Machine learning Coursework: MATLAB code PDF

Read CSV file & check missing value

```
%read CSV file
filename = "water_potability.csv";
tbl = readtable(filename,'TextType','String');
TF_checkmissing = sum(ismissing(tbl))
```

```
TF_checkmissing = 1×10
   491     0     0     0   781     0     0   162     0     0
```

Impute mean value in the missing row

```
% 1.)Fill missing with 0
%T1 = fillmissing(tbl,'constant',0)

% 2.)Remove missing value
%T2 = rmmissing(tbl);

%Find mean in each feature before filling
% T3 = rmmissing(tbl);
% func = @mean;
% xx = varfun(func,T3)

% 3.)Fill missing with mean value
A = fillmissing(tbl,'constant',7.0860,'DataVariables',{'ph'});
B = fillmissing(A,'constant',333.22 ,'DataVariables',{'Sulfate'});
T_mean = fillmissing(B,'constant', 66.401 ,'DataVariables',{'Trihalomethanes'});


%Split Data into two sets Train(80) and Test(20)
tb=T_mean;
hpartition = cvpartition(size(tb,1),'Holdout',0.2); % Nonstratified partition
% Extract indices for training and test
trainId = training(hpartition);
testId = test(hpartition);
% Use Indices to parition the matrix
trainData = tb(trainId,:);
testData = tb(testId,:);

X_train = trainData(:,1:9);
y_train = trainData(:,"Potability");
X_test=testData(:,1:9);
y_test=testData{:,"Potability"};

size(X_train)
```

```
ans = 1×2
        2621           9
```

```
% Standardized Data into same scale
```

```
X_train= normalize(X_train);
X_test = normalize(X_test);
X_train
```

X_train = 2621×9 table

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity
1	0.0041	0.2613	-0.1464	0.1051	0.9684	1.6899
2	0.7023	0.8501	-0.2473	1.3540	-0.0103	-0.0984
3	0.8523	0.5499	-0.0060	0.5849	0.6459	-0.7777
4	1.3868	-0.4628	-0.4681	-0.3713	-0.6504	-0.3463
5	-1.0310	-0.2433	0.7639	0.2597	-0.1917	-1.7939
6	2.1667	1.5757	0.7640	0.2399	1.6657	-1.7548
7	1.0722	0.2148	-0.9608	-1.6252	-0.8396	0.5889
8	0.0041	-2.3534	-0.8907	0.4237	-1.8008	-0.4572
9	0.1934	-0.9371	1.1876	0.2634	-0.1932	-0.0153
10	0.0274	-1.2054	-0.3821	-2.2301	-1.4210	-0.9685
11	0.0041	-1.4042	0.6018	-0.1869	-0.9476	-0.5752
12	0.2868	0.2751	0.7226	-1.3031	-0.0103	0.2212
13	-0.5050	-0.2914	2.1729	1.5776	0.8567	1.1061
14	-0.0195	0.4488	1.0192	1.8716	-0.0103	-1.3683

⋮

Decision Tree model

```
mdl=fitctree(X_train, y_train,'CrossVal','On')
```

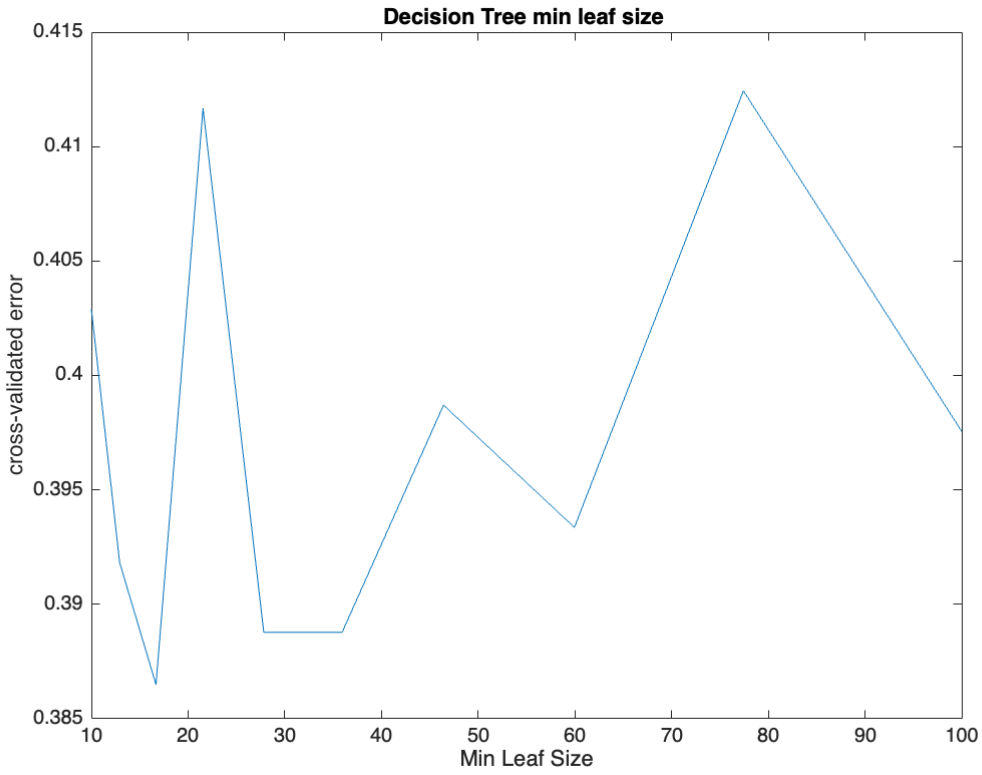
```
mdl =
  ClassificationPartitionedModel
    CrossValidatedModel: 'Tree'
         PredictorNames: {'ph'  'Hardness'  'Solids'  'Chloramines'  'Sulfate'  'Conductivity'  'Organic_c
           ResponseName: 'Potability'
        NumObservations: 2621
                  KFold: 10
              Partition: [1×1 cvpartition]
             ClassNames: [0 1]
         ScoreTransform: 'none'


  Properties, Methods
```

```
accuracy= 1- kfoldLoss(mdl)
```

```
accuracy = 0.5769
```

```
%% code from MATLAB Control Depth or "Leafiness": https://www.mathworks.com/help/stats/
%We know that adjust MinLeafSize can improve model accuracy
```

```
%using crossval defult by 10 to prevent ovetfitting

leafs = logspace(1,2,10);
rng('default')
N = numel(leafs);
err = zeros(N,1);
for n=1:N
    t = fitctree(X_train,y_train,'CrossVal','On',...
        'MinLeafSize',leafs(n));
    err(n) = kfoldLoss(t);
end
plot(leafs,err);
xlabel('Min Leaf Size');
ylabel('cross-validated error');
title('Decision Tree min leaf size ')
```



Fitctree has 3 hyperparameter to control tree depth MaxNumSplits, MinLeafSize, or MinParentSize. We will optimise only MaxNumSplits and MinLeafSize.

