

fitcensemble

Fit ensemble of learners for classification

Syntax

```
Mdl = fitcensemble(Tbl,ResponseVarName)
Mdl = fitcensemble(Tbl,formula)
Mdl = fitcensemble(Tbl,Y)

Mdl = fitcensemble(X,Y)

Mdl = fitcensemble(__,Name,Value)
[Mdl,AggregateOptimizationResults] = fitcensemble(__)
```

Description

Mdl = **fitcensemble**(**Tbl**,**ResponseVarName**) returns the trained classification ensemble model object (**Mdl**) [example](#) that contains the results of boosting 100 classification trees and the predictor and response data in the table **Tbl**. **ResponseVarName** is the name of the response variable in **Tbl**. By default, **fitcensemble** uses LogitBoost for binary classification and AdaBoostM2 for multiclass classification.

Mdl = **fitcensemble**(**Tbl**,**formula**) applies **formula** to fit the model to the predictor and response data in the table **Tbl**. **formula** is an explanatory model of the response and a subset of predictor variables in **Tbl** used to fit **Mdl**. For example, 'Y~X1+X2+X3' fits the response variable **Tbl.Y** as a function of the predictor variables **Tbl.X1**, **Tbl.X2**, and **Tbl.X3**. [example](#)

Mdl = **fitcensemble**(**Tbl**,**Y**) treats all variables in the table **Tbl** as predictor variables. **Y** is the array of class labels that is not in **Tbl**. [example](#)

Mdl = **fitcensemble**(**X**,**Y**) uses the predictor data in the matrix **X** and the array of class labels in **Y**. [example](#)

Mdl = **fitcensemble**(__ ,**Name**,**Value**) uses additional options specified by one or more **Name**,**Value** pair arguments and any of the input arguments in the previous syntaxes. For example, you can specify the number of learning cycles, the ensemble aggregation method, or to implement 10-fold cross-validation. [example](#)

[Mdl,AggregateOptimizationResults] = **fitcensemble**(__) also returns **AggregateOptimizationResults**, which contains hyperparameter optimization results when you specify the [OptimizeHyperparameters](#) and [HyperparameterOptimizationOptions](#) name-value arguments. You must also specify the **ConstraintType** and **ConstraintBounds** options of **HyperparameterOptimizationOptions**. You can use this syntax to optimize on compact model size instead of cross-validation loss, and to perform a set of multiple optimization problems that have the same options but different constraint bounds.

Examples

[collapse all](#)

Train Classification Ensemble

Create a predictive classification ensemble using all available predictor variables in the data. Then, train another ensemble using fewer predictors. Compare the in-sample predictive accuracies of the ensembles.

Load the census1994 data set.

Open in MATLAB
Online

 Copy Command

```
load census1994
```

 Get ▾

Train an ensemble of classification models using the entire data set and default options.

```
Mdl1 = fitcensemble(adultdata,'salary')
```

 Get ▾

```
Mdl1 =
    ClassificationEnsemble
        PredictorNames: {'age' 'workClass' 'fnlwtg' 'education' 'education_num' 'marit
        ResponseName: 'salary'
    CategoricalPredictors: [2 4 6 7 8 9 10 14]
        ClassNames: [<=50K >50K]
        ScoreTransform: 'none'
    NumObservations: 32561
        NumTrained: 100
        Method: 'LogitBoost'
        LearnerNames: {'Tree'}
    ReasonForTermination: 'Terminated normally after completing the requested number of train
        FitInfo: [100x1 double]
    FitInfoDescription: {2x1 cell}
```


Properties, Methods

Mdl is a ClassificationEnsemble model. Some notable characteristics of Mdl are:

- Because two classes are represented in the data, LogitBoost is the ensemble aggregation algorithm.
- Because the ensemble aggregation method is a boosting algorithm, classification trees that allow a maximum of 10 splits compose the ensemble.
- One hundred trees compose the ensemble.

Use the classification ensemble to predict the labels of a random set of five observations from the data. Compare the predicted labels with their true values.

```
rng(1) % For reproducibility
[pX,pIdx] = datasample(adultdata,5);
label = predict(Mdl1,pX);
table(label,adultdata.salary(pIdx),'VariableNames',{'Predicted','Truth'})
```

 Get ▾

```
ans=5x2 table
```

Predicted	Truth
<=50K	<=50K
<=50K	<=50K
<=50K	<=50K
<=50K	<=50K
<=50K	<=50K

Train a new ensemble using age and education only.

```
Mdl2 = fitcensemble(adultdata,'salary ~ age + education');
```

 Get ▾

Compare the resubstitution losses between Mdl1 and Mdl2.

```
rsLoss1 = resubLoss(Mdl1)
```

 Get ▾

```
rsLoss1 =
0.1058
```

```
rsLoss2 = resubLoss(Mdl12)
```

[Get](#)

```
rsLoss2 =
0.2037
```

The in-sample misclassification rate for the ensemble that uses all predictors is lower.

Speed Up Training by Binning Numeric Predictor Values

Train an ensemble of boosted classification trees by using `fitcensemble`. Reduce training time by specifying the `'NumBins'` name-value pair argument to bin numeric predictors. This argument is valid only when `fitcensemble` uses a tree learner. After training, you can reproduce binned predictor data by using the `BinEdges` property of the trained model and the `discretize` function.

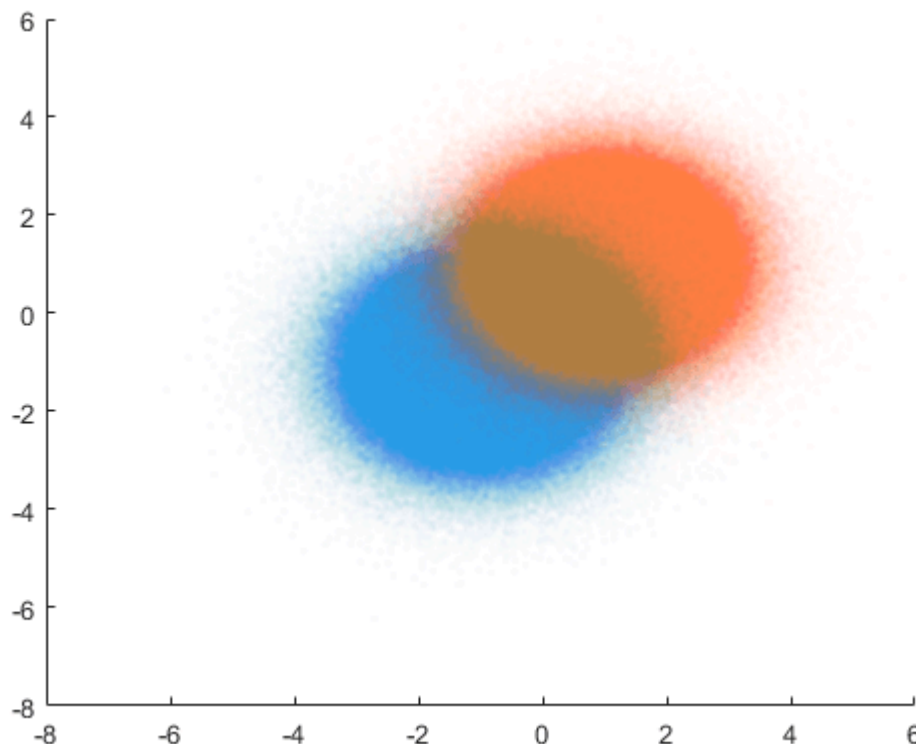
Generate a sample data set.

```
rng('default') % For reproducibility
N = 1e6;
X = [mvnrnd([-1 -1],eye(2),N); mvnrnd([1 1],eye(2),N)];
y = [zeros(N,1); ones(N,1)];
```

[Get](#)

Visualize the data set.

```
figure
scatter(X(1:N,1),X(1:N,2), 'Marker', '.', 'MarkerEdgeAlpha',0.01)
hold on
scatter(X(N+1:2*N,1),X(N+1:2*N,2), 'Marker', '.', 'MarkerEdgeAlpha',0.01)
```

[Get](#)


Train an ensemble of boosted classification trees using adaptive logistic regression (LogitBoost, the default for binary classification). Time the function for comparison purposes.

```
tic
Md11 = fitcensemble(X,y);
toc
```

 Get ▾

Elapsed time is 478.988422 seconds.

Speed up training by using the 'NumBins' name-value pair argument. If you specify the 'NumBins' value as a positive integer scalar, then the software bins every numeric predictor into a specified number of equiprobable bins, and then grows trees on the bin indices instead of the original data. The software does not bin categorical predictors.

```
tic
Md12 = fitcensemble(X,y,'NumBins',50);
toc
```

 Get ▾

Elapsed time is 165.598434 seconds.

The process is about three times faster when you use binned data instead of the original data. Note that the elapsed time can vary depending on your operating system.

Compare the classification errors by resubstitution.

```
rsLoss1 = resubLoss(Md11)
```

 Get ▾

```
rsLoss1 = 0.0788
```

```
rsLoss2 = resubLoss(Md12)
```

 Get ▾

```
rsLoss2 = 0.0788
```

In this example, binning predictor values reduces training time without loss of accuracy. In general, when you have a large data set like the one in this example, using the binning option speeds up training but causes a potential decrease in accuracy. If you want to reduce training time further, specify a smaller number of bins.

Reproduce binned predictor data by using the BinEdges property of the trained model and the [discretize](#) function.

```
X = Md12.X; % Predictor data
Xbinned = zeros(size(X));
edges = Md12.BinEdges;
% Find indices of binned predictors.
idxNumeric = find(~cellfun(@isempty,edges));
if iscolumn(idxNumeric)
    idxNumeric = idxNumeric';
end
for j = idxNumeric
    x = X(:,j);
    % Convert x to array if x is a table.
    if istable(x)
        x = table2array(x);
    end
    % Group x into bins by using the discretize function.
    xbinned = discretize(x,[-inf; edges{j}; inf]);
    Xbinned(:,j) = xbinned;
end
```

 Get ▾

Xbinned contains the bin indices, ranging from 1 to the number of bins, for numeric predictors. Xbinned values are 0 for categorical predictors. If X contains NaNs, then the corresponding Xbinned values are NaNs.

Estimate Generalization Error of Boosting Ensemble

Estimate the generalization error of ensemble of boosted classification trees.

Load the `ionosphere` data set.

