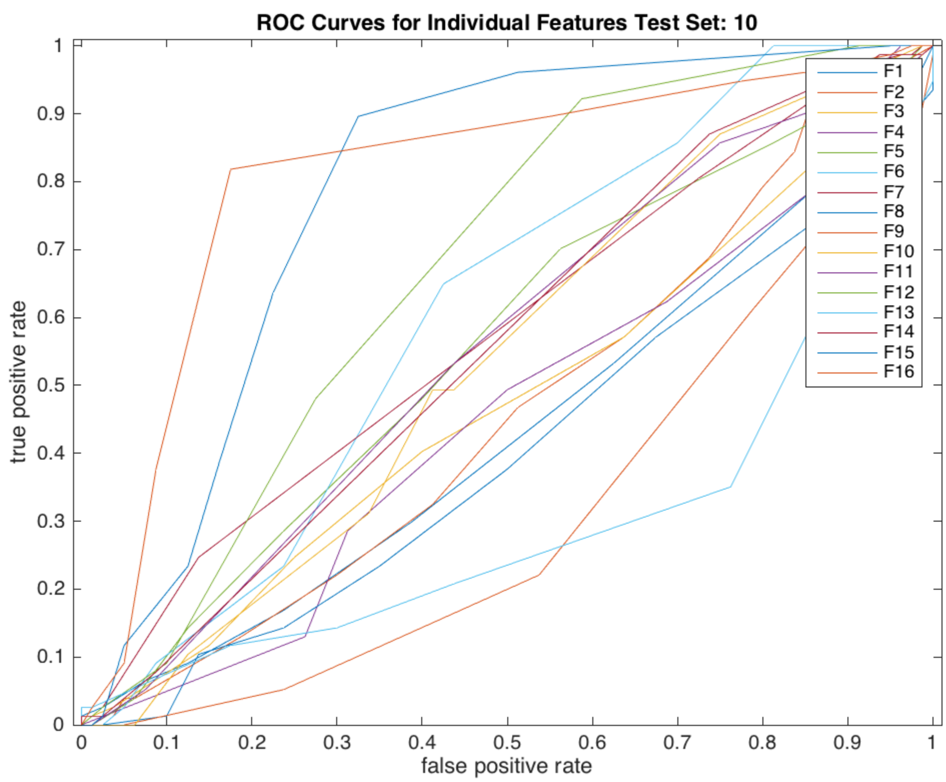

ROC Curves for Individual Features Test Set: 5
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16


ROC Curves for Individual Features Test Set: 6
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16


ROC Curves for Individual Features Test Set: 7
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16


ROC Curves for Individual Features Test Set: 8
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16


ROC Curves for Individual Features Test Set: 9
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16


ROC Curves for Individual Features Test Set: 10
true positive rate
false positive rate
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16

For 5 different Algorithms

```
AUCglm = zeros(10,1);
AUCsvm = zeros(10,1);
AUCcart = zeros(10,1);
AUCrf = zeros(10,1);
AUCbt = zeros(10,1);

for test_ind = 1:10

    fprintf('Performing Testing using Fold %d of 10 \n',test_ind);
    p = 0.9; % proportion of the dataset for training
    m = size(data,1);

    % Circ shift to create new test and training set
    data_perm = circshift(data_perm,round((1-p)*m),1);
    class_perm = circshift(class_perm,round((1-p)*m),1);

    X = data_perm(1:round(p*m),:);
    Y = class_perm(1:round(p*m));
    Xtest = data_perm(round(p*m)+1:end,:);
    Ytest = class_perm(round(p*m)+1:end);
    % Generalized Linear Model (Logistic Regression)

    glmModel = fitglm(X,
 Y, 'Distribution', 'binomial', 'Link', 'logit');
    Yscores = predict(glmModel, Xtest); % these are the posterior
 probabilities
    % of class 1 for the test data

    % ... compute the standard ROC curve and the AUROC
    [Xglm, Yglm, Tglm, AUCglm(test_ind)] = perfcurve(Ytest,
 Yscores, 'true');
```

```
Performing Testing using Fold 1 of 10

Performing Testing using Fold 2 of 10

Performing Testing using Fold 3 of 10

Performing Testing using Fold 4 of 10

Performing Testing using Fold 5 of 10

Performing Testing using Fold 6 of 10

Performing Testing using Fold 7 of 10

Performing Testing using Fold 8 of 10

Performing Testing using Fold 9 of 10

Performing Testing using Fold 10 of 10
```