Decision tree after tuning parameter

Using two technique for opimise hyperparameters : manual by using for loop and auto OptimizeHyperparameters method from MATLAB

1).hyperparameter optimise by using for loop

```
leafs = [1:2:100]; %MinLeafSize
```

```
np = [1:2:100];   %MaxNumsplit

rng('default')
N = numel(leafs);
NN= numel(np);
err = [];
for n=1:N
    for m=1:NN
        t = fitctree(X_train,y_train,'CrossVal','On',...
            'MaxNumSplits',np(m),'MinLeafSize',leafs(n));
             err(n,m) = kfoldLoss(t);
    end
end

[minValue,I] = min(err(:));
[I_row, I_col] = ind2sub(size(err),I)
```

```
I_row = 2
I_col = 29
```

```
t = fitctree(X_train,y_train,'CrossVal','On', 'MaxNumSplits',I_row,'MinLeafSize',I_col
```

```
t =
  ClassificationPartitionedModel
    CrossValidatedModel: 'Tree'
         PredictorNames: {'ph'  'Hardness'  'Solids'  'Chloramines'  'Sulfate'  'Conductivity'  'Organic_c
           ResponseName: 'Potability'
        NumObservations: 2621
                  KFold: 10
              Partition: [1×1 cvpartition]
             ClassNames: [0 1]
         ScoreTransform: 'none'


  Properties, Methods
```

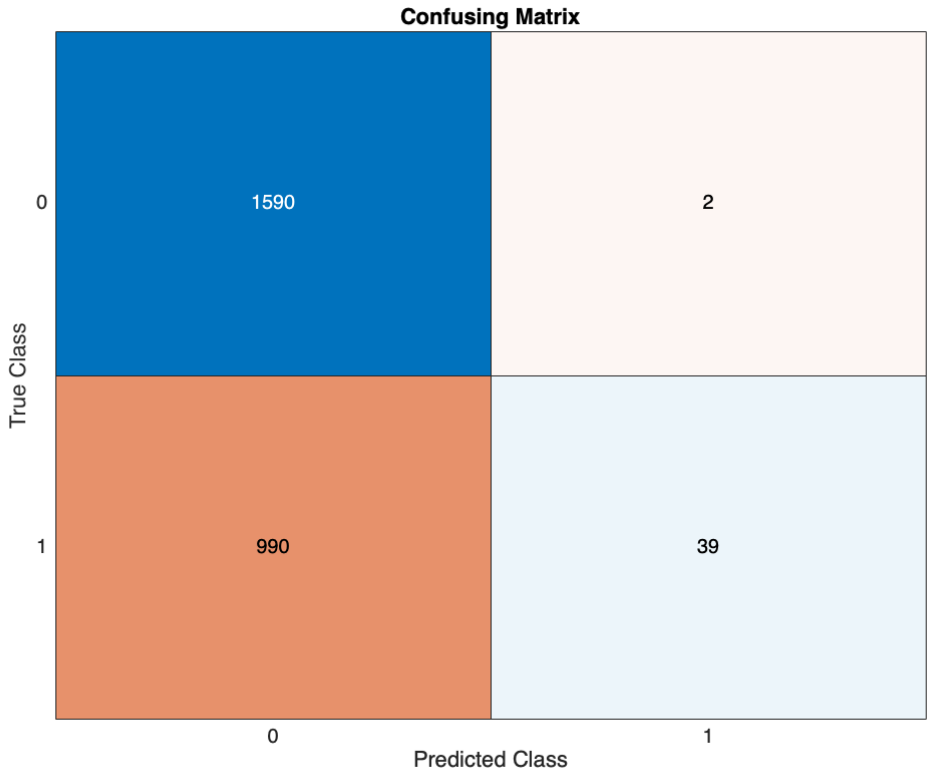
```
modelLosses = kfoldLoss(t,'mode','individual')
```

```
modelLosses = 10×1
    0.3817
    0.3840
    0.3817
    0.4046
    0.3855
    0.3817
    0.3969
    0.3740
    0.3740
    0.3779
```

```
y_predict= predict(t.Trained{1},X_train);
y_train_check= y_train{:,:};
confusion_score(y_train_check,y_predict)
```

```
precision = 0.7837
recall = 0.5183
```

```
f1_score = 0.6240
Accuracy = 0.6215
```