[Open in MATLAB Online](#)
[Copy Command](#)

```
load ionosphere
```

[Get](#)

Cross-validate an ensemble of classification trees using AdaBoostM1 and 10-fold cross-validation. Specify that each tree should be split a maximum of five times using a decision tree template.

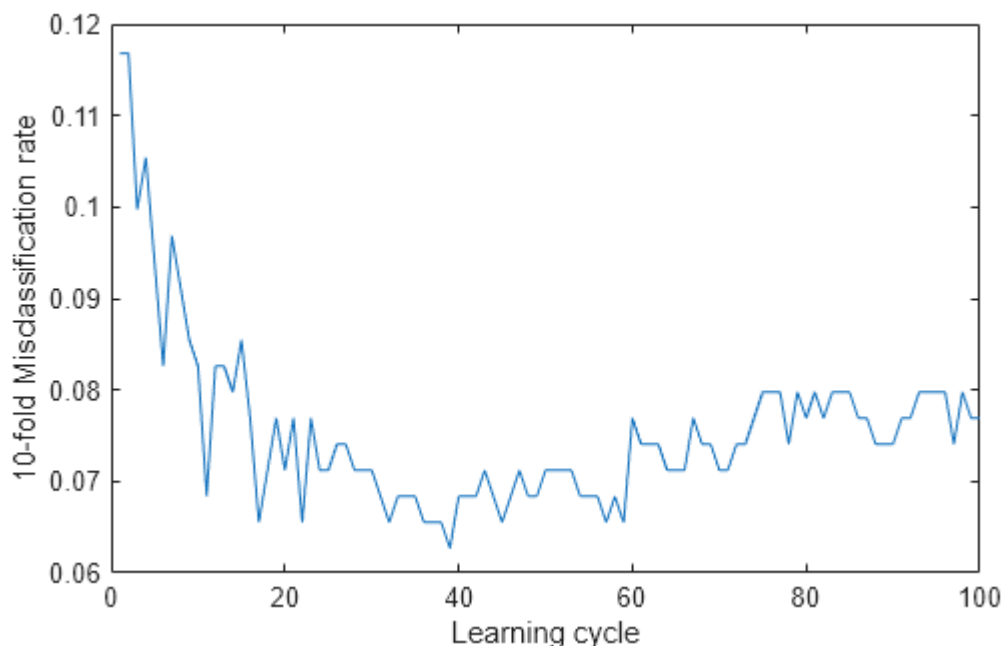
```
rng(5); % For reproducibility
t = templateTree('MaxNumSplits',5);
Mdl = fitcensemble(X,Y,'Method','AdaBoostM1','Learners',t,'CrossVal','on');
```

[Get](#)

Mdl is a `ClassificationPartitionedEnsemble` model.

Plot the cumulative, 10-fold cross-validated, misclassification rate. Display the estimated generalization error of the ensemble.

```
kflc = kfoldLoss(Mdl,'Mode','cumulative');
figure;
plot(kflc);
ylabel('10-fold Misclassification rate');
xlabel('Learning cycle');
```

[Get](#)


```
estGenError = kflc(end)
```

[Get](#)

```
estGenError =
0.0769
```

`kfoldLoss` returns the generalization error by default. However, plotting the cumulative loss allows you to monitor how the loss changes as weak learners accumulate in the ensemble.

The ensemble achieves a misclassification rate of around 0.06 after accumulating about 50 weak learners. Then, the misclassification rate increase slightly as more weak learners enter the ensemble.

If you are satisfied with the generalization error of the ensemble, then, to create a predictive model, train the ensemble again using all of the settings except cross-validation. However, it is good practice to tune

hyperparameters, such as the maximum number of decision splits per tree and the number of learning cycles.

Optimize Classification Ensemble

Optimize hyperparameters automatically using `fitcensemble`.

Load the `ionosphere` data set.

Open in MATLAB
Online

Copy Command

load `ionosphere`

Get

You can find hyperparameters that minimize five-fold cross-validation loss by using automatic hyperparameter optimization.

```
Mdl = fitcensemble(X,Y,'OptimizeHyperparameters','auto')
```

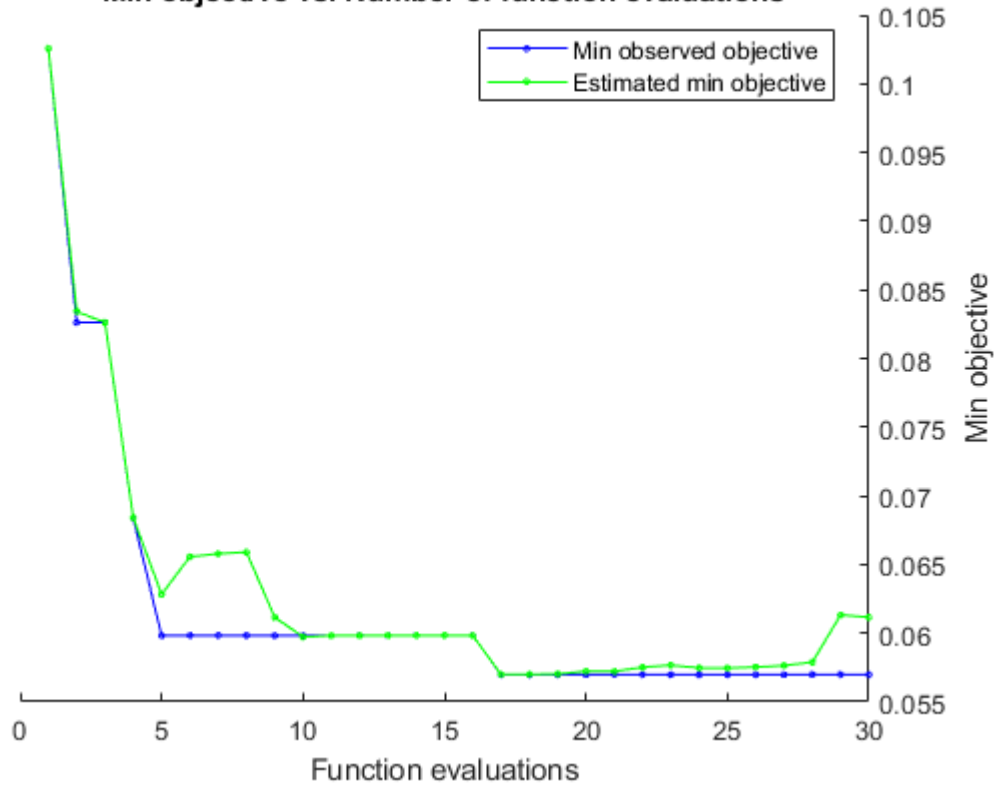
In this example, for reproducibility, set the random seed and use the 'expected-improvement-plus' acquisition function. Also, for reproducibility of random forest algorithm, specify the 'Reproducible' name-value pair argument as true for tree learners.

```
rng('default')
t = templateTree('Reproducible',true);
Mdl = fitcensemble(X,Y,'OptimizeHyperparameters','auto','Learners',t, ...
    'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName','expecti
```

Get

Iter	Eval	Objective	Objective	BestSoFar	BestSoFar	Method	NumLe
	result		runtime	(observed)	(estim.)		ycles
1	Best	0.10256	2.8201	0.10256	0.10256	RUSBoost	
2	Best	0.082621	6.3089	0.082621	0.083414	LogitBoost	
3	Accept	0.099715	4.0004	0.082621	0.082624	AdaBoostM1	
4	Best	0.068376	1.5887	0.068376	0.068395	Bag	
5	Best	0.059829	1.7618	0.059829	0.062829	LogitBoost	
6	Accept	0.068376	1.6662	0.059829	0.065561	LogitBoost	
7	Accept	0.088319	13.07	0.059829	0.065786	LogitBoost	
8	Accept	0.065527	0.79673	0.059829	0.065894	LogitBoost	
9	Accept	0.15385	0.93354	0.059829	0.061156	LogitBoost	
10	Accept	0.059829	3.8828	0.059829	0.059731	LogitBoost	
11	Accept	0.35897	2.3272	0.059829	0.059826	Bag	
12	Accept	0.068376	0.53634	0.059829	0.059825	Bag	
13	Accept	0.12251	9.5155	0.059829	0.059826	AdaBoostM1	
14	Accept	0.11966	4.9323	0.059829	0.059827	RUSBoost	
15	Accept	0.062678	4.2429	0.059829	0.059826	GentleBoost	
16	Accept	0.065527	3.0688	0.059829	0.059824	GentleBoost	
17	Best	0.05698	1.659	0.05698	0.056997	GentleBoost	
18	Accept	0.13675	2.0647	0.05698	0.057002	GentleBoost	
19	Accept	0.062678	2.4037	0.05698	0.05703	GentleBoost	
20	Accept	0.065527	1.029	0.05698	0.057228	GentleBoost	
Iter	Eval	Objective	Objective	BestSoFar	BestSoFar	Method	NumLe
	result		runtime	(observed)	(estim.)		ycles

	21		Accept		0.079772		0.44308		0.05698		0.057214		LogitBoost	
	22		Accept		0.065527		21.191		0.05698		0.057523		Bag	
	23		Accept		0.068376		20.294		0.05698		0.057671		Bag	
	24		Accept		0.64103		1.2793		0.05698		0.057468		RUSBoost	
	25		Accept		0.088319		0.53606		0.05698		0.057456		RUSBoost	
	26		Accept		0.074074		0.36802		0.05698		0.05753		AdaBoostM1	
	27		Accept		0.099715		12.133		0.05698		0.057646		AdaBoostM1	
	28		Accept		0.079772		10.877		0.05698		0.057886		AdaBoostM1	
	29		Accept		0.068376		12.326		0.05698		0.061326		GentleBoost	
	30		Accept		0.065527		0.3945		0.05698		0.061165		LogitBoost	

Min objective vs. Number of function evaluations

Optimization completed.

MaxObjectiveEvaluations of 30 reached.