Nested CV to choose kernel for SVM

```
   AUCsvm_nested = zeros(10,3);
   for val_ind = 1:10
       pn = 0.9; % proportion of data for training
       mn = size(X,1);

       X = circshift(X,round((1-pn)*mn),1);
       Y = circshift(Y,round((1-pn)*mn),1);

       X_nested = X(1:round(pn*mn),:);
       Y_nested = Y(1:round(pn*mn));
       X_nested_test = X(round(pn*mn)+1:end,:);
       Y_nested_test = Y(round(pn*mn)+1:end);


       for k_index = 1:3
           % Define K parameter values
           if k_index == 1
               K = 'linear';
           elseif k_index == 2
               K = 'polynomial';
           elseif k_index == 3
               K = 'rbf';
           end


           % Support Vector Machine (SVM)

           svmModel = fitcsvm(X_nested, Y_nested, 'Standardize',
true, 'KernelFunction', K);
           svmModel = fitPosterior(svmModel);
           [~, Yscores] = predict(svmModel, X_nested_test);

           % ... compute the standard ROC curve and the AUROC
           [Xsvm_nested, Ysvm_nested, Tsvm_nested,
AUCsvm_nested(val_ind,k_index)] = perfcurve(Y_nested_test, Yscores(:,
2), 'true');

       end
   end

   % Pick the Paramter with Highest Mean AUROC across validation
folds
   AUCsvm_nested_mean = mean(AUCsvm_nested,1);
   k_best_index = find(AUCsvm_nested_mean ==
max(AUCsvm_nested_mean));

   if k_best_index == 1
       K_best_svm = 'linear'
   elseif k_best_index == 2
       K_best_svm = 'polynomial'
   elseif k_best_index == 3
```

```
        K_best_svm = 'rbf'
    end



    % Support Vector Machine (SVM) for Test Set

    svmModel = fitcsvm(X, Y, 'Standardize', true, 'KernelFunction',
 K_best_svm);
    svmModel = fitPosterior(svmModel);
    [~, Yscores] = predict(svmModel, Xtest);

    % ... compute the standard ROC curve and the AUROC
    [Xsvm, Ysvm, Tsvm, AUCsvm(test_ind)] = perfcurve(Ytest, Yscores(:,
 2), 'true');
```

K_best_svm =

rbf

K_best_svm =

rbf

K_best_svm =

rbf

K_best_svm =

rbf

K_best_svm =

rbf

K_best_svm =

rbf

K_best_svm =

polynomial

K_best_svm =

rbf

K_best_svm =

rbf

K_best_svm =

rbf

Classification Tree (CART)

```
    ctreeModel = fitctree(X, Y);
    [~, Yscores, ~, ~] = predict(ctreeModel, Xtest);

    % ... compute the standard ROC curve and the AUROC
    [Xcart, Ycart, Tcart, AUCcart(test_ind)] = perfcurve(Ytest,
 Yscores(:, 2), 'true');
```

Nested CV to choose number of trees for RF

```
    AUCrf_nested = zeros(10,5);
    k_values = [3,10,30,60,100];
    for val_ind = 1:10
        pn = 0.9; % proportion of data for training
        mn = size(X,1);

        X = circshift(X,round((1-pn)*mn),1);
        Y = circshift(Y,round((1-pn)*mn),1);

        X_nested = X(1:round(pn*mn),:);
        Y_nested = Y(1:round(pn*mn));
        X_nested_test = X(round(pn*mn)+1:end,:);
        Y_nested_test = Y(round(pn*mn)+1:end);


        for k_index = 1:length(k_values)

            % Random Forest (RF)

            rfModel = fitensemble(X_nested, Y_nested, 'Bag',
 k_values(k_index), 'Tree', 'Type', 'Classification');
```