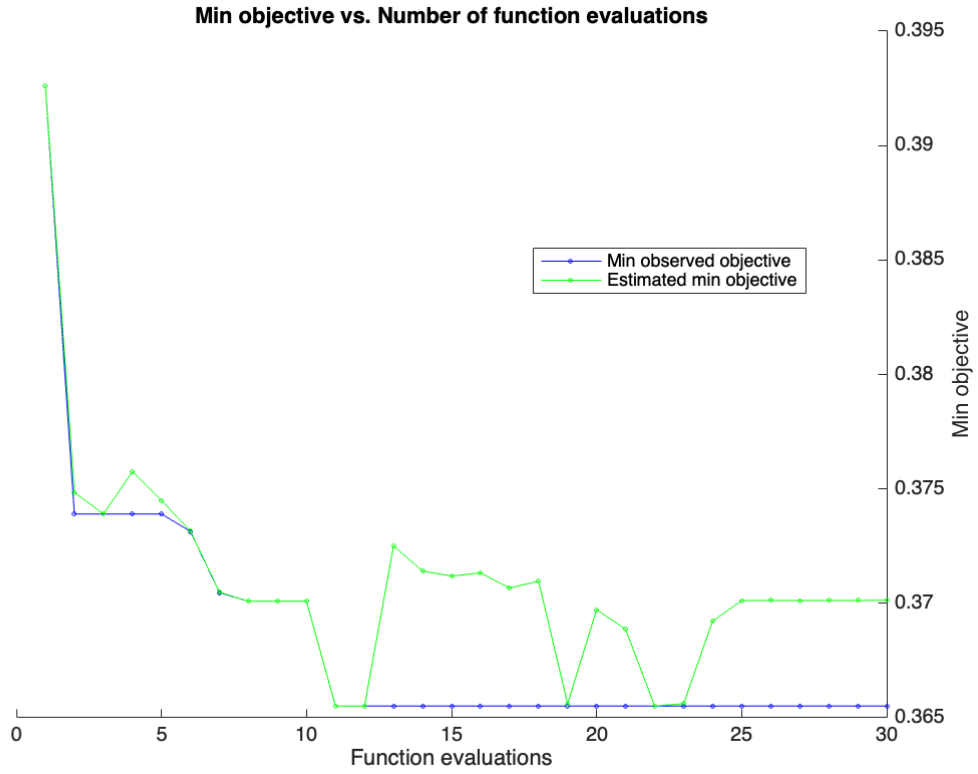
2).Using auto hyperparameter optimise

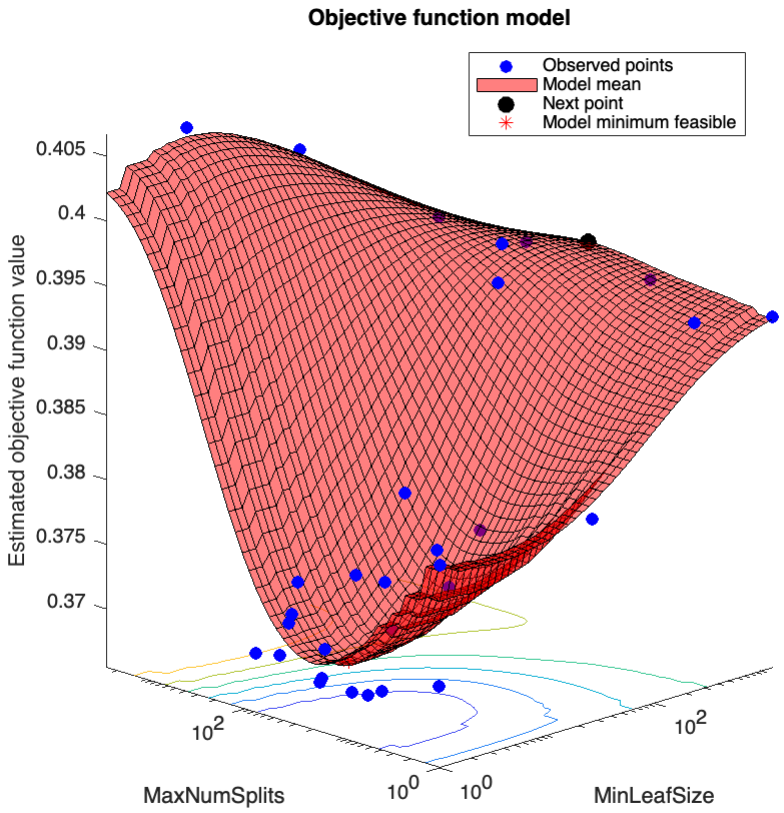
optimize hyperparameters automatically : reference from MATLAB https://www.mathworks.com/help/stats/fitctree.html

```
MDL = fitctree(X_train,y_train,'OptimizeHyperparameters',{'MaxNumSplits','MinLeafSize'
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize	MaxNumSplits
1	Best	0.3926	0.10083	0.3926	0.3926	455	2
2	Best	0.3739	0.15599	0.3739	0.37485	1	33
3	Accept	0.39565	0.098913	0.3739	0.3739	232	90
4	Accept	0.40671	0.12241	0.3739	0.37576	3	1317
5	Accept	0.37543	0.079641	0.3739	0.3745	4	33
6	Best	0.37314	0.075918	0.37314	0.37315	1	35
7	Best	0.37047	0.080732	0.37047	0.37049	1	43
8	Best	0.37009	0.096136	0.37009	0.37009	1	77
9	Accept	0.37505	0.07912	0.37009	0.37009	2	60
10	Accept	0.37467	0.074828	0.37009	0.37009	10	10
11	Best	0.36551	0.085221	0.36551	0.36551	18	24
12	Accept	0.37734	0.088526	0.36551	0.36551	32	17
13	Accept	0.38039	0.074075	0.36551	0.37251	13	38
14	Accept	0.36932	0.065718	0.36551	0.37141	1	17
15	Accept	0.36971	0.069109	0.36551	0.37119	1	16
16	Accept	0.372	0.063523	0.36551	0.37133	1	15
17	Accept	0.38115	0.091081	0.36551	0.37067	1	1
18	Accept	0.3926	0.03869	0.36551	0.37097	1276	2555
19	Accept	0.3678	0.075644	0.36551	0.36561	2	17

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	MinLeafSize	MaxNumSplits
20	Accept	0.37123	0.067575	0.36551	0.36972	6	22
21	Accept	0.36704	0.12075	0.36551	0.36887	3	18
22	Accept	0.36704	0.06394	0.36551	0.36552	4	18
23	Accept	0.39527	0.071506	0.36551	0.36561	67	25
24	Accept	0.37657	0.070255	0.36551	0.36923	14	19
25	Accept	0.37619	0.060382	0.36551	0.37012	3	12
26	Accept	0.38115	0.044251	0.36551	0.37013	27	1
27	Accept	0.40099	0.071128	0.36551	0.37012	64	2609
28	Accept	0.3926	0.040593	0.36551	0.37013	1305	337
29	Accept	0.3926	0.035674	0.36551	0.37013	1302	1
30	Accept	0.3926	0.036778	0.36551	0.37014	1303	18



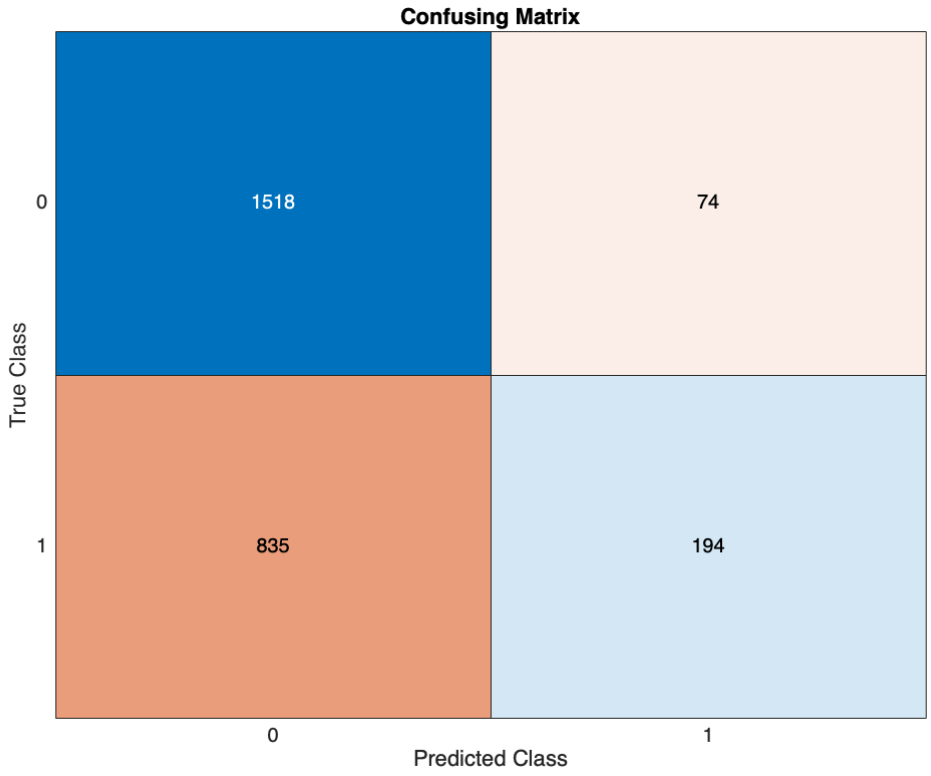


```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 15.8905 seconds
Total objective function evaluation time: 2.2989

Best observed feasible point:
    MinLeafSize    MaxNumSplits
    ___________    ____________

        18             24

Observed objective function value = 0.36551
Estimated objective function value = 0.37723
Function evaluation time = 0.085221

Best estimated feasible point (according to models):
    MinLeafSize    MaxNumSplits
    ___________    ____________

         2             17

Estimated objective function value = 0.37014
Estimated function evaluation time = 0.075801
MDL =
  ClassificationTree
                      PredictorNames: {'ph'  'Hardness'  'Solids'  'Chloramines'  'Sulfate'  'Conductivit
                        ResponseName: 'Potability'
               CategoricalPredictors: []
                          ClassNames: [0 1]
                      ScoreTransform: 'none'
```

```
                    NumObservations: 2621
    HyperparameterOptimizationResults: [1×1 BayesianOptimization]

  Properties, Methods
```

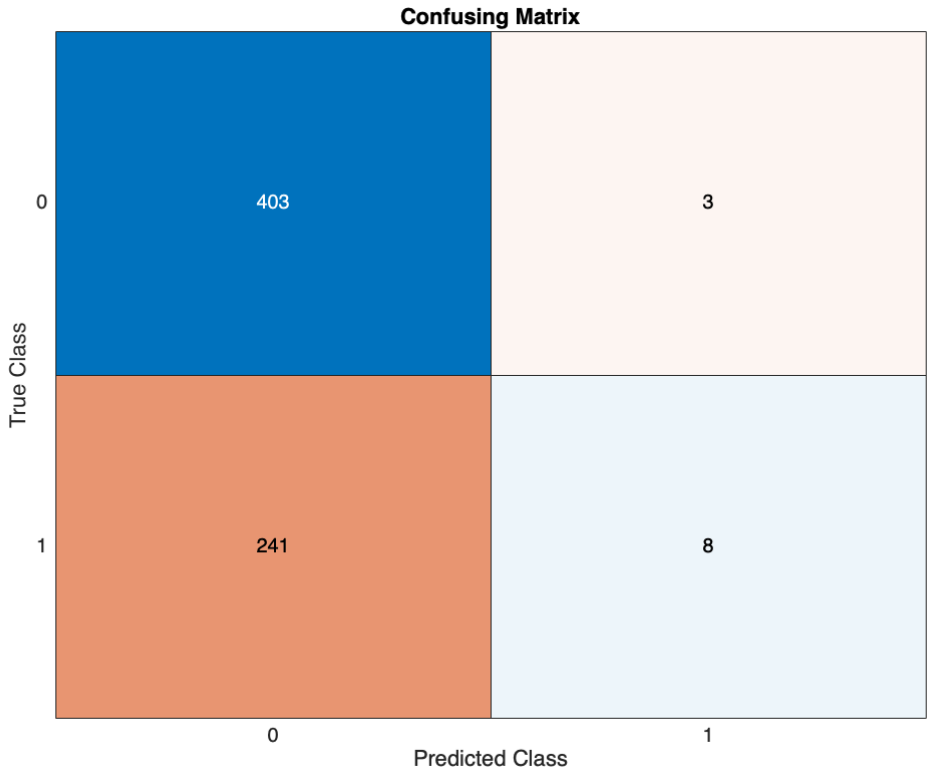
```
y_predict= predict(MDL,X_train);
y_train_check= y_train{:,:};
confusion_score(y_train_check,y_predict)
```