Total function evaluations: 30

Total elapsed time: 165.9329 seconds

Total objective function evaluation time: 148.4504

Best observed feasible point:

Method	NumLearningCycles	LearnRate	MinLeafSize
GentleBoost	60	0.0010045	3

Observed objective function value = 0.05698

Estimated objective function value = 0.061165

Function evaluation time = 1.659

Best estimated feasible point (according to models):

Method	NumLearningCycles	LearnRate	MinLeafSize
GentleBoost	60	0.0010045	3

```

Estimated objective function value = 0.061165
Estimated function evaluation time = 1.6503
Mdl =
    ClassificationEnsemble
        ResponseName: 'Y'
    CategoricalPredictors: []
        ClassNames: {'b' 'g'}
        ScoreTransform: 'none'
        NumObservations: 351
    HyperparameterOptimizationResults: [1x1 BayesianOptimization]
        NumTrained: 60
        Method: 'GentleBoost'
        LearnerNames: {'Tree'}
    ReasonForTermination: 'Terminated normally after completing the requested num
        FitInfo: [60x1 double]
    FitInfoDescription: {2x1 cell}

```

Properties, Methods

The optimization searched over the ensemble aggregation methods for binary classification, over NumLearningCycles, over the LearnRate for applicable methods, and over the tree learner MinLeafSize. The output is the ensemble classifier with the minimum estimated cross-validation loss.


Optimize Classification Ensemble Using Cross-Validation

One way to create an ensemble of boosted classification trees that has satisfactory predictive performance is by tuning the decision tree complexity level using cross-validation. While searching for an optimal complexity level, tune the learning rate to minimize the number of learning cycles.

This example manually finds optimal parameters by using the cross-validation option (the 'KFold' name-value pair argument) and the kfoldLoss function. Alternatively, you can use the 'OptimizeHyperparameters' name-value pair argument to optimize hyperparameters automatically. See [Optimize Classification Ensemble](#).

Load the ionosphere data set.

```
load ionosphere
```

 Get ▾

To search for the optimal tree-complexity level:

1. Cross-validate a set of ensembles. Exponentially increase the tree-complexity level for subsequent ensembles from decision stump (one split) to at most $n - 1$ splits. n is the sample size. Also, vary the learning rate for each ensemble between 0.1 to 1.
2. Estimate the cross-validated misclassification rate of each ensemble.
3. For tree-complexity level j , $j = 1 \dots J$, compare the cumulative, cross-validated misclassification rate of the ensembles by plotting them against number of learning cycles. Plot separate curves for each learning rate on the same figure.
4. Choose the curve that achieves the minimal misclassification rate, and note the corresponding learning cycle and learning rate.

Cross-validate a deep classification tree and a stump. These classification trees serve as benchmarks.

```

rng(1) % For reproducibility
MdlDeep = fitctree(X,Y,'CrossVal','on','MergeLeaves','off', ...

```

 Get ▾


```
'MinParentSize',1);
MdlStump = fitctree(X,Y,'MaxNumSplits',1,'CrossVal','on');
```

Cross-validate an ensemble of 150 boosted classification trees using 5-fold cross-validation. Using a tree template, vary the maximum number of splits using the values in the sequence $\{3^0, 3^1, \dots, 3^m\}$. m is such that 3^m is no greater than $n - 1$. For each variant, adjust the learning rate using each value in the set $\{0.1, 0.25, 0.5, 1\}$;

```
n = size(X,1);
m = floor(log(n - 1)/log(3));
learnRate = [0.1 0.25 0.5 1];
numLR = numel(learnRate);
maxNumSplits = 3.^(0:m);
numMNS = numel(maxNumSplits);
numTrees = 150;
Mdl = cell(numMNS,numLR);

for k = 1:numLR
    for j = 1:numMNS
        t = templateTree('MaxNumSplits',maxNumSplits(j));
        Mdl{j,k} = fitcensemble(X,Y,'NumLearningCycles',numTrees,...
            'Learners',t,'KFold',5,'LearnRate',learnRate(k));
    end
end
```

 Get ▾

Estimate the cumulative, cross-validated misclassification rate for each ensemble and the classification trees serving as benchmarks.

```
kf1All = @(x)kfoldLoss(x,'Mode','cumulative');
errorCell = cellfun(kf1All,Mdl,'Uniform',false);
error = reshape(cell2mat(errorCell),[numTrees numel(maxNumSplits) numel(learnRate)]);
errorDeep = kfoldLoss(MdlDeep);
errorStump = kfoldLoss(MdlStump);
```

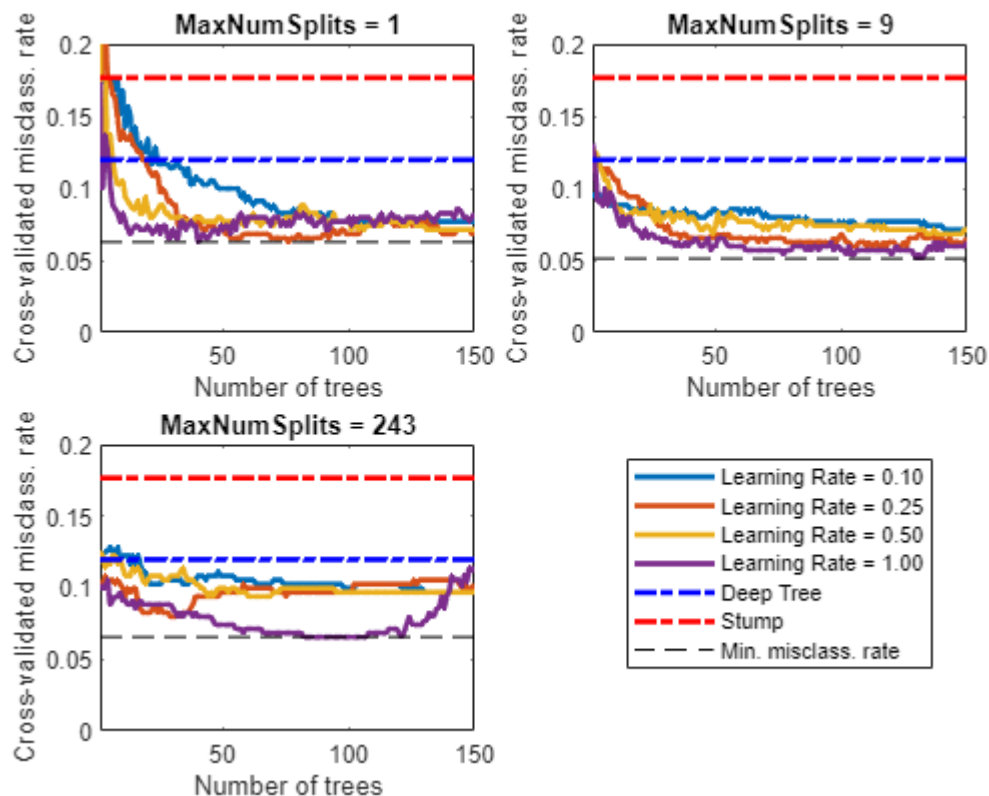
 Get ▾

Plot how the cross-validated misclassification rate behaves as the number of trees in the ensemble increases. Plot the curves with respect to learning rate on the same plot, and plot separate plots for varying tree-complexity levels. Choose a subset of tree complexity levels to plot.

```
mnsPlot = [1 round(numel(maxNumSplits)/2) numel(maxNumSplits)];
figure
for k = 1:3
    subplot(2,2,k)
    plot(squeeze(error(:,mnsPlot(k),:)), 'LineWidth',2)
    axis tight
    hold on
    h = gca;
    plot(h.XLim,[errorDeep errorDeep], '-.b', 'LineWidth',2)
    plot(h.XLim,[errorStump errorStump], '-.r', 'LineWidth',2)
    plot(h.XLim,min(min(error(:,mnsPlot(k),:))).*[1 1], '--k')
    h.YLim = [0 0.2];
    xlabel('Number of trees')
    ylabel('Cross-validated misclass. rate')
    title(sprintf('MaxNumSplits = %0.3g', maxNumSplits(mnsPlot(k))))
    hold off
end
hL = legend([cellstr(num2str(learnRate),'Learning Rate = %0.2f')); ...
```

 Get ▾

```
'Deep Tree';'Stump';'Min. misclass. rate']]);
hL.Position(1) = 0.6;
```



Each curve contains a minimum cross-validated misclassification rate occurring at the optimal number of trees in the ensemble.

Identify the maximum number of splits, number of trees, and learning rate that yields the lowest misclassification rate overall.

```
[minErr,minErrIdxLin] = min(error(:));
[idxNumTrees,idxMNS,idxLR] = ind2sub(size(error),minErrIdxLin);

fprintf('\nMin. misclass. rate = %0.5f',minErr)
```

☐ Get ▾

```
Min. misclass. rate = 0.05128
```

```
fprintf('\nOptimal Parameter Values:\nNum. Trees = %d',idxNumTrees);
```

☐ Get ▾

```
Optimal Parameter Values:
```

```
Num. Trees = 130
```

```
fprintf('\nMaxNumSplits = %d\nLearning Rate = %0.2f\n',...
    maxNumSplits(idxMNS),learnRate(idxLR))
```

☐ Get ▾

```
MaxNumSplits = 9
```

```
Learning Rate = 1.00
```

Create a predictive ensemble based on the optimal hyperparameters and the entire training set.

```
tFinal = templateTree('MaxNumSplits',maxNumSplits(idxMNS));
MdlFinal = fitcensemble(X,Y,'NumLearningCycles',idxNumTrees,...
    'Learners',tFinal,'LearnRate',learnRate(idxLR))
```

☐ Get ▾

```
MdlFinal =
    ClassificationEnsemble
        ResponseName: 'Y'
    CategoricalPredictors: []
```

```

    ClassNames: {'b' 'g'}
    ScoreTransform: 'none'
    NumObservations: 351
    NumTrained: 130
    Method: 'LogitBoost'
    LearnerNames: {'Tree'}
    ReasonForTermination: 'Terminated normally after completing the requested number of training iterations'
    FitInfo: [130x1 double]
    FitInfoDescription: {2x1 cell}