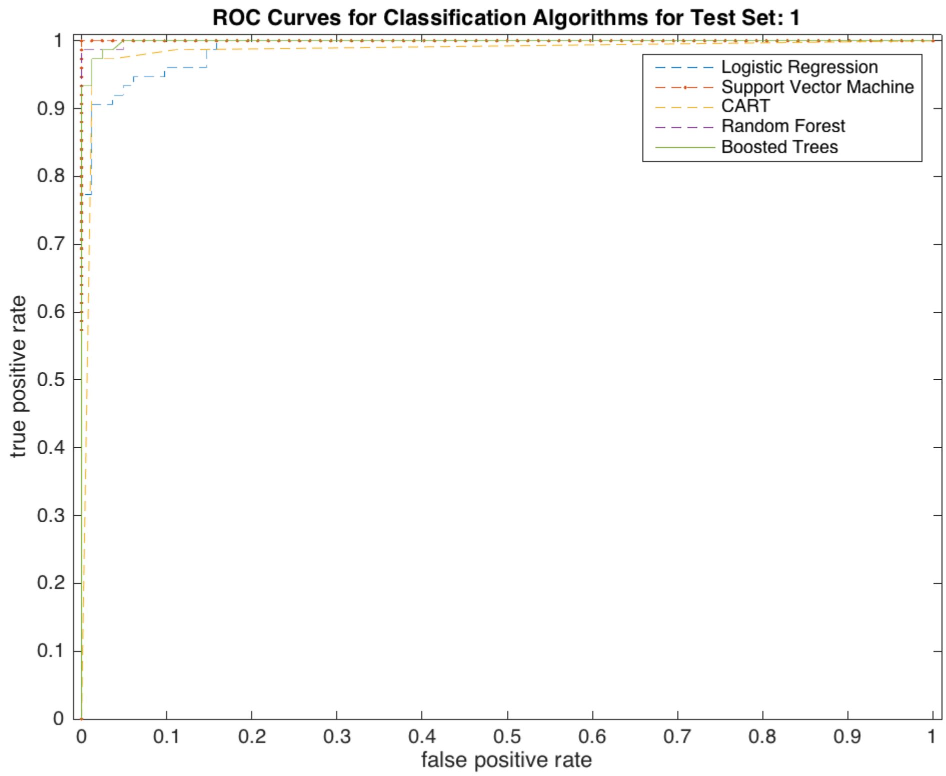
```
            [~, Yscores] = predict(rfModel, X_nested_test);

            % ... compute the standard ROC curve and the AUROC
            [Xrf_nested, Yrf_nested, Trf_nested,
 AUCrf_nested(val_ind,k_index)] = perfcurve(Y_nested_test, Yscores(:,
 2), 'true');

        end
    end

    % Pick the Paramter with Highest Mean AUROC across validation
 folds
    AUCrf_nested_mean = mean(AUCrf_nested,1);
    k_best_index = find(AUCrf_nested_mean == max(AUCrf_nested_mean));

    K_best_rf = k_values(k_best_index)

    % Random Forest (RF)

    rfModel = fitensemble(X, Y, 'Bag',
 K_best_rf, 'Tree', 'Type', 'Classification');
    [~, Yscores] = predict(rfModel, Xtest);


    % ... compute the standard ROC curve and the AUROC
    [Xrf, Yrf, Trf, AUCrf(test_ind)] = perfcurve(Ytest, Yscores(:,
 2), 'true');
```

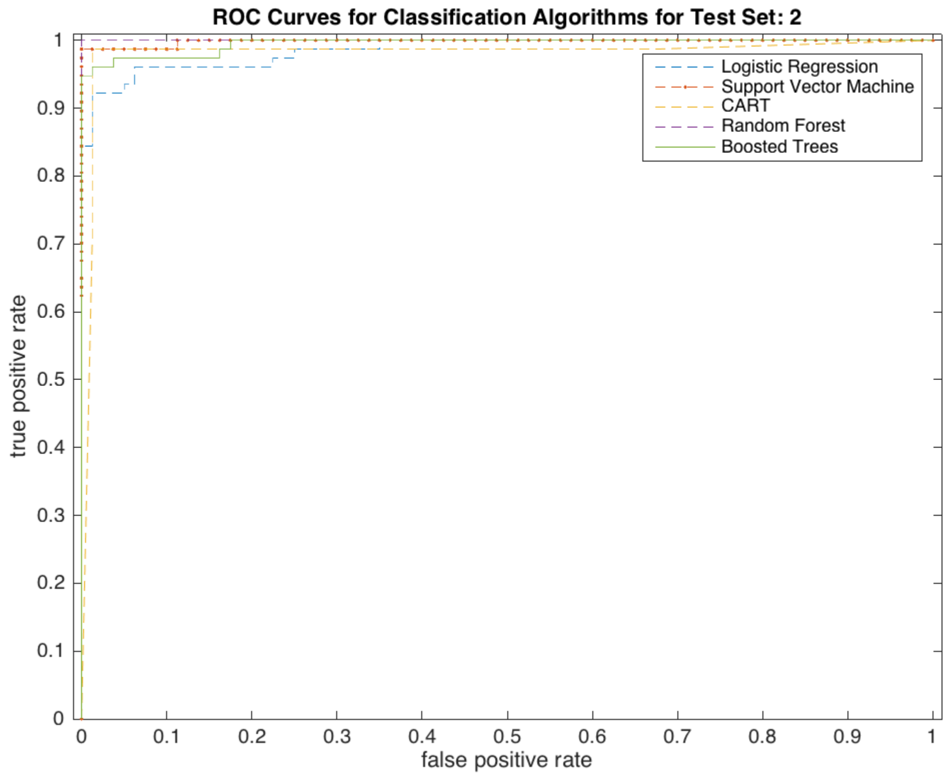
```
K_best_rf =

   100


K_best_rf =

   100


K_best_rf =

   100


K_best_rf =

   100


K_best_rf =
```