```
precision = 0.6845
recall = 0.5710
f1_score = 0.6226
Accuracy = 0.6532
```

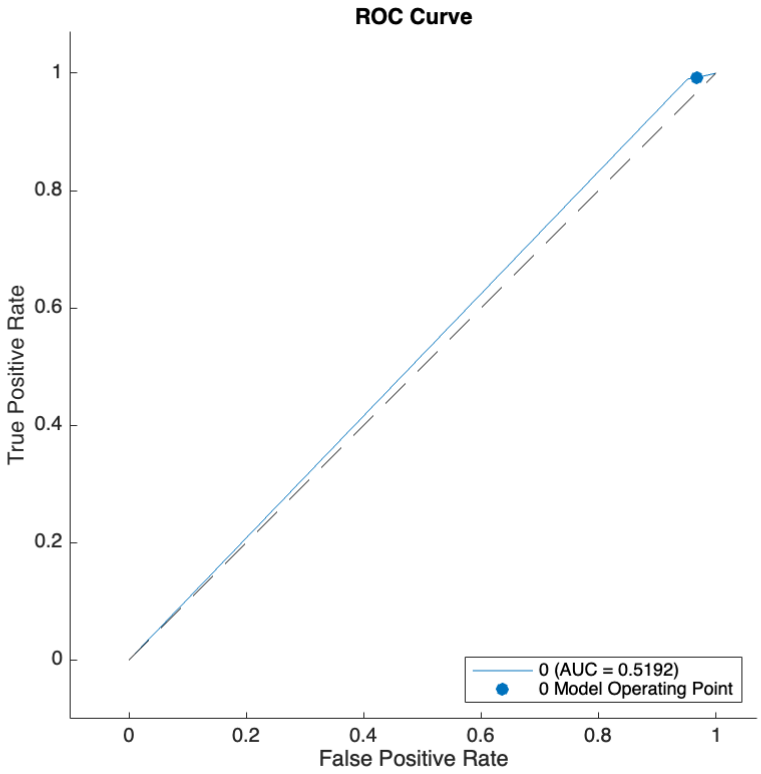



```
%performance of Decisioin tree on test dataset
y_predict = predict(t.Trained{1},X_test);
confusion_score(y_test,y_predict)
```

```
precision = 0.6765
recall = 0.5124
f1_score = 0.5831
Accuracy = 0.6275
```




```
%% ROC curve Decision tree
[y_predict,scores] = predict(t.Trained{1},X_test);
size(scores);
rocObj_DT = rocmetrics(y_test,scores,t.Trained{1}.ClassNames);
rocObj_DT.AUC;
figure
plot(rocObj_DT,ClassNames=t.Trained{1}.ClassNames(1))
```



Random Forest model

Can use both TreeBagger or fitcensemble to produce Random forest model

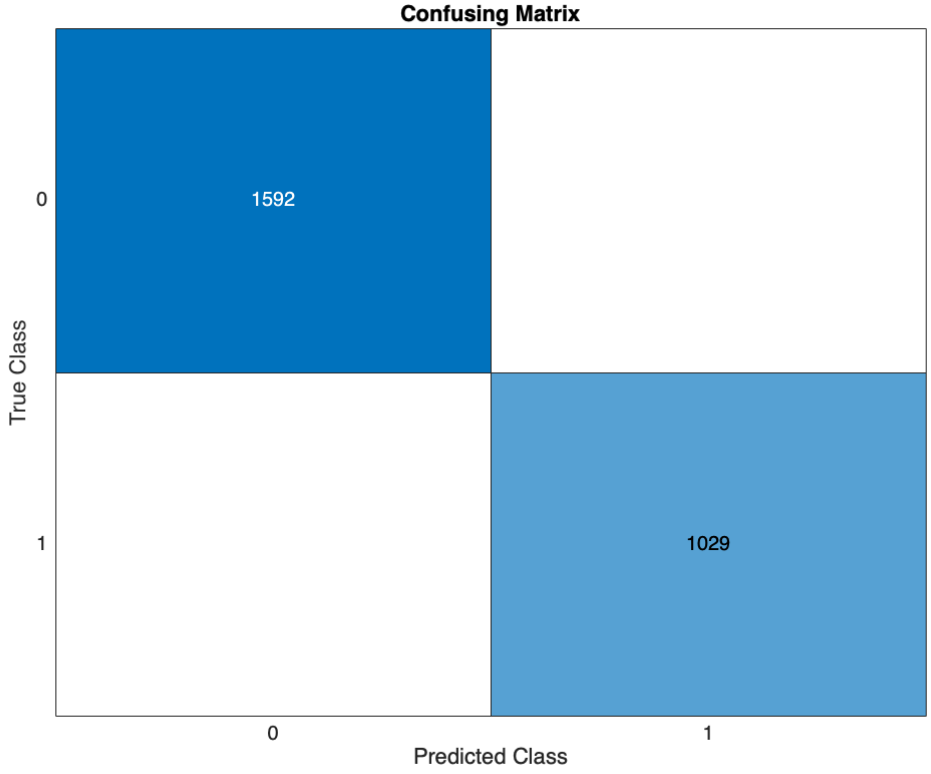
```
bag = TreeBagger(100,X_train,y_train,'method','classification')
```

```
bag =
  TreeBagger
Ensemble with 100 bagged decision trees:
                    Training X:              [2621x9]
                    Training Y:              [2621x1]
                        Method:       classification
                NumPredictors:                    9
       NumPredictorsToSample:                     3
                   MinLeafSize:                    1
                InBagFraction:                     1
       SampleWithReplacement:                      1
        ComputeOOBPrediction:                      0
 ComputeOOBPredictorImportance:                    0
                     Proximity:                   []
                    ClassNames:             '0'            '1'

  Properties, Methods
```

```
y_predict= predict(bag,X_train);
y_predict = str2double(y_predict);
y_train_check= y_train{:,:};
confusion_score(y_train_check,y_predict)
```