```

Properties, Methods

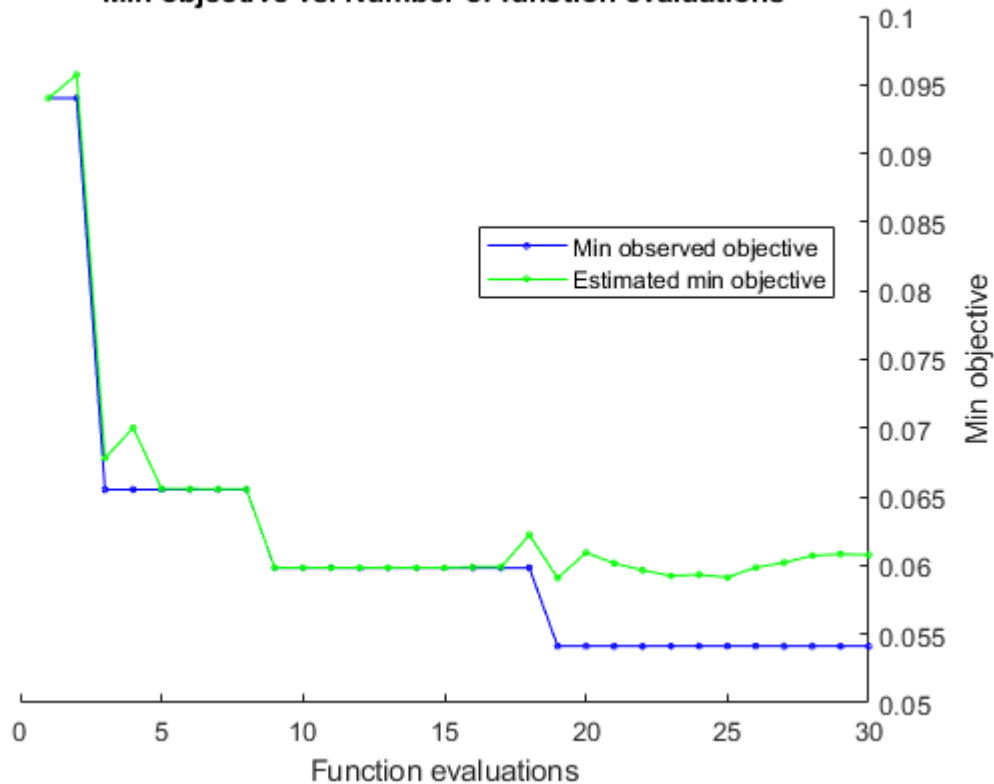
MdlFinal is a ClassificationEnsemble. To predict whether a radar return is good given predictor data, you can pass the predictor data and MdlFinal to predict.

Instead of searching optimal values manually by using the cross-validation option ('KFold') and the kfoldLoss function, you can use the 'OptimizeHyperparameters' name-value pair argument. When you specify 'OptimizeHyperparameters', the software finds optimal parameters automatically using Bayesian optimization. The optimal values obtained by using 'OptimizeHyperparameters' can be different from those obtained using manual search.

```
mdl = fitcensemble(X,Y,'OptimizeHyperparameters',{ 'NumLearningCycles','LearnRate' })
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	NumLearningCycles	LearnRate
1	Best	0.094017	3.7194	0.094017	0.094017	137	0.001
2	Accept	0.12251	0.66511	0.094017	0.095735	15	0.001
3	Best	0.065527	0.90035	0.065527	0.067815	31	0.001
4	Accept	0.19943	8.6107	0.065527	0.070015	340	0.001
5	Accept	0.071225	0.90081	0.065527	0.065583	32	0.001
6	Accept	0.099715	0.688	0.065527	0.065573	23	0.001
7	Accept	0.11681	0.90799	0.065527	0.065565	28	0.001
8	Accept	0.17379	0.82143	0.065527	0.065559	29	0.001
9	Best	0.059829	0.59677	0.059829	0.059844	18	0.001
10	Accept	0.11111	0.40132	0.059829	0.059843	10	0.001
11	Accept	0.08547	0.41121	0.059829	0.059842	10	0.001
12	Accept	0.11681	0.41538	0.059829	0.059841	10	0.001
13	Accept	0.082621	0.46504	0.059829	0.059842	10	0.001
14	Accept	0.079772	0.46297	0.059829	0.05984	11	0.001
15	Accept	0.088319	0.69297	0.059829	0.05984	19	0.001
16	Accept	0.062678	0.3637	0.059829	0.059886	10	0.001
17	Accept	0.065527	1.9404	0.059829	0.059887	78	0.001
18	Accept	0.065527	0.39816	0.059829	0.062228	11	0.001
19	Best	0.054131	0.36381	0.054131	0.059083	10	0.001
20	Accept	0.065527	0.38429	0.054131	0.060938	10	0.001
21	Accept	0.079772	0.40405	0.054131	0.060161	10	0.001
22	Accept	0.05698	0.37983	0.054131	0.059658	10	0.001
23	Accept	0.10826	0.36128	0.054131	0.059244	10	0.001
24	Accept	0.074074	0.38056	0.054131	0.05933	10	0.001

	25	Accept		0.11966		0.35336		0.054131		0.059132		10	
	26	Accept		0.065527		0.77041		0.054131		0.059859		26	
	27	Accept		0.068376		0.38116		0.054131		0.060205		10	
	28	Accept		0.062678		0.47015		0.054131		0.060713		14	
	29	Accept		0.11966		0.41033		0.054131		0.060826		10	
	30	Accept		0.08547		0.45352		0.054131		0.060771		10	

Min objective vs. Number of function evaluations

Optimization completed.

MaxObjectiveEvaluations of 30 reached.

Total function evaluations: 30

Total elapsed time: 41.5854 seconds

Total objective function evaluation time: 28.4744

Best observed feasible point:

NumLearningCycles	LearnRate	MaxNumSplits
10	0.69072	3

Observed objective function value = 0.054131

Estimated objective function value = 0.061741

Function evaluation time = 0.36381

Best estimated feasible point (according to models):

NumLearningCycles	LearnRate	MaxNumSplits
14	0.99445	3

Estimated objective function value = 0.060771

Estimated function evaluation time = 0.48009

mdl =

ClassificationEnsemble