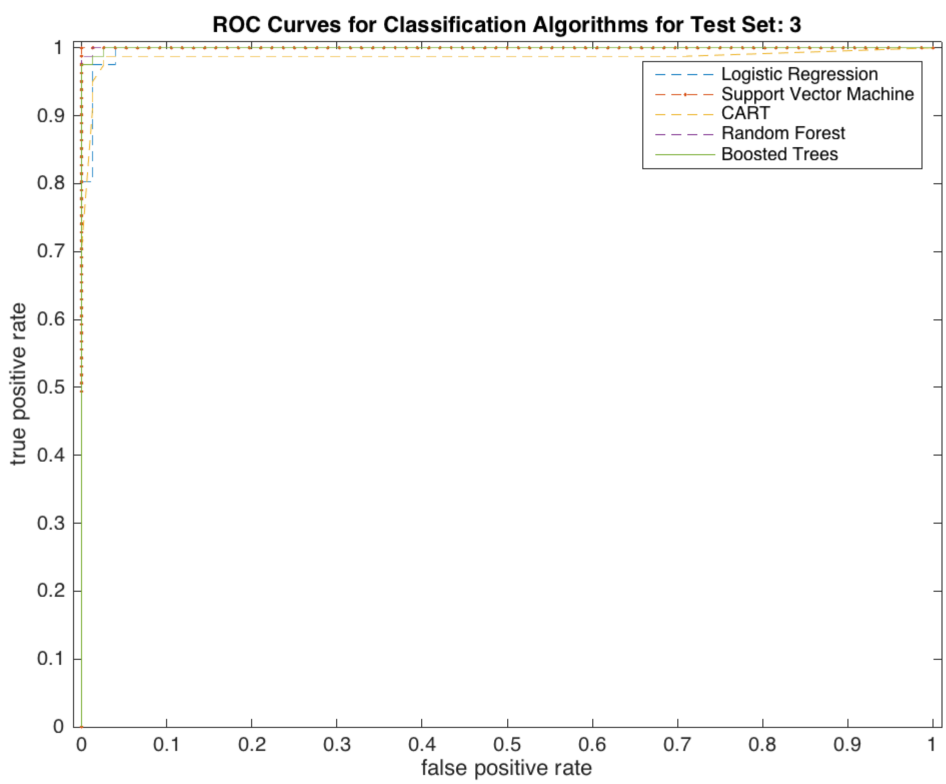
```
   100


K_best_rf =

    60


K_best_rf =

    30


K_best_rf =

    60


K_best_rf =

   100


K_best_rf =

   100
```