```
precision = 1
```

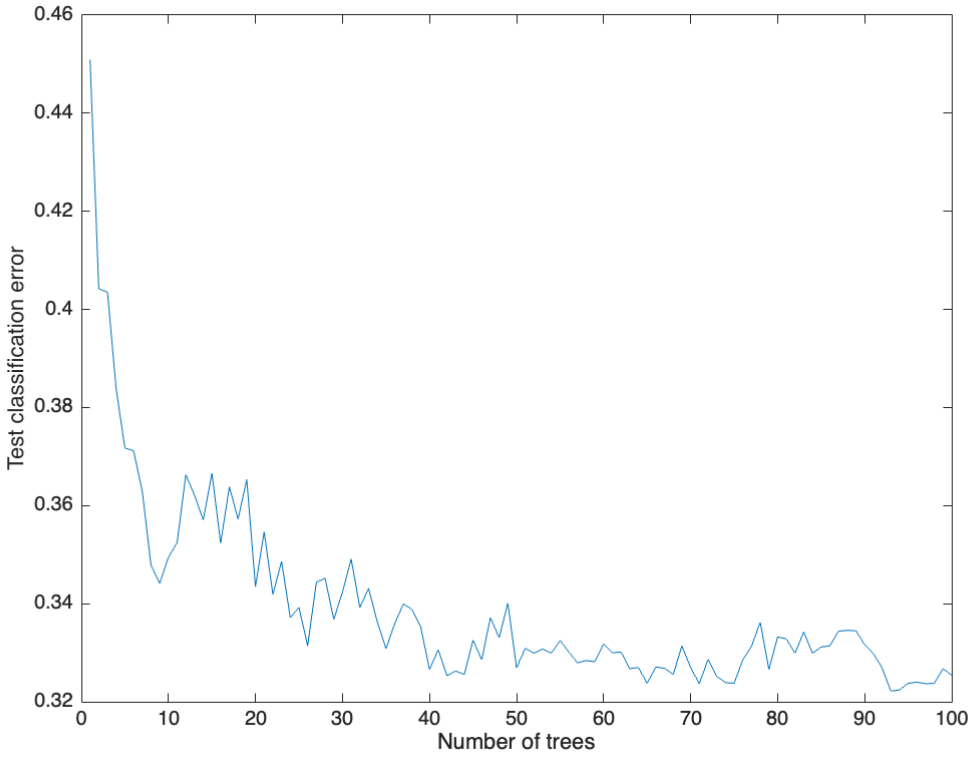
```
recall = 1
f1_score = 1
Accuracy = 1
```



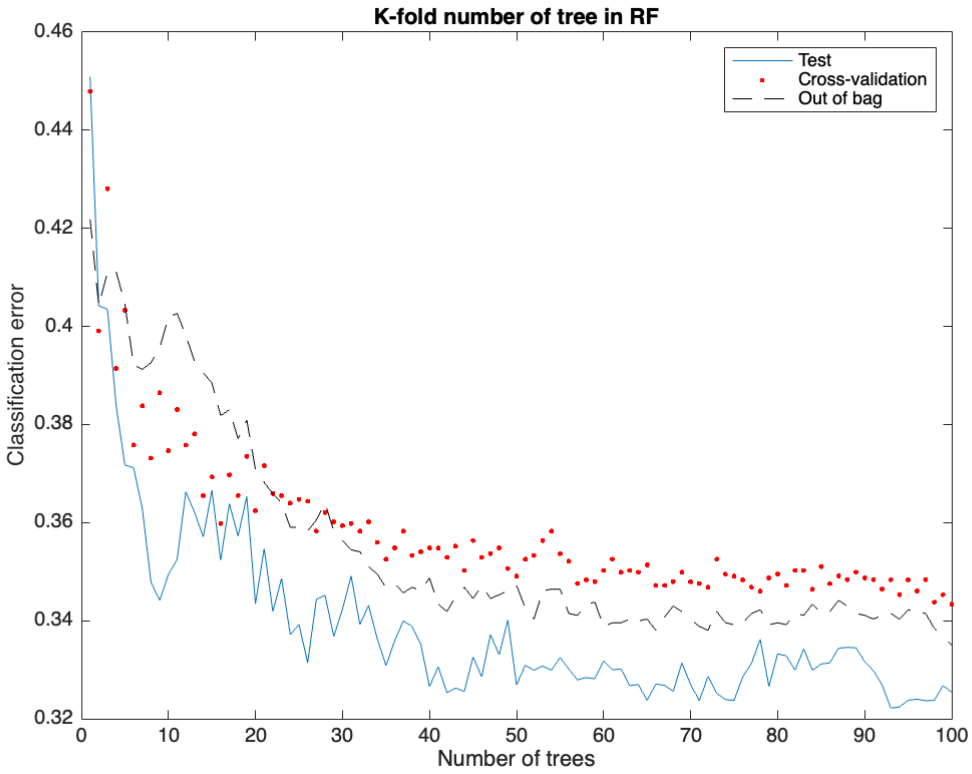

```
t = templateTree('Reproducible',true);  % For reproducibility of random predictor sele
bag = fitcensemble(X_train,y_train,'Method','Bag','NumLearningCycles',100,'Learners',t
```

```
bag =
  ClassificationBaggedEnsemble
             PredictorNames: {'ph'  'Hardness'  'Solids'  'Chloramines'  'Sulfate'  'Conductivity'  'Organic
               ResponseName: 'Potability'
      CategoricalPredictors: []
                 ClassNames: [0 1]
             ScoreTransform: 'none'
            NumObservations: 2621
                 NumTrained: 100
                     Method: 'Bag'
               LearnerNames: {'Tree'}
     ReasonForTermination: 'Terminated normally after completing the requested number of training cycles.
                  FitInfo: []
       FitInfoDescription: 'None'
                 FResample: 1
                   Replace: 1
          UseObsForLearner: [2621×100 logical]


  Properties, Methods
```

```
figure
plot(loss(bag,X_test,y_test,'mode','cumulative'))
xlabel('Number of trees')
ylabel('Test classification error')
```




```
%% Graph number of tree & Cross-Val reference from from test ensemble quality MATLAB :
cv_mol = fitcensemble(X_train,y_train,'Method','Bag','NumLearningCycles',100,'Kfold',5
figure
plot(loss(bag,X_test,y_test,'mode','cumulative'))
hold on
plot(kfoldLoss(cv_mol,'mode','cumulative'),'r.')
plot(oobLoss(bag,'mode','cumulative'),'k--')
hold off
xlabel('Number of trees')
ylabel('Classification error')
title('K-fold number of tree in RF')
legend('Test','Cross-validation','Out of bag','Location','NE')
```



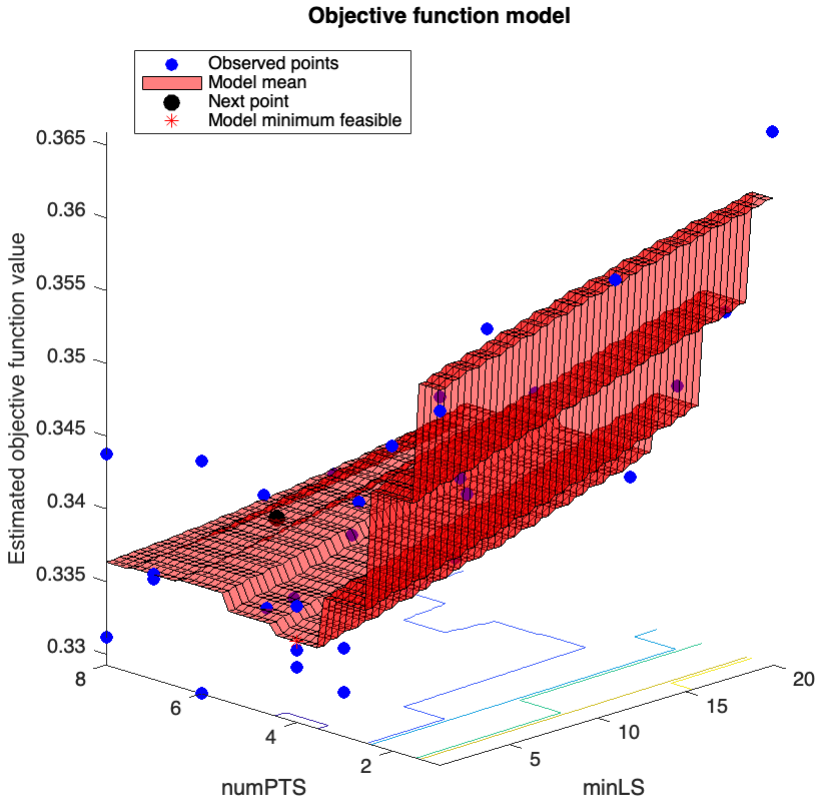
From the two graphs above we can see that increasing the number of trees can improve our model accuracy. Therefore, we shall adjust the number of trees in forest and the minimum leaf size as a hyperparameter for random forest

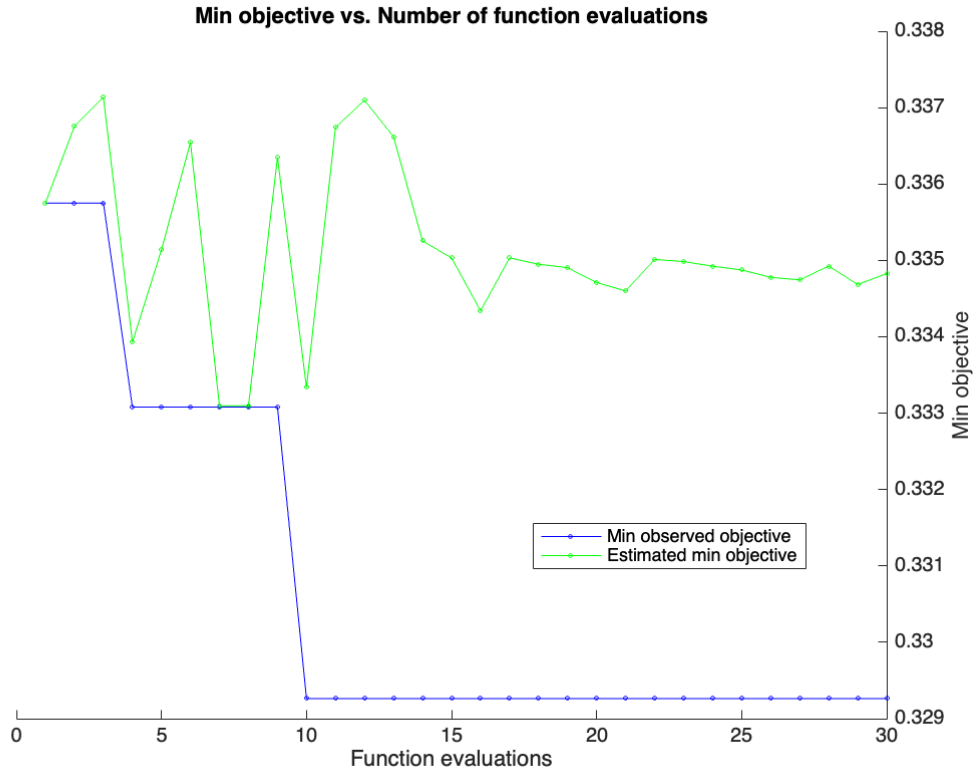
Random forest hyperparameter tuning

1.) Using Baysian optimization with TreeBagger

inspiration from the example of Tune Random Forest Using Quantile Error and Bayesian Optimization in MATLAB