```

        ResponseName: 'Y'
    CategoricalPredictors: []
        ClassNames: {'b' 'g'}
        ScoreTransform: 'none'
        NumObservations: 351
HyperparameterOptimizationResults: [1x1 BayesianOptimization]
        NumTrained: 14
        Method: 'LogitBoost'
        LearnerNames: {'Tree'}
ReasonForTermination: 'Terminated normally after completing the requested num
        FitInfo: [14x1 double]
        FitInfoDescription: {2x1 cell}

```

Properties, Methods

Input Arguments

[collapse all](#)

Tb1 — Sample data

table

Sample data used to train the model, specified as a table. Each row of Tb1 corresponds to one observation, and each column corresponds to one predictor variable. Tb1 can contain one additional column for the response variable. Multicolumn variables and cell arrays other than cell arrays of character vectors are not allowed.

- If Tb1 contains the response variable and you want to use all remaining variables as predictors, then specify the response variable using [ResponseVarName](#).
- If Tb1 contains the response variable, and you want to use a subset of the remaining variables only as predictors, then specify a formula using [formula](#).
- If Tb1 does not contain the response variable, then specify the response data using [Y](#). The length of response variable and the number of rows of Tb1 must be equal.



Note

To save memory and execution time, supply [X](#) and [Y](#) instead of Tb1.

Data Types: table

ResponseVarName — Response variable name

name of response variable in Tb1

Response variable name, specified as the name of the response variable in Tb1.

You must specify ResponseVarName as a character vector or string scalar. For example, if Tb1.Y is the response variable, then specify ResponseVarName as 'Y'. Otherwise, fitcensemble treats all columns of [Tb1](#) as predictor variables.

The response variable must be a categorical, character, or string array, logical or numeric vector, or cell array of character vectors. If the response variable is a character array, then each element must correspond to one row of the array.

For classification, you can specify the order of the classes using the [ClassNames](#) name-value pair argument. Otherwise, fitcensemble determines the class order, and stores it in the Mdl.ClassNames.

Data Types: char | string

formula — Explanatory model of response variable and subset of predictor variables

character vector | string scalar

Explanatory model of the response variable and a subset of the predictor variables, specified as a character vector or string scalar in the form "Y~x1+x2+x3". In this form, Y represents the response variable, and x1, x2, and x3 represent the predictor variables.

To specify a subset of variables in [Tb1](#) as predictors for training the model, use a formula. If you specify a formula, then the software does not use any variables in [Tb1](#) that do not appear in [formula](#).

The variable names in the formula must be both variable names in [Tb1](#) ([Tb1.Properties.VariableNames](#)) and valid MATLAB® identifiers. You can verify the variable names in [Tb1](#) by using the [isvarname](#) function. If the variable names are not valid, then you can convert them by using the [matlab.lang.makeValidName](#) function.

Data Types: char | string

X — Predictor data

numeric matrix

Predictor data, specified as numeric matrix.

Each row corresponds to one observation, and each column corresponds to one predictor variable.

The length of [Y](#) and the number of rows of X must be equal.

To specify the names of the predictors in the order of their appearance in X, use the [PredictorNames](#) name-value pair argument.

Data Types: single | double

Y — Response data

categorical array | character array | string array | logical vector | numeric vector | cell array of character vectors

Response data, specified as a categorical, character, or string array, logical or numeric vector, or cell array of character vectors. Each entry in Y is the response to or label for the observation in the corresponding row of [X](#) or [Tb1](#). The length of Y and the number of rows of X or [Tb1](#) must be equal. If the response variable is a character array, then each element must correspond to one row of the array.

You can specify the order of the classes using the [ClassNames](#) name-value pair argument. Otherwise, [fitcensemble](#) determines the class order, and stores it in the [Md1.ClassNames](#).

Data Types: categorical | char | string | logical | single | double | cell

Name-Value Arguments

[expand all](#)

Specify optional pairs of arguments as Name1=Value1, ..., NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name in quotes.

Example: 'CrossVal', 'on', 'LearnRate', 0.05 specifies to implement 10-fold cross-validation and to use 0.05 as the learning rate.

Note

You cannot use any cross-validation name-value argument together with the `OptimizeHyperparameters` name-value argument. You can modify the cross-validation for `OptimizeHyperparameters` only by using the `HyperparameterOptimizationOptions` name-value argument.

General Ensemble Options

[collapse all](#)

Method — Ensemble aggregation method

'Bag' | 'Subspace' | 'AdaBoostM1' | 'AdaBoostM2' | 'GentleBoost' | 'LogitBoost' | 'LPBoost' | 'RobustBoost' | 'RUSBoost' | 'TotalBoost'

Ensemble aggregation method, specified as the comma-separated pair consisting of 'Method' and one of the following values.

Value	Method	Classification Problem Support	Related Name-Value Pair Arguments
'Bag'	Bootstrap aggregation (bagging, for example, random forest [2]) — If 'Method' is 'Bag', then <code>fitcensemble</code> uses bagging with random predictor selections at each split (random forest) by default. To use bagging without the random selections, use tree learners whose <code>'NumVariablesToSample'</code> value is 'all' or use discriminant analysis learners.	Binary and multiclass	N/A
'Subspace'	Random subspace	Binary and multiclass	<code>NPredToSample</code>
'AdaBoostM1'	Adaptive boosting	Binary only	<code>LearnRate</code>
'AdaBoostM2'	Adaptive boosting	Multiclass only	<code>LearnRate</code>
'GentleBoost'	Gentle adaptive boosting	Binary only	<code>LearnRate</code>
'LogitBoost'	Adaptive logistic regression	Binary only	<code>LearnRate</code>
'LPBoost'	Linear programming boosting — Requires Optimization Toolbox™	Binary and multiclass	<code>MarginPrecision</code>
'RobustBoost'	Robust boosting — Requires Optimization Toolbox	Binary only	<code>RobustErrorGoal</code> , <code>RobustMarginSigma</code> , <code>RobustMaxMargin</code>

Value	Method	Classification Problem Support	Related Name-Value Pair Arguments
'RUSBoost'	Random undersampling boosting	Binary and multiclass	LearnRate, RatioToSmallest
'TotalBoost'	Totally corrective boosting — Requires Optimization Toolbox	Binary and multiclass	MarginPrecision

You can specify sampling options ([FResample](#), [Replace](#), [Resample](#)) for training data when you use bagging ('Bag') or boosting ('TotalBoost', 'RUSBoost', 'AdaBoostM1', 'AdaBoostM2', 'GentleBoost', 'LogitBoost', 'RobustBoost', or 'LPBoost').

The defaults are:

- 'LogitBoost' for binary problems and 'AdaBoostM2' for multiclass problems if '[Learners](#)' includes only tree learners
- 'AdaBoostM1' for binary problems and 'AdaBoostM2' for multiclass problems if '[Learners](#)' includes both tree and discriminant analysis learners
- 'Subspace' if '[Learners](#)' does not include tree learners

For details about ensemble aggregation algorithms and examples, see [Algorithms](#), [Tips](#), [Ensemble Algorithms](#), and [Choose an Applicable Ensemble Aggregation Method](#).

Example: 'Method', 'Bag'

NumLearningCycles — Number of ensemble learning cycles
100 (default) | positive integer | 'AllPredictorCombinations'

Number of ensemble learning cycles, specified as the comma-separated pair consisting of 'NumLearningCycles' and a positive integer or 'AllPredictorCombinations'.

- If you specify a positive integer, then, at every learning cycle, the software trains one weak learner for every template object in [Learners](#). Consequently, the software trains NumLearningCycles*numel(Learners) learners.
- If you specify 'AllPredictorCombinations', then set Method to 'Subspace' and specify one learner only for Learners. With these settings, the software trains learners for all possible combinations of predictors taken [NPredToSample](#) at a time. Consequently, the software trains [nchoosek](#)(size(X,2),NPredToSample) learners.

The software composes the ensemble using all trained learners and stores them in Md1.Trained.

For more details, see [Tips](#).

Example: 'NumLearningCycles',500

Data Types: single | double | char | string

Learners — Weak learners to use in ensemble
'discriminant' | 'knn' | 'tree' | weak-learner template object | cell vector of weak-learner template objects

Weak learners to use in the ensemble, specified as the comma-separated pair consisting of 'Learners' and a weak-learner name, weak-learner template object, or cell vector of weak-learner template objects.

Weak Learner	Weak-Learner Name	Template Object Creation Function	Method Setting
Discriminant analysis	'discriminant'	templateDiscriminant	Recommended for 'Subspace'
k -nearest neighbors	'knn'	templateKNN	For 'Subspace' only
Decision tree	'tree'	templateTree	All methods except 'Subspace'

- Weak-learner name ('discriminant', 'knn', or 'tree') — `fitcensemble` uses weak learners created by a template object creation function with default settings. For example, specifying 'Learners', 'discriminant' is the same as specifying 'Learners', `templateDiscriminant()`. See the template object creation function pages for the default settings of a weak learner.
- Weak-learner template object — `fitcensemble` uses the weak learners created by a template object creation function. Use the name-value pair arguments of the template object creation function to specify the settings of the weak learners.
- Cell vector of m weak-learner template objects — `fitcensemble` grows m learners per learning cycle (see [NumLearningCycles](#)). For example, for an ensemble composed of two types of classification trees, supply {t1 t2}, where t1 and t2 are classification tree template objects returned by `templateTree`.

The default 'Learners' value is 'knn' if 'Method' is 'Subspace'.

The default 'Learners' value is 'tree' if 'Method' is 'Bag' or any boosting method. The default values of `templateTree()` depend on the value of 'Method'.

- For bagged decision trees, the maximum number of decision splits ('[MaxNumSplits](#)') is $n-1$, where n is the number of observations. The number of predictors to select at random for each split ('[NumVariablesToSample](#)') is the square root of the number of predictors. Therefore, `fitcensemble` grows deep decision trees. You can grow shallower trees to reduce model complexity or computation time.
- For boosted decision trees, 'MaxNumSplits' is 10 and 'NumVariablesToSample' is 'all'. Therefore, `fitcensemble` grows shallow decision trees. You can grow deeper trees for better accuracy.

See [templateTree](#) for the default settings of a weak learner. To obtain reproducible results, you must specify the '[Reproducible](#)' name-value pair argument of `templateTree` as true if 'NumVariablesToSample' is not 'all'.

For details on the number of learners to train, see [NumLearningCycles](#) and [Tips](#).

Example: 'Learners', `templateTree('MaxNumSplits',5)`

NPrint — Printout frequency

"off" (default) | positive integer

Printout frequency, specified as a positive integer or "off".

To track the number of *weak learners* or *folds* that `fitcensemble` trained so far, specify a positive integer. That is, if you specify the positive integer m :

- Without also specifying any cross-validation option (for example, [CrossVal](#)), then `fitcensemble` displays a message to the command line every time it completes training m weak learners.
- And a cross-validation option, then `fitcensemble` displays a message to the command line every time it finishes training m folds.

If you specify "off", then `fitcensemble` does not display a message when it completes training weak learners.

i Tip

For fastest training of some boosted decision trees, set `NPrint` to the default value "off". This tip holds when the classification Method is "AdaBoostM1", "AdaBoostM2", "GentleBoost", or "LogitBoost", or when the regression Method is "LSBoost".

Example: `NPrint=5`

Data Types: `single` | `double` | `char` | `string`

NumBins — Number of bins for numeric predictors

`[]`(empty) (default) | positive integer scalar

Number of bins for numeric predictors, specified as a positive integer scalar. This argument is valid only when `fitcensemble` uses a tree learner, that is, `Learners` is either "tree" or a template object created by using `templateTree`.

- If the `NumBins` value is empty (default), then `fitcensemble` does not bin any predictors.