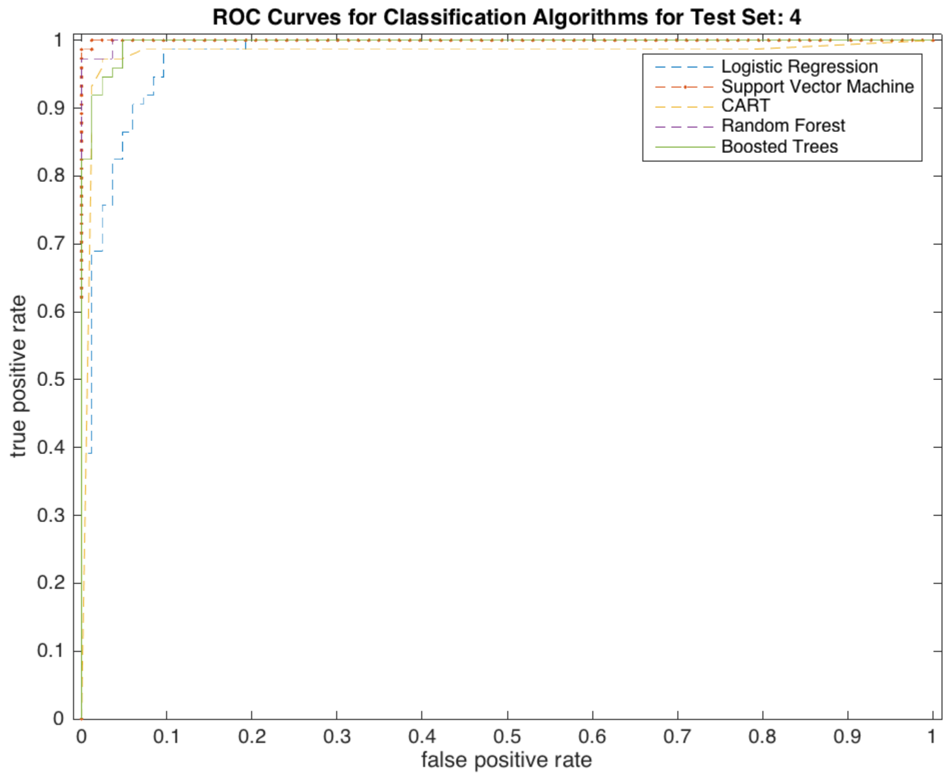
```
    % Boosted Trees

    btModel = fitensemble(X, Y, 'AdaBoostM1', 100, 'Tree');
    [~, Yscores] = predict(btModel, Xtest);

    % ... compute the standard ROC curve and the AUROC
    [Xbt, Ybt, Tbt, AUCbt(test_ind)] = perfcurve(Ytest,
 sigmf(Yscores(:, 2), [1 0]), ...
        'true');

    % ROC Curves
    figure;
    plot(Xglm, Yglm,'--')
    hold on
    plot(Xsvm, Ysvm,'--.')
    plot(Xcart, Ycart,'--')
    plot(Xrf, Yrf,'--')
    plot(Xbt, Ybt)
    legend('Logistic Regression', 'Support Vector
 Machine', 'CART', ...
```

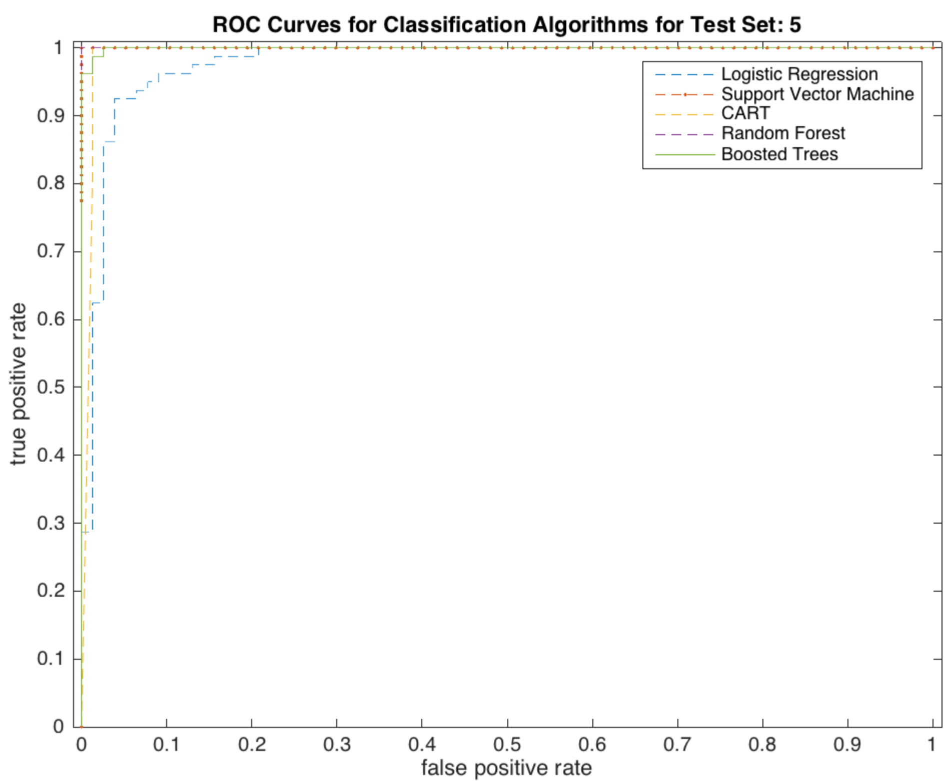
```
        'Random Forest', 'Boosted Trees')
    xlabel('false positive rate');
    ylabel('true positive rate');
    title_str = ['ROC Curves for Classification Algorithms for Test
Set: ',num2str(test_ind)];
    title(title_str)
    axis([-0.01,1.01,0,1.01])
    hold off
```