```
%% Optimise Random Forest Hyperparameter using Baysian optimization
maxMinLS = 20;
minLS = optimizableVariable('minLS',[1,maxMinLS],'Type','integer');
numPTS = optimizableVariable('numPTS',[1,size(X_train,2)-1],'Type','integer');
hyperparametersRF = [minLS; numPTS];

results = bayesopt(@(params)oobErrRF(params,X_train,y_train),hyperparametersRF,...
    'AcquisitionFunctionName','expected-improvement-plus','Verbose',0);
```






```
bestOOBErr = results.MinObjective;
```

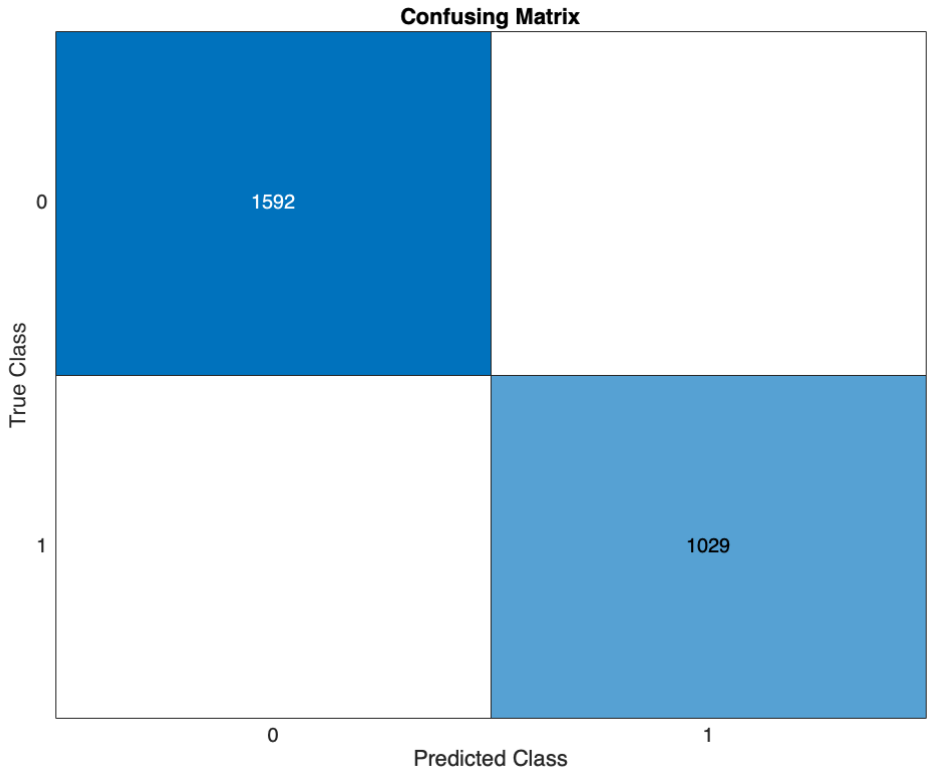
```
bestHyperparameters = results.XAtMinObjective
```

```
bestHyperparameters = 1×2 table
```

	minLS	numPTS
1	1	6

```
Mdl = TreeBagger(300,X_train,y_train,'Method','classification',...
    'MinLeafSize',bestHyperparameters.minLS,...
    'NumPredictorstoSample',bestHyperparameters.numPTS);

y_predict= predict(Mdl,X_train);
y_predict = str2double(y_predict);
y_train_check= y_train{:,:};
confusion_score(y_train_check,y_predict)
```

```
precision = 1
recall = 1
f1_score = 1
Accuracy = 1
```



Note: the disadvantage of using the bayesian optimise function is we cannot apply a cross-validation function to check whether the model is overfitted or not. As we have seen from the confusion matrix it does perform well to predict the training set but poor performance when trying to predict the test set. So, we decide to use fitcensemble optimise hyperparameters instead of using TreeBagger-Baysian

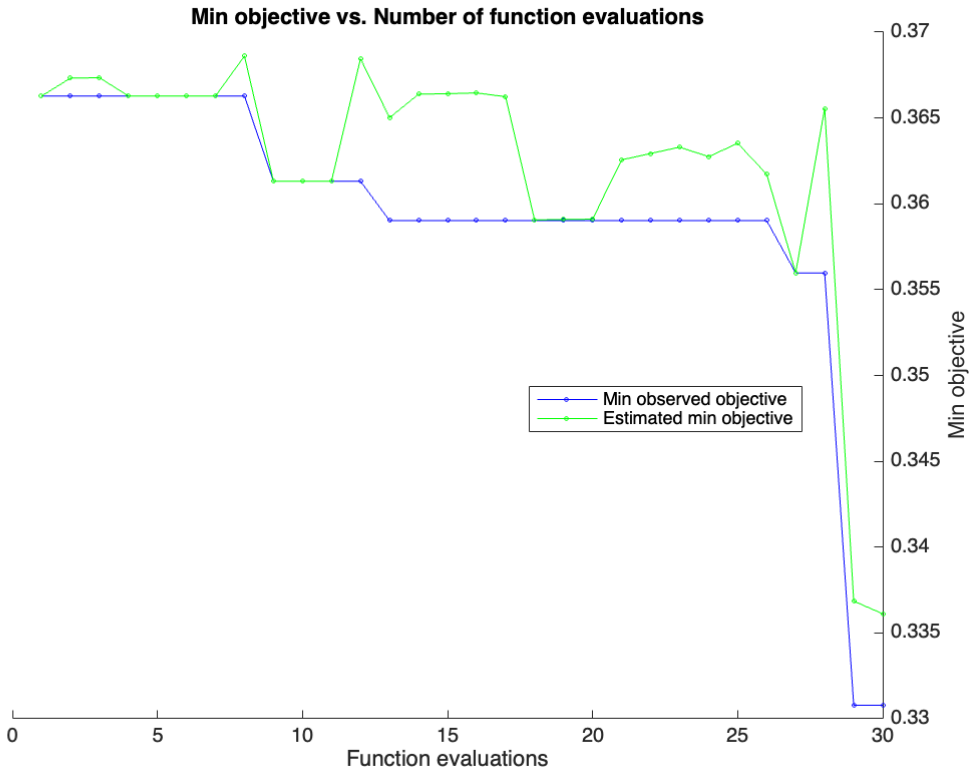
2.) using optimiseHyperparameter with fitcensemble