- If you specify the `NumBins` value as a positive integer scalar (`numBins`), then `fitcensemble` bins every numeric predictor into at most `numBins` equiprobable bins, and then grows trees on the bin indices instead of the original data.
 - The number of bins can be less than `numBins` if a predictor has fewer than `numBins` unique values.
 - `fitcensemble` does not bin categorical predictors.

When you use a large training data set, this binning option speeds up training but might cause a potential decrease in accuracy. You can try "NumBins", 50 first, and then change the value depending on the accuracy and training speed.

A trained model stores the bin edges in the `BinEdges` property.

Example: "NumBins", 50

Data Types: `single` | `double`

CategoricalPredictors — Categorical predictors list

vector of positive integers | logical vector | character matrix | string array | cell array of character vectors | 'all'

Categorical predictors list, specified as one of the values in this table.

Value	Description
Vector of positive integers	Each entry in the vector is an index value indicating that the corresponding predictor is categorical. The index values are between 1 and <code>p</code> , where <code>p</code> is the number of predictors used to train the model. If <code>fitcensemble</code> uses a subset of input variables as predictors, then the function indexes the predictors using only the subset. The <code>CategoricalPredictors</code> values do not count any response variable, observation weights variable, or other variable that the function does not use.
Logical vector	A true entry means that the corresponding predictor is categorical. The length of the vector is <code>p</code> .

Value	Description
Character matrix	Each row of the matrix is the name of a predictor variable. The names must match the entries in PredictorNames . Pad the names with extra blanks so each row of the character matrix has the same length.
String array or cell array of character vectors	Each element in the array is the name of a predictor variable. The names must match the entries in PredictorNames .
"all"	All predictors are categorical.

Specification of 'CategoricalPredictors' is appropriate if:

- 'Learners' specifies tree learners.
- 'Learners' specifies k -nearest learners where all predictors are categorical.

Each learner identifies and treats categorical predictors in the same way as the fitting function corresponding to the learner. See 'CategoricalPredictors' of [fitcknn](#) for k -nearest learners and 'CategoricalPredictors' of [fitctree](#) for tree learners.

Example: 'CategoricalPredictors','all'

Data Types: single | double | logical | char | string | cell

PredictorNames — Predictor variable names

string array of unique names | cell array of unique character vectors

Predictor variable names, specified as a string array of unique names or cell array of unique character vectors. The functionality of PredictorNames depends on the way you supply the training data.

- If you supply [X](#) and [Y](#), then you can use PredictorNames to assign names to the predictor variables in [X](#).
 - The order of the names in PredictorNames must correspond to the column order of [X](#). That is, PredictorNames{1} is the name of [X\(:,1\)](#), PredictorNames{2} is the name of [X\(:,2\)](#), and so on. Also, [size\(X,2\)](#) and [numel\(PredictorNames\)](#) must be equal.
 - By default, PredictorNames is {'x1','x2',...}.
- If you supply [Tbl](#), then you can use PredictorNames to choose which predictor variables to use in training. That is, [fitcensemble](#) uses only the predictor variables in PredictorNames and the response variable during training.
 - PredictorNames must be a subset of [Tbl.Properties.VariableNames](#) and cannot include the name of the response variable.
 - By default, PredictorNames contains the names of all predictor variables.
 - A good practice is to specify the predictors for training using either PredictorNames or [formula](#), but not both.

Example: "PredictorNames",["SepalLength","SepalWidth","PetalLength","PetalWidth"]

Data Types: string | cell

ResponseName — Response variable name

"Y" (default) | character vector | string scalar

Response variable name, specified as a character vector or string scalar.

- If you supply **Y**, then you can use `ResponseName` to specify a name for the response variable.
- If you supply `ResponseVarName` or `formula`, then you cannot use `ResponseName`.

Example: `ResponseName="response"`

Data Types: char | string

Parallel Options

[collapse all](#)

Options — Options for computing in parallel and setting random numbers structure

Options for computing in parallel and setting random numbers, specified as a structure. Create the Options structure using `statset`.



Note

You need Parallel Computing Toolbox™ to run computations in parallel.

This table describes the option fields and their values.

Field Name	Value	Default
<code>UseParallel</code>	Set this value to true to compute in parallel. Parallel ensemble training requires you to set the <code>Method</code> name-value argument to "Bag". Parallel training is available only for tree learners, the default type for <code>Method</code> ="Bag".	false
<code>UseSubstreams</code>	Set this value to true to perform computations in a reproducible manner. To compute reproducibly, set <code>Streams</code> to a type that allows substreams: "mlfg6331_64" or "mrg32k3a". Also, use a tree template with the <code>Reproducible</code> name-value argument set to true. See Reproducibility in Parallel Statistical Computations .	false
<code>Streams</code>	Specify this value as a RandStream object or cell array of such objects. Use a single object except when the <code>UseParallel</code> value is true and the <code>UseSubstreams</code> value is false. In that case, use a cell array that has the same size as the parallel pool.	If you do not specify <code>Streams</code> , <code>fitcensemble</code> uses the default stream or streams.

For an example using reproducible parallel training, see [Train Classification Ensemble in Parallel](#).

For dual-core systems and above, `fitcensemble` parallelizes training using Intel® Threading Building Blocks (TBB). Therefore, specifying the `UseParallel` option as `true` might not provide a significant speedup on a single computer. For details on Intel TBB, see

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html>.

Example: `Options=statset(UseParallel=true)`

Data Types: `struct`

Cross-Validation Options

[collapse all](#)

CrossVal — Cross-validation flag

'off' (default) | 'on'

Cross-validation flag, specified as the comma-separated pair consisting of 'Crossval' and 'on' or 'off'.

If you specify 'on', then the software implements 10-fold cross-validation.

To override this cross-validation setting, use one of these name-value pair arguments: [CVPartition](#), [Holdout](#), [KFold](#), or [Leaveout](#). To create a cross-validated model, you can use one cross-validation name-value pair argument at a time only.

Alternatively, cross-validate later by passing `Mdl` to [crossval](#).

Example: 'Crossval', 'on'

CVPartition — Cross-validation partition

[] (default) | `cvpartition` object

Cross-validation partition, specified as a [cvpartition](#) object that specifies the type of cross-validation and the indexing for the training and validation sets.

To create a cross-validated model, you can specify only one of these four name-value arguments: `CVPartition`, [Holdout](#), [KFold](#), or [Leaveout](#).

Example: Suppose you create a random partition for 5-fold cross-validation on 500 observations by using `cvp = cvpartition(500,KFold=5)`. Then, you can specify the cross-validation partition by setting `CVPartition=cvp`.

Holdout — Fraction of data for holdout validation

scalar value in the range (0,1)

Fraction of the data used for holdout validation, specified as a scalar value in the range (0,1). If you specify `Holdout=p`, then the software completes these steps:

1. Randomly select and reserve $p \cdot 100\%$ of the data as validation data, and train the model using the rest of the data.
2. Store the compact trained model in the `Trained` property of the cross-validated model.

To create a cross-validated model, you can specify only one of these four name-value arguments: [CVPartition](#), `Holdout`, [KFold](#), or [Leaveout](#).

Example: `Holdout=0.1`

Data Types: `double` | `single`

KFold — Number of folds

10 (default) | positive integer value greater than 1

Number of folds to use in the cross-validated model, specified as a positive integer value greater than 1. If you specify `KFold=k`, then the software completes these steps:

1. Randomly partition the data into k sets.
2. For each set, reserve the set as validation data, and train the model using the other $k - 1$ sets.
3. Store the k compact trained models in a k -by-1 cell vector in the `Trained` property of the cross-validated model.

To create a cross-validated model, you can specify only one of these four name-value arguments: `CVPartition`, `Holdout`, `KFold`, or `Leaveout`.

Example: `KFold=5`

Data Types: `single` | `double`

Leaveout — Leave-one-out cross-validation flag

"off" (default) | "on"

Leave-one-out cross-validation flag, specified as "on" or "off". If you specify `Leaveout="on"`, then for each of the n observations (where n is the number of observations, excluding missing observations, specified in the `NumObservations` property of the model), the software completes these steps:

1. Reserve the one observation as validation data, and train the model using the other $n - 1$ observations.
2. Store the n compact trained models in an n -by-1 cell vector in the `Trained` property of the cross-validated model.

To create a cross-validated model, you can specify only one of these four name-value arguments: `CVPartition`, `Holdout`, `KFold`, or `Leaveout`.

Example: `Leaveout="on"`

Data Types: `char` | `string`

Other Classification Options[collapse all](#)**ClassNames — Names of classes to use for training**

categorical array | character array | string array | logical vector | numeric vector | cell array of character vectors

Names of classes to use for training, specified as a categorical, character, or string array; a logical or numeric vector; or a cell array of character vectors. `ClassNames` must have the same data type as the response variable in `Tbl` or `Y`.

If `ClassNames` is a character array, then each element must correspond to one row of the array.

Use `ClassNames` to:

- Specify the order of the classes during training.
- Specify the order of any input or output argument dimension that corresponds to the class order. For example, use `ClassNames` to specify the order of the dimensions of `Cost` or the column order of classification scores returned by `predict`.
- Select a subset of classes for training. For example, suppose that the set of all distinct class names in `Y` is `["a", "b", "c"]`. To train the model using observations from classes "a" and "c" only, specify

ClassNames=["a","c"].

The default value for ClassNames is the set of all distinct class names in the response variable in Tb1 or Y.

Example: ClassNames=["b","g"]

Data Types: categorical | char | string | logical | single | double | cell

Cost — Misclassification cost

square matrix | structure array

Misclassification cost, specified as the comma-separated pair consisting of 'Cost' and a square matrix or structure. If you specify:

- The square matrix Cost, then Cost(i,j) is the cost of classifying a point into class j if its true class is i. That is, the rows correspond to the true class and the columns correspond to the predicted class. To specify the class order for the corresponding rows and columns of Cost, also specify the [ClassNames](#) name-value pair argument.
- The structure S, then it must have two fields:
 - S.ClassNames, which contains the class names as a variable of the same data type as Y
 - S.ClassificationCosts, which contains the cost matrix with rows and columns ordered as in S.ClassNames

The default is ones(K) - eye(K), where K is the number of distinct classes.

fitcensemble uses Cost to adjust the prior class probabilities specified in [Prior](#). Then, fitcensemble uses the adjusted prior probabilities for training.

Example: 'Cost',[0 1 2 ; 1 0 2; 2 2 0]

Data Types: double | single | struct

Prior — Prior probabilities

'empirical' (default) | 'uniform' | numeric vector | structure array

Prior probabilities for each class, specified as the comma-separated pair consisting of 'Prior' and a value in this table.

Value	Description
'empirical'	The class prior probabilities are the class relative frequencies in Y.
'uniform'	All class prior probabilities are equal to 1/K, where K is the number of classes.
numeric vector	Each element is a class prior probability. Order the elements according to Md1.ClassNames or specify the order using the ClassNames name-value pair argument. The software normalizes the elements such that they sum to 1.