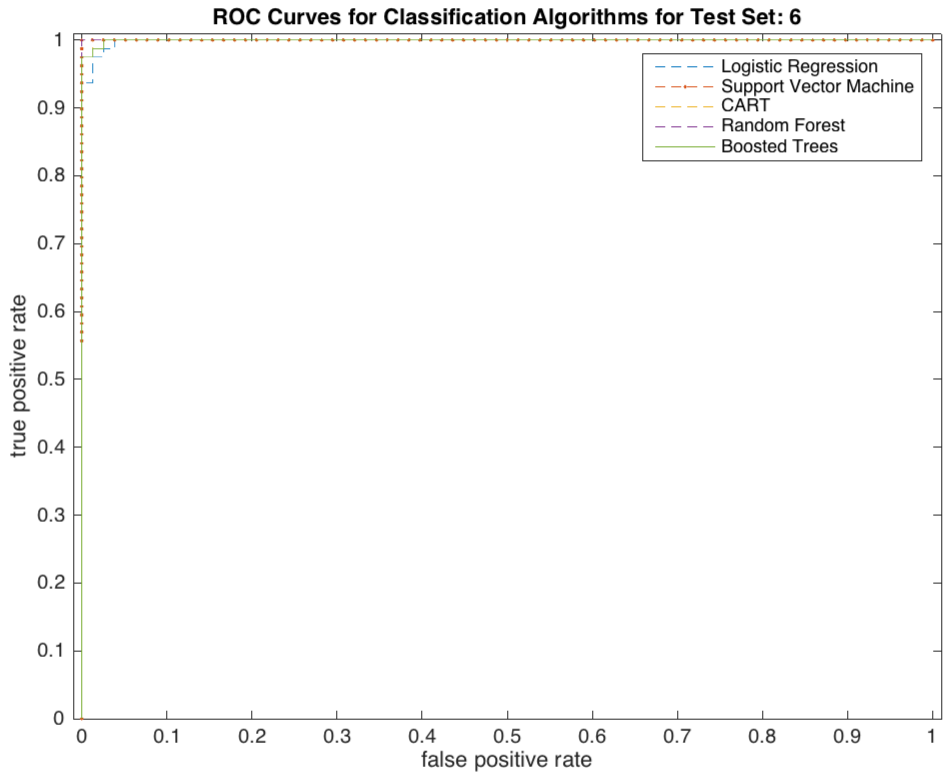


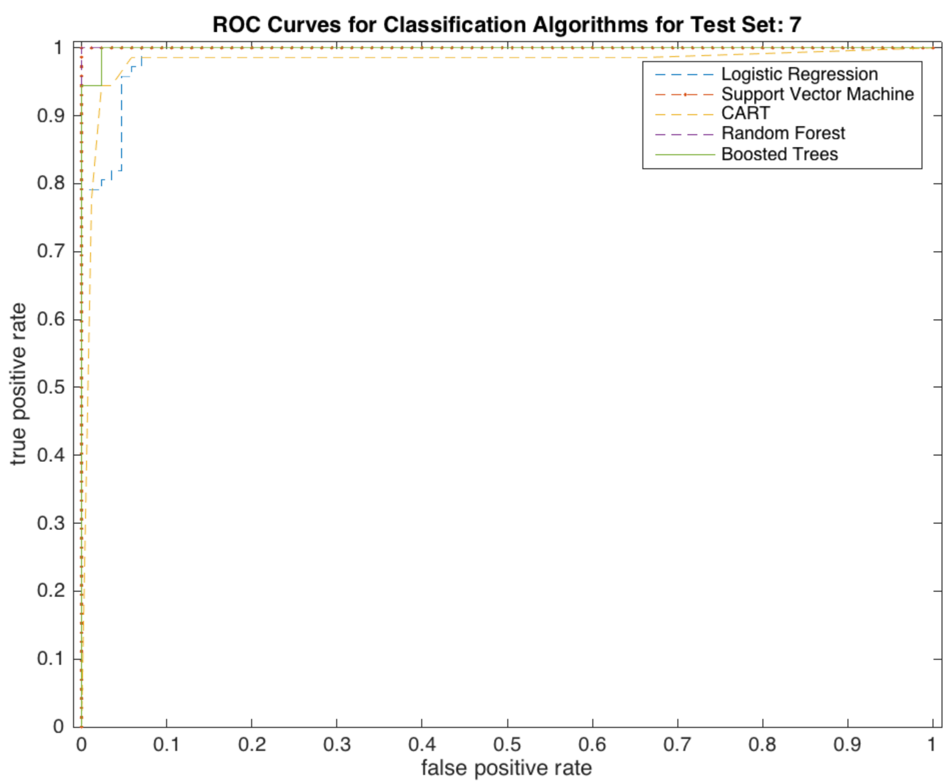

ROC Curves for Classification Algorithms for Test Set: 2
true positive rate
false positive rate
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