```
%%optimiseHyperparameter
```

```
t = templateTree('Reproducible',true);
Mdl = fitcensemble(X_train,y_train,'OptimizeHyperparameters','auto','Learners',t, ...
    'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName','expected-imp
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles
1	Best	0.36627	1.3296	0.36627	0.36627	AdaBoostM1	21
2	Accept	0.3926	0.56852	0.36627	0.36732	Bag	20
3	Accept	0.36894	7.2973	0.36627	0.36735	AdaBoostM1	186
4	Accept	0.38993	7.638	0.36627	0.36628	GentleBoost	359
5	Accept	0.3926	1.774	0.36627	0.36627	AdaBoostM1	423
6	Accept	0.3926	0.54879	0.36627	0.36628	AdaBoostM1	31
7	Accept	0.36818	3.2687	0.36627	0.36628	AdaBoostM1	80
8	Accept	0.37085	0.91017	0.36627	0.36861	AdaBoostM1	18
9	Best	0.36131	2.0379	0.36131	0.36132	AdaBoostM1	45
10	Accept	0.3781	3.1008	0.36131	0.36132	AdaBoostM1	75
11	Accept	0.36246	19.719	0.36131	0.36132	AdaBoostM1	459
12	Accept	0.37619	1.6035	0.36131	0.36846	AdaBoostM1	35
13	Best	0.35902	2.1063	0.35902	0.36501	AdaBoostM1	49
14	Accept	0.36971	15.83	0.35902	0.36638	AdaBoostM1	382
15	Accept	0.37696	4.9414	0.35902	0.36642	RUSBoost	93
16	Accept	0.3926	2.3322	0.35902	0.36646	RUSBoost	92
17	Accept	0.36093	1.3934	0.35902	0.36624	LogitBoost	46
18	Accept	0.36169	1.538	0.35902	0.35906	LogitBoost	49
19	Accept	0.36208	0.682	0.35902	0.35909	AdaBoostM1	14
20	Accept	0.36322	0.40763	0.35902	0.35911	LogitBoost	11

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Method	NumLearningCycles
21	Accept	0.37085	4.1981	0.35902	0.36256	AdaBoostM1	99
22	Accept	0.3636	10.808	0.35902	0.36293	LogitBoost	404
23	Accept	0.37238	9.2752	0.35902	0.3633	LogitBoost	345
24	Accept	0.37734	0.5522	0.35902	0.36274	LogitBoost	17
25	Accept	0.3781	10.615	0.35902	0.36353	LogitBoost	394
26	Accept	0.36131	0.76471	0.35902	0.36172	AdaBoostM1	16
27	Best	0.35597	1.0727	0.35597	0.35598	AdaBoostM1	22
28	Accept	0.37696	1.3428	0.35597	0.36556	AdaBoostM1	30
29	Best	0.33079	48.937	0.33079	0.33685	Bag	491
30	Accept	0.33575	49.188	0.33079	0.33612	Bag	495



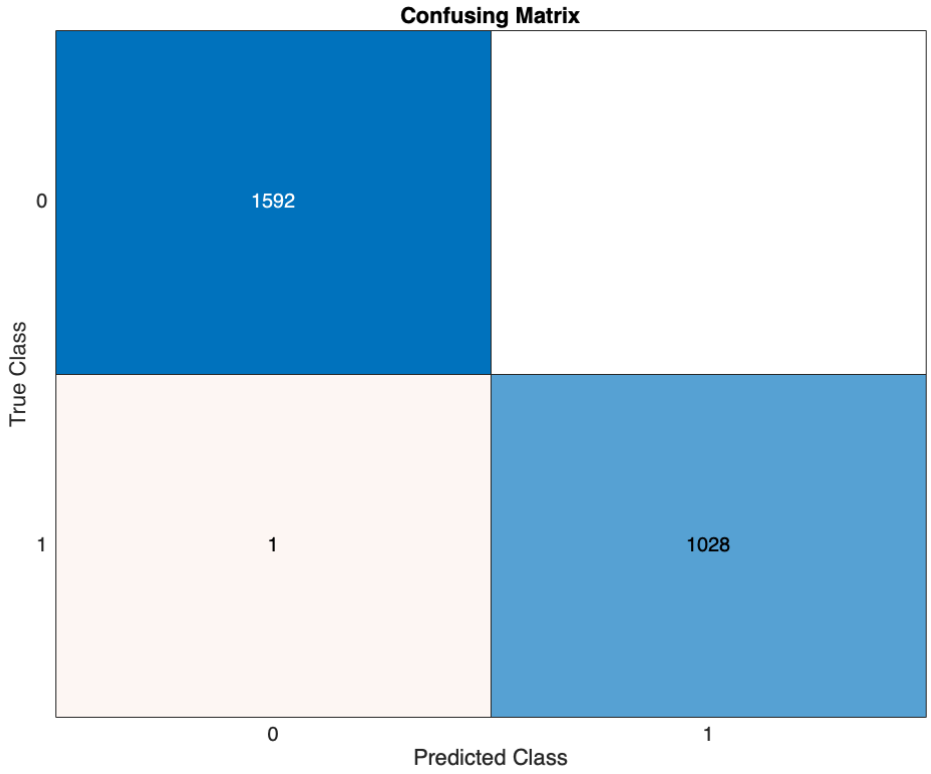

```
_____________________________________________________________
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 226.8171 seconds
Total objective function evaluation time: 215.7815

Best observed feasible point:
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag             491               NaN            1

Observed objective function value = 0.33079
Estimated objective function value = 0.33612
Function evaluation time = 48.9367

Best estimated feasible point (according to models):
    Method    NumLearningCycles    LearnRate    MinLeafSize
    ______    _________________    _________    ___________

     Bag             495               NaN            1

Estimated objective function value = 0.33612
Estimated function evaluation time = 49.0725
Mdl =
  ClassificationBaggedEnsemble
                      PredictorNames: {'ph'  'Hardness'  'Solids'  'Chloramines'  'Sulfate'  'Conductivit
                        ResponseName: 'Potability'
               CategoricalPredictors: []
                          ClassNames: [0 1]
                      ScoreTransform: 'none'
                     NumObservations: 2621
    HyperparameterOptimizationResults: [1×1 BayesianOptimization]
```

```
                NumTrained: 495
                    Method: 'Bag'
               LearnerNames: {'Tree'}
ReasonForTermination: 'Terminated normally after completing the requested number of trai
                   FitInfo: []
        FitInfoDescription: 'None'
                  FResample: 1
                    Replace: 1
      UseObsForLearner: [2621×495 logical]
```