structure array	A structure S with two fields: <ul style="list-style-type: none"> • S.ClassNames contains the class names as a variable of the same type as Y. • S.ClassProbs contains a vector of corresponding prior probabilities. The

Value	Description
	software normalizes the elements such that they sum to 1.

fitcensemble normalizes the prior probabilities in Prior to sum to 1.

Example: struct('ClassNames',{ 'setosa', 'versicolor', 'virginica' }, 'ClassProbs', 1:3)

Data Types: char | string | double | single | struct

ScoreTransform — Score transformation

"none" (default) | "doublelogit" | "invlogit" | "ismax" | "logit" | function handle | ...

Score transformation, specified as a character vector, string scalar, or function handle.

This table summarizes the available character vectors and string scalars.

Value	Description
"doublelogit"	$1/(1 + e^{-2x})$
"invlogit"	$\log(x / (1 - x))$
"ismax"	Sets the score for the class with the largest score to 1, and sets the scores for all other classes to 0
"logit"	$1/(1 + e^{-x})$
"none" or "identity"	x (no transformation)
"sign"	-1 for $x < 0$ 0 for $x = 0$ 1 for $x > 0$
"symmetric"	$2x - 1$
"symmetricismax"	Sets the score for the class with the largest score to 1, and sets the scores for all other classes to -1
"symmetriclogit"	$2/(1 + e^{-x}) - 1$

For a MATLAB function or a function you define, use its function handle for the score transform. The function handle must accept a matrix (the original scores) and return a matrix of the same size (the transformed scores).

Example: ScoreTransform="logit"

Data Types: char | string | function_handle

Weights — Observation weights

numeric vector of positive values | name of variable in Tb1

Observation weights, specified as a numeric vector of positive values or name of a variable in [Tb1](#). The software weighs the observations in each row of X or Tb1 with the corresponding value in Weights. The size of Weights must equal the number of rows of X or Tb1.

If you specify the input data as a table `Tbl`, then `Weights` can be the name of a variable in `Tbl` that contains a numeric vector. In this case, you must specify `Weights` as a character vector or string scalar. For example, if the weights vector `W` is stored as `Tbl.W`, then specify it as `"W"`. Otherwise, the software treats all columns of `Tbl`, including `W`, as predictors or the response when training the model.

By default, `Weights` is `ones(n,1)`, where n is the number of observations in `X` or `Tbl`.

The software normalizes `Weights` to sum up to the value of the prior probability in the respective class. Inf weights are not supported.

Data Types: `double` | `single` | `char` | `string`

Sampling Options for Boosting Methods and Bagging

[collapse all](#)

FResample — Fraction of training set to resample

1 (default) | positive scalar in (0,1]

Fraction of the training set to resample for every weak learner, specified as a positive scalar in (0,1]. To use 'FResample', set `Resample` to 'on'.

Example: 'FResample',0.75

Data Types: `single` | `double`

Replace — Flag indicating to sample with replacement

'on' (default) | 'off'

Flag indicating sampling with replacement, specified as the comma-separated pair consisting of 'Replace' and 'off' or 'on'.

- For 'on', the software samples the training observations with replacement.
- For 'off', the software samples the training observations without replacement. If you set `Resample` to 'on', then the software samples training observations assuming uniform weights. If you also specify a boosting method, then the software boosts by reweighting observations.

Unless you set `Method` to 'bag' or set `Resample` to 'on', `Replace` has no effect.

Example: 'Replace','off'

Resample — Flag indicating to resample

'off' | 'on'

Flag indicating to resample, specified as the comma-separated pair consisting of 'Resample' and 'off' or 'on'.

- If `Method` is a boosting method, then:
 - 'Resample','on' specifies to sample training observations using updated weights as the multinomial sampling probabilities.
 - 'Resample','off' (default) specifies to reweight observations at every learning iteration.
- If `Method` is 'bag', then 'Resample' must be 'on'. The software resamples a fraction of the training observations (see `FResample`) with or without replacement (see `Replace`).

If you specify to resample using `Resample`, then it is good practice to resample to entire data set. That is, use the default setting of 1 for `FResample`.

AdaBoostM1, AdaBoostM2, LogitBoost, and GentleBoost Method Options

[collapse all](#)

LearnRate — Learning rate for shrinkage

1 (default) | numeric scalar in (0,1]

Learning rate for shrinkage, specified as the comma-separated pair consisting of 'LearnRate' and a numeric scalar in the interval (0,1].

To train an ensemble using shrinkage, set `LearnRate` to a value less than 1, for example, 0.1 is a popular choice. Training an ensemble using shrinkage requires more learning iterations, but often achieves better accuracy.

Example: 'LearnRate',0.1

Data Types: single | double

RUSBoost Method Options

[collapse all](#)

LearnRate — Learning rate for shrinkage

1 (default) | numeric scalar in (0,1]

Learning rate for shrinkage, specified as the comma-separated pair consisting of 'LearnRate' and a numeric scalar in the interval (0,1].

To train an ensemble using shrinkage, set `LearnRate` to a value less than 1, for example, 0.1 is a popular choice. Training an ensemble using shrinkage requires more learning iterations, but often achieves better accuracy.

Example: 'LearnRate',0.1

Data Types: single | double

RatioToSmallest — Sampling proportion with respect to lowest-represented class

positive numeric scalar | numeric vector of positive values

Sampling proportion with respect to the lowest-represented class, specified as the comma-separated pair consisting of 'RatioToSmallest' and a numeric scalar or numeric vector of positive values with length equal to the number of distinct classes in the training data.

Suppose that there are K classes in the training data and the lowest-represented class has m observations in the training data.

- If you specify the positive numeric scalar s , then `fitcensemble` samples $s*m$ observations from each class, that is, it uses the same sampling proportion for each class. For more details, see [Algorithms](#).
- If you specify the numeric vector $[s_1, s_2, \dots, s_K]$, then `fitcensemble` samples s_i*m observations from class i , $i = 1, \dots, K$. The elements of `RatioToSmallest` correspond to the order of the class names specified using [ClassNames](#) (see [Tips](#)).

The default value is `ones(K,1)`, which specifies to sample m observations from each class.

Example: 'RatioToSmallest',[2,1]

Data Types: single | double

LPBoost and TotalBoost Method Options

[collapse all](#)

MarginPrecision — Margin precision to control convergence speed

0.1 (default) | numeric scalar in [0,1]

Margin precision to control convergence speed, specified as the comma-separated pair consisting of 'MarginPrecision' and a numeric scalar in the interval [0,1]. MarginPrecision affects the number of boosting iterations required for convergence.



Tip

To train an ensemble using many learners, specify a small value for MarginPrecision. For training using a few learners, specify a large value.

Example: 'MarginPrecision',0.5

Data Types: single | double

RobustBoost Method Options

[collapse all](#)

RobustErrorGoal — Target classification error

0.1 (default) | nonnegative numeric scalar

Target classification error, specified as the comma-separated pair consisting of 'RobustErrorGoal' and a nonnegative numeric scalar. The upper bound on possible values depends on the values of [RobustMarginSigma](#) and [RobustMaxMargin](#). However, the upper bound cannot exceed 1.



Tip

For a particular training set, usually there is an optimal range for RobustErrorGoal. If you set it too low or too high, then the software can produce a model with poor classification accuracy. Try cross-validating to search for the appropriate value.

Example: 'RobustErrorGoal',0.05

Data Types: single | double

RobustMarginSigma — Classification margin distribution spread

0.1 (default) | positive numeric scalar

Classification margin distribution spread over the training data, specified as the comma-separated pair consisting of 'RobustMarginSigma' and a positive numeric scalar. Before specifying RobustMarginSigma, consult the literature on RobustBoost, for example, [\[3\]](#).

Example: 'RobustMarginSigma',0.5

Data Types: single | double

RobustMaxMargin — Maximal classification margin

0 (default) | nonnegative numeric scalar

Maximal classification margin in the training data, specified as the comma-separated pair consisting of 'RobustMaxMargin' and a nonnegative numeric scalar. The software minimizes the number of observations in the training data having classification margins below RobustMaxMargin.

Example: 'RobustMaxMargin',1

Data Types: single | double

Random Subspace Method Options[collapse all](#)**NPredToSample — Number of predictors to sample**

1 (default) | positive integer

Number of predictors to sample for each random subspace learner, specified as the comma-separated pair consisting of 'NPredToSample' and a positive integer in the interval $1, \dots, p$, where p is the number of predictor variables (`size(X,2)` or `size(Tbl,2)`).

Data Types: single | double

Hyperparameter Optimization Options[collapse all](#)**OptimizeHyperparameters — Parameters to optimize**

'none' (default) | 'auto' | 'all' | string array or cell array of eligible parameter names | vector of `optimizableVariable` objects

Parameters to optimize, specified as the comma-separated pair consisting of 'OptimizeHyperparameters' and one of the following:

- 'none' — Do not optimize.
- 'auto' — Use {'Method', 'NumLearningCycles', 'LearnRate'} along with the default parameters for the specified [Learners](#):
 - Learners = 'tree' (default) — {'MinLeafSize'}
 - Learners = 'discriminant' — {'Delta', 'Gamma'}
 - Learners = 'knn' — {'Distance', 'NumNeighbors', 'Standardize'}

**Note**

For hyperparameter optimization, Learners must be a single argument, not a string array or cell array.

- 'all' — Optimize all eligible parameters.
- String array or cell array of eligible parameter names
- Vector of `optimizableVariable` objects, typically the output of [hyperparameters](#)

The optimization attempts to minimize the cross-validation loss (error) for `fitcensemble` by varying the parameters. To control the cross-validation type and other aspects of the optimization, use the [HyperparameterOptimizationOptions](#) name-value argument. When you use `HyperparameterOptimizationOptions`, you can use the (compact) model size instead of the cross-validation loss as the optimization objective by setting the `ConstraintType` and `ConstraintBounds` options.

**Note**

The values of `OptimizeHyperparameters` override any values you specify using other name-value arguments. For example, setting `OptimizeHyperparameters` to "auto" causes `fitcensemble` to optimize hyperparameters corresponding to the "auto" option and to ignore any specified values for the hyperparameters.

The eligible parameters for `fitcensemble` are:

- **Method** — Depends on the number of classes.
 - Two classes — Eligible methods are 'Bag', 'GentleBoost', 'LogitBoost', 'AdaBoostM1', and 'RUSBoost'.
 - Three or more classes — Eligible methods are 'Bag', 'AdaBoostM2', and 'RUSBoost'.
- **NumLearningCycles** — `fitcensemble` searches among positive integers, by default log-scaled with range [10,500].
- **LearnRate** — `fitcensemble` searches among positive reals, by default log-scaled with range [1e-3,1].
- The eligible hyperparameters for the chosen Learners:

Learners	Eligible Hyperparameters Bold = Used By Default	Default Range
'discriminant'	Delta	Log-scaled in the range [1e-6,1e3]
	DiscrimType	'linear', 'quadratic', 'diagLinear', 'diagQuadratic', 'pseudoLinear', and 'pseudoQuadratic'
	Gamma	Real values in [0,1]
'knn'	Distance	'cityblock', 'chebychev', 'correlation', 'cosine', 'euclidean', 'hamming', 'jaccard', 'mahalanobis', 'minkowski', 'seuclidean', and 'spearman'
	DistanceWeight	'equal', 'inverse', and 'squaredinverse'
	Exponent	Positive values in [0.5,3]
	NumNeighbors	Positive integer values log-scaled in the range [1, max(2,round(NumObservations/2))]
	Standardize	'true' and 'false'
'tree'	MaxNumSplits	Integers log-scaled in the range [1,max(2,NumObservations-1)]
	MinLeafSize	Integers log-scaled in the range [1,max(2,floor(NumObservations/2))]
	NumVariablesToSample	Integers in the range [1,max(2,NumPredictors)]
	SplitCriterion	'gdi', 'deviance', and 'twoing'

Alternatively, use [hyperparameters](#) with your chosen Learners. Note that you must specify the predictor data and response when creating an `optimizableVariable` object.