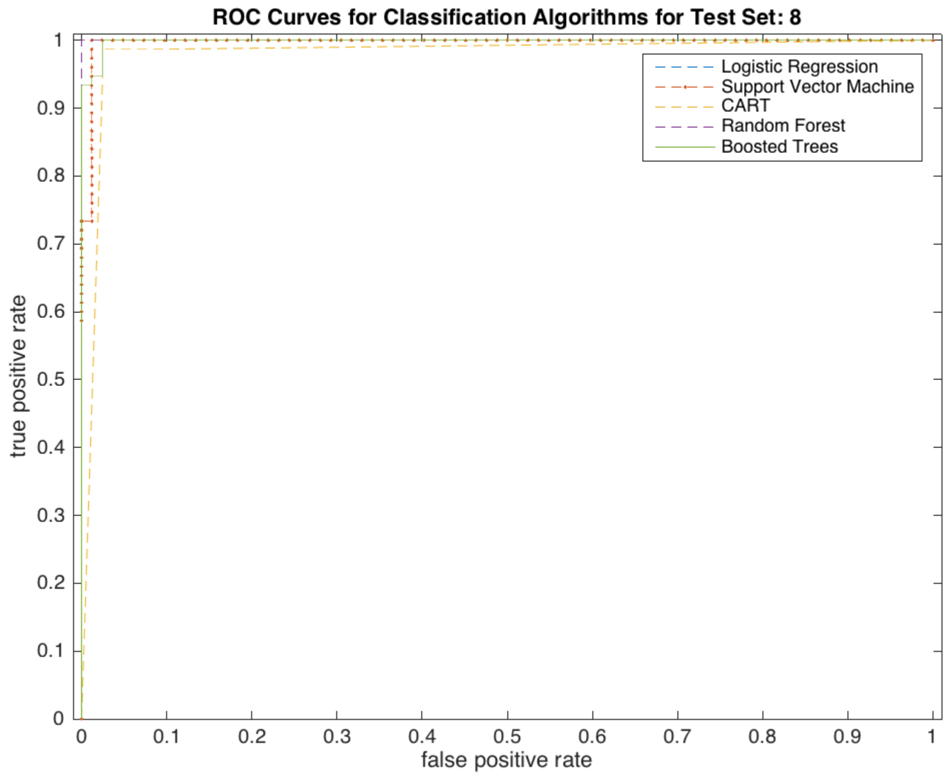

ROC Curves for Classification Algorithms for Test Set: 3
true positive rate
false positive rate
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


ROC Curves for Classification Algorithms for Test Set: 4
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
true positive rate
false positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