Properties, Methods

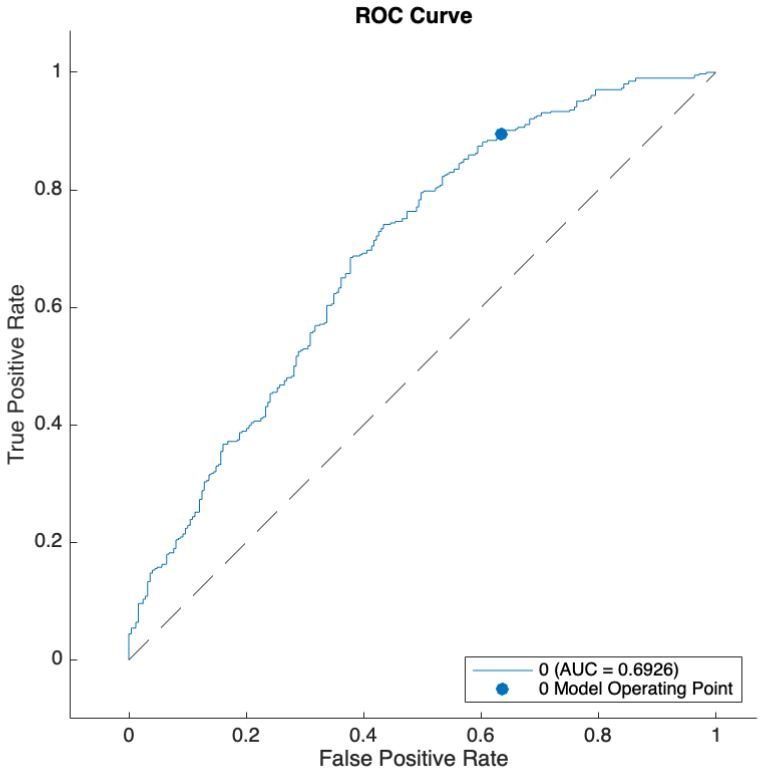
```
y_predict= predict(Mdl,X_train);
y_train_check= y_train{:,:};
confusion_score(y_train_check,y_predict)
```

```
precision = 0.9997
recall = 0.9995
f1_score = 0.9996
Accuracy = 0.9996
```



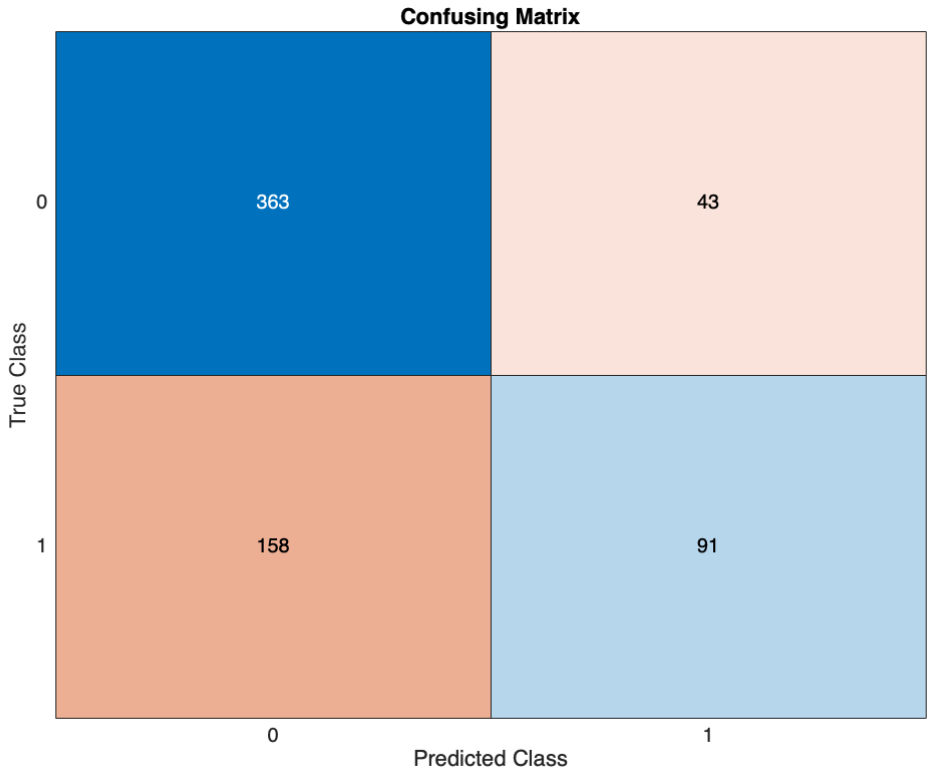

```
%% ROC curve Randomforest
[y_predict,scores] = predict(Mdl,X_test);
size(scores);

rocObj_RF = rocmetrics(y_test,scores,mdl.ClassNames);
rocObj_RF.AUC;
figure
```

```
plot(rocObj_RF,ClassNames=mdl.ClassNames(1))
```

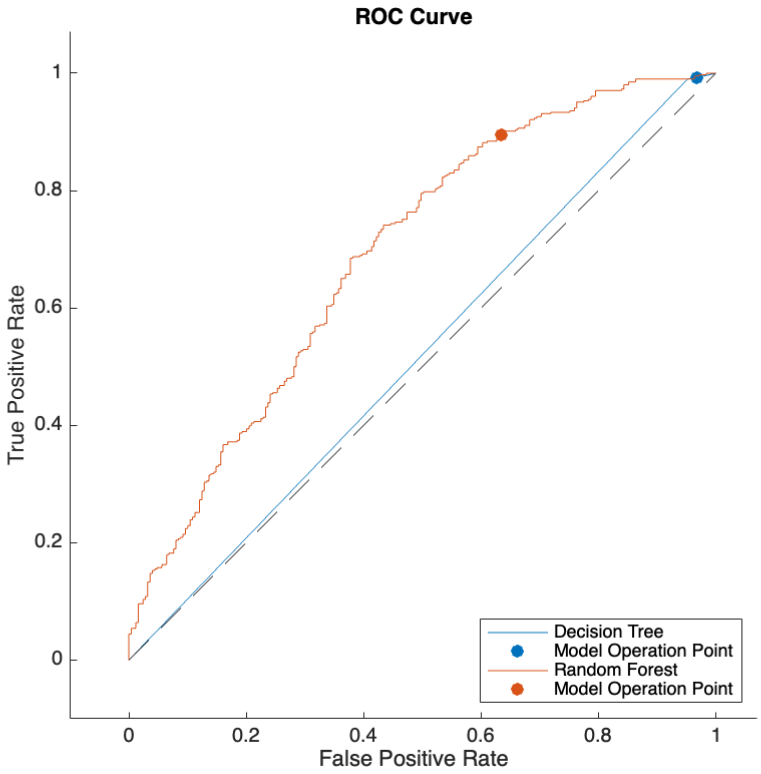



```
%performance of Random forest on test dataset
y_predict = predict(Mdl,X_test);
confusion_score(y_test,y_predict)
```

```
precision = 0.6879
recall = 0.6298
f1_score = 0.6576
Accuracy = 0.6931
```




```
%% ROC curve : DT vs RF
figure
plot(rocObj_DT,ClassNames=mdl.ClassNames(1))
hold on
plot(rocObj_RF,ClassNames=mdl.ClassNames(1))
legend('Decision Tree','Model Operation Point','Random Forest','Model Operation Point'
```



The comparison between the two models in the ROC curve shows that random forest performs better than decision tree to predict the potability of water in this test dataset

```
function confusion_score(x,y)
cm= confusionmat(x, y);
cmt = cm';
diagonal = diag(cmt);
sum_of_rows = sum(cmt, 2);

precision = diagonal ./ sum_of_rows;
precision = mean(precision)

sum_of_columns = sum(cmt, 1);
recall = diagonal ./ sum_of_columns';
recall = mean(recall)

f1_score = 2*((precision*recall)/(precision+recall))
Accuracy= (diagonal(1,1)+diagonal(2,1))/(sum_of_rows(1,1)+sum_of_rows(2,1))

figure
confusionchart(x,y)
title('Confusing Matrix')
end
%reference from: Precision Recall F1 Score from Confusion Matrix in MATLAB https://www
```

```
function oobErr = oobErrRF(params,X_train,y_train)
randomForest = TreeBagger(300,X_train,y_train,'method','classification',...
    'OOBPrediction','on','MinLeafSize',params.minLS,...
    'NumPredictorstoSample',params.numPTS);
oobErr = oobError(randomForest, 'Mode','ensemble');
end

%reference from: https://www.mathworks.com/help/stats/tune-random-forest-using-quantil
```