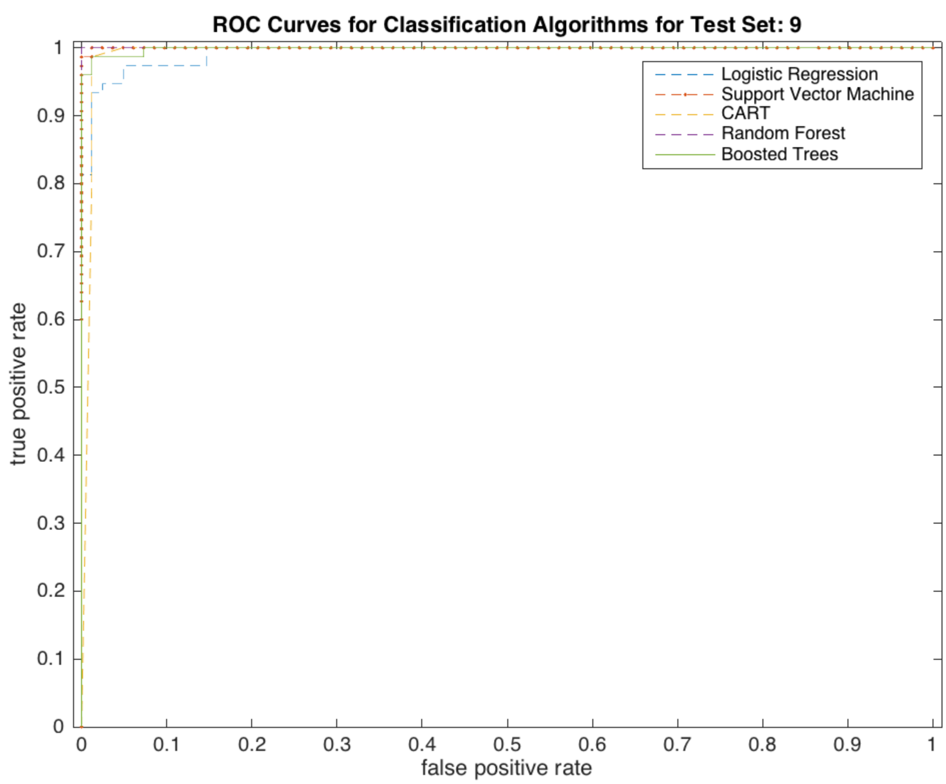

ROC Curves for Classification Algorithms for Test Set: 5
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
true positive rate
false positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


ROC Curves for Classification Algorithms for Test Set: 6
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
true positive rate
false positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

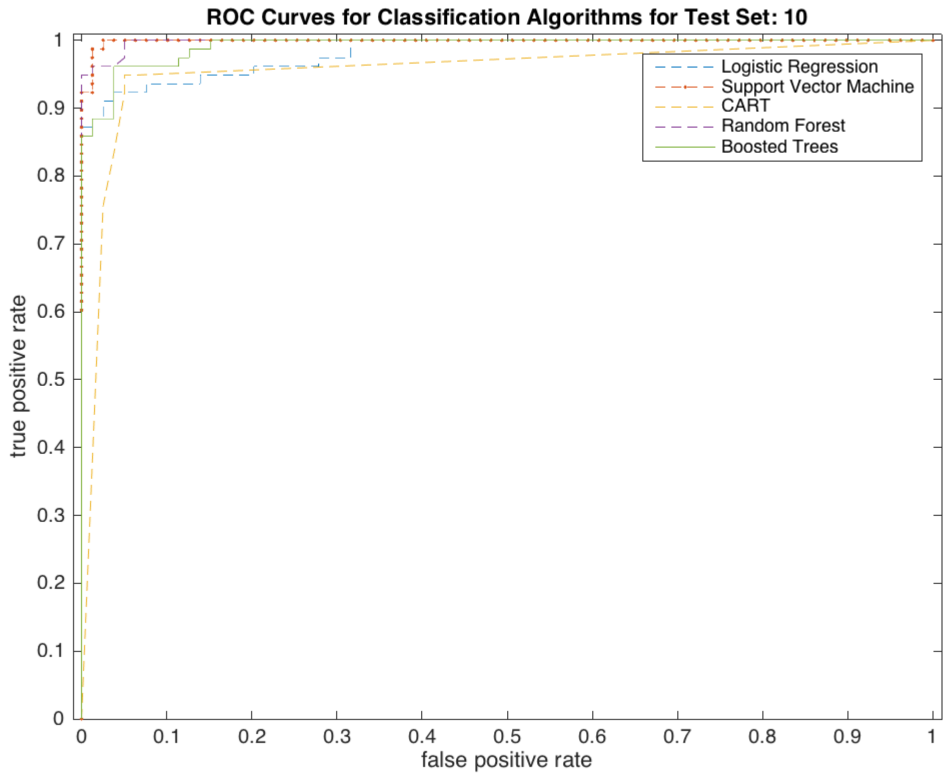

ROC Curves for Classification Algorithms for Test Set: 7
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
true positive rate
false positive rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



ROC Curves for Classification Algorithms for Test Set: 8
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
true positive rate
false positive rate



ROC Curves for Classification Algorithms for Test Set: 9
Logistic Regression
Support Vector Machine
CART
Random Forest
Boosted Trees
true positive rate
false positive rate




```
end
```

AUROC

```
fprintf('AUROC (mean +/- standard deviation) for\n');
fprintf('Logistic Regression: %f +/- %f\n', mean(AUCglm),
 std(AUCglm));
fprintf('Support Vector Machine: %f +/- %f\n', mean(AUCsvm),
 std(AUCsvm));
fprintf('CART: %f +/- %f\n', mean(AUCcart),std(AUCcart));
fprintf('Random Forest: %f +/- %f\n', mean(AUCrf),std(AUCrf));
fprintf('Boosted Trees: %f +/- %f\n', mean(AUCbt),std(AUCbt));

return
```

```
AUROC (mean +/- standard deviation) for
Logistic Regression: 0.989124 +/- 0.008126
Support Vector Machine: 0.999383 +/- 0.001067
CART: 0.983270 +/- 0.011873
Random Forest: 0.999618 +/- 0.000672
Boosted Trees: 0.997538 +/- 0.002571
```

Published with MATLAB® R2015b