```
load fisheriris
params = hyperparameters('fitcensemble',meas,species,'Tree');
```

To see the eligible and default hyperparameters, examine `params`.

Set nondefault parameters by passing a vector of `optimizableVariable` objects that have nondefault values. For example,

```
load fisheriris
params = hyperparameters('fitcensemble',meas,species,'Tree');
params(4).Range = [1,30];
```

Pass `params` as the value of `OptimizeHyperparameters`.

By default, the iterative display appears at the command line, and plots appear according to the number of hyperparameters in the optimization. For the optimization and plots, the objective function is the misclassification rate. To control the iterative display, set the `Verbose` option of the `HyperparameterOptimizationOptions` name-value argument. To control the plots, set the `ShowPlots` field of the `HyperparameterOptimizationOptions` name-value argument.

For an example, see [Optimize Classification Ensemble](#).

Example: `'OptimizeHyperparameters',`
`{'Method','NumLearningCycles','LearnRate','MinLeafSize','MaxNumSplits'}`

HyperparameterOptimizationOptions — Options for optimization
HyperparameterOptimizationOptions object | structure

Options for optimization, specified as a [HyperparameterOptimizationOptions](#) object or a structure. This argument modifies the effect of the [OptimizeHyperparameters](#) name-value argument. If you specify `HyperparameterOptimizationOptions`, you must also specify `OptimizeHyperparameters`. All the options are optional. However, you must set `ConstraintBounds` and `ConstraintType` to return `AggregateOptimizationResults`. The options that you can set in a structure are the same as those in the `HyperparameterOptimizationOptions` object.

Option	Values	Default
Optimizer	<ul style="list-style-type: none">"bayesopt" — Use Bayesian optimization. Internally, this setting calls bayesopt."gridsearch" — Use grid search with <code>NumGridDivisions</code> values per dimension. "gridsearch" searches in a random order, using uniform sampling without replacement from the grid. After optimization, you can get a table in grid order by using the command <code>sortrows(Mdl.HyperparameterOptimizationResults)</code>."randomsearch" — Search at random among <code>MaxObjectiveEvaluations</code> points.	"bayes"
ConstraintBounds	Constraint bounds for N optimization problems, specified as an N -by-2 numeric matrix or <code>[]</code> . The columns of <code>ConstraintBounds</code> contain the lower and upper bound values of the optimization problems. If you specify <code>ConstraintBounds</code> as a numeric vector, the software assigns the values to the second column of <code>ConstraintBounds</code> , and zeros to the first	<code>[]</code>

Option	Values	Default
	column. If you specify <code>ConstraintBounds</code> , you must also specify <code>ConstraintType</code> .	
<code>ConstraintTarget</code>	Constraint target for the optimization problems, specified as "matlab" or "coder". If <code>ConstraintBounds</code> and <code>ConstraintType</code> are [] and you set <code>ConstraintTarget</code> , then the software sets <code>ConstraintTarget</code> to []. The values of <code>ConstraintTarget</code> and <code>ConstraintType</code> determine the objective and constraint functions. For more information, see HyperparameterOptimizationOptions .	If you s Constr and Constr then the value is Otherw default
<code>ConstraintType</code>	Constraint type for the optimization problems, specified as "size" or "loss". If you specify <code>ConstraintType</code> , you must also specify <code>ConstraintBounds</code> . The values of <code>ConstraintTarget</code> and <code>ConstraintType</code> determine the objective and constraint functions. For more information, see HyperparameterOptimizationOptions .	[]
<code>AcquisitionFunctionName</code>	Type of acquisition function: <ul style="list-style-type: none"> "expected-improvement-per-second-plus" "expected-improvement" "expected-improvement-plus" "expected-improvement-per-second" "lower-confidence-bound" "probability-of-improvement" Acquisition functions whose names include per-second do not yield reproducible results, because the optimization depends on the run time of the objective function. Acquisition functions whose names include plus modify their behavior when they overexploit an area. For more details, see Acquisition Function Types .	"expec improv second
<code>MaxObjectiveEvaluations</code>	Maximum number of objective function evaluations. If you specify multiple optimization problems using <code>ConstraintBounds</code> , the value of <code>MaxObjectiveEvaluations</code> applies to each optimization problem individually.	30 for " and "rando and the for "gr
<code>MaxTime</code>	Time limit for the optimization, specified as a nonnegative real scalar. The time limit is in seconds, as measured by tic and toc . The software performs at least one optimization iteration, regardless of the value of <code>MaxTime</code> . The run time can exceed <code>MaxTime</code> because <code>MaxTime</code> does not interrupt function evaluations. If you specify multiple optimization problems using <code>ConstraintBounds</code> , the time limit applies to each optimization problem individually.	Inf
<code>NumGridDivisions</code>	For <code>Optimizer="gridsearch"</code> , the number of values in each dimension. The value can be a vector of positive integers giving the number of values for each dimension, or a scalar that applies to all dimensions. The software ignores this option for categorical variables.	10
<code>ShowPlots</code>	Logical value indicating whether to show plots of the optimization progress. If this option is true, the software plots the best observed objective function value against the iteration	true

Option	Values	Default
	number. If you use Bayesian optimization (Optimizer="bayesopt"), the software also plots the best estimated objective function value. The best observed objective function values and best estimated objective function values correspond to the values in the BestSoFar (observed) and BestSoFar (estim.) columns of the iterative display, respectively. You can find these values in the properties ObjectiveMinimumTrace and EstimatedObjectiveMinimumTrace of <code>Mdl.HyperparameterOptimizationResults</code> . If the problem includes one or two optimization parameters for Bayesian optimization, then <code>ShowPlots</code> also plots a model of the objective function against the parameters.	
SaveIntermediateResults	Logical value indicating whether to save the optimization results. If this option is true, the software overwrites a workspace variable named "BayesoptResults" at each iteration. The variable is a BayesianOptimization object. If you specify multiple optimization problems using <code>ConstraintBounds</code> , the workspace variable is an AggregateBayesianOptimization object named "AggregateBayesoptResults".	false
Verbose	Display level at the command line: <ul style="list-style-type: none"> 0 – No iterative display 1 – Iterative display 2 – Iterative display with additional information For details, see the bayesopt Verbose name-value argument and the example Optimize Classifier Fit Using Bayesian Optimization .	1
UseParallel	Logical value indicating whether to run the Bayesian optimization in parallel, which requires Parallel Computing Toolbox. Due to the nonreproducibility of parallel timing, parallel Bayesian optimization does not necessarily yield reproducible results. For details, see Parallel Bayesian Optimization .	false
Repartition	Logical value indicating whether to repartition the cross-validation at every iteration. If this option is false, the optimizer uses a single partition for the optimization. A value of true usually gives the most robust results because this setting takes partitioning noise into account. However, for optimal results, true requires at least twice as many function evaluations.	false
Specify only one of the following three options.		
CVPartition	cvpartition object created by cvpartition	KFold= not spe validati
Holdout	Scalar in the range (0,1) representing the holdout fraction	
KFold	Integer greater than 1	

Example: `HyperparameterOptimizationOptions=struct(UseParallel=true)`

Output Arguments

[collapse all](#)

Mdl — Trained classification ensemble model

ClassificationBaggedEnsemble model object | ClassificationEnsemble model object |

ClassificationPartitionedEnsemble cross-validated model object

Trained ensemble model, returned as one of the model objects in this table.

Model Object	Specify Any Cross-Validation Options?	Method Setting	Resample Setting
ClassificationBaggedEnsemble	No	'Bag'	'on'
ClassificationEnsemble	No	Any ensemble aggregation method for classification	'off'
ClassificationPartitionedEnsemble	Yes	Any ensemble aggregation method for classification	'off' or 'on'

The name-value pair arguments that control cross-validation are [CrossVal](#), [Holdout](#), [KFold](#), [Leaveout](#), and [CVPartition](#).

To reference properties of Mdl, use dot notation. For example, to access or display the cell vector of weak learner model objects for an ensemble that has not been cross-validated, enter Mdl.Trained at the command line.

If you specify [OptimizeHyperparameters](#) and set the ConstraintType and ConstraintBounds options of [HyperparameterOptimizationOptions](#), then Mdl is an N -by-1 cell array of model objects, where N is equal to the number of rows in ConstraintBounds. If none of the optimization problems yields a feasible model, then each cell array value is [].

AggregateOptimizationResults — Aggregate optimization results

AggregateBayesianOptimization object

Aggregate optimization results for multiple optimization problems, returned as an [AggregateBayesianOptimization](#) object. To return AggregateOptimizationResults, you must specify [OptimizeHyperparameters](#) and [HyperparameterOptimizationOptions](#). You must also specify the ConstraintType and ConstraintBounds options of HyperparameterOptimizationOptions. For an example that shows how to produce this output, see [Hyperparameter Optimization with Multiple Constraint Bounds](#).

Tips

- NumLearningCycles can vary from a few dozen to a few thousand. Usually, an ensemble with good predictive power requires from a few hundred to a few thousand weak learners. However, you do not have to train an ensemble for that many cycles at once. You can start by growing a few dozen learners, inspect the ensemble performance and then, if necessary, train more weak learners using [resume](#) for classification problems.
- Ensemble performance depends on the ensemble setting and the setting of the weak learners. That is, if you specify weak learners with default parameters, then the ensemble can perform poorly. Therefore, like ensemble

settings, it is good practice to adjust the parameters of the weak learners using templates, and to choose values that minimize generalization error.

- If you specify to resample using [Resample](#), then it is good practice to resample to entire data set. That is, use the default setting of 1 for [FResample](#).
- If the ensemble aggregation method ([Method](#)) is 'bag' and:
 - The misclassification cost ([Cost](#)) is highly imbalanced, then, for in-bag samples, the software oversamples unique observations from the class that has a large penalty.
 - The class prior probabilities ([Prior](#)) are highly skewed, the software oversamples unique observations from the class that has a large prior probability.

For smaller sample sizes, these combinations can result in a low relative frequency of out-of-bag observations from the class that has a large penalty or prior probability. Consequently, the estimated out-of-bag error is highly variable and it can be difficult to interpret. To avoid large estimated out-of-bag error variances, particularly for small sample sizes, set a more balanced misclassification cost matrix using [Cost](#) or a less skewed prior probability vector using [Prior](#).

- Because the order of some input and output arguments correspond to the distinct classes in the training data, it is good practice to specify the class order using the [ClassNames](#) name-value pair argument.
 - To determine the class order quickly, remove all observations from the training data that are unclassified (that is, have a missing label), obtain and display an array of all the distinct classes, and then specify the array for [ClassNames](#). For example, suppose the response variable (Y) is a cell array of labels. This code specifies the class order in the variable `classNames`.

```
Ycat = categorical(Y);
classNames = categories(Ycat)
```

[categorical](#) assigns <undefined> to unclassified observations and [categories](#) excludes <undefined> from its output. Therefore, if you use this code for cell arrays of labels or similar code for categorical arrays, then you do not have to remove observations with missing labels to obtain a list of the distinct classes.

- To specify that the class order from lowest-represented label to most-represented, then quickly determine the class order (as in the previous bullet), but arrange the classes in the list by frequency before passing the list to [ClassNames](#). Following from the previous example, this code specifies the class order from lowest- to most-represented in `classNamesLH`.

```
Ycat = categorical(Y);
classNames = categories(Ycat);
freq = countcats(Ycat);
[~,idx] = sort(freq);
classNamesLH = classNames(idx);
```

- After training a model, you can generate C/C++ code that predicts labels for new data. Generating C/C++ code requires MATLAB Coder™. For details, see [Introduction to Code Generation](#).

Algorithms

- For details of ensemble aggregation algorithms, see [Ensemble Algorithms](#).
- If you set [Method](#) to be a boosting algorithm and [Learners](#) to be decision trees, then the software grows shallow decision trees by default. You can adjust tree depth by specifying the `MaxNumSplits`, `MinLeafSize`, and `MinParentSize` name-value pair arguments using [templateTree](#).
- If you specify the [Cost](#), [Prior](#), and [Weights](#) name-value arguments, the output model object stores the specified values in the `Cost`, `Prior`, and `W` properties, respectively. The `Cost` property stores the user-specified cost matrix (`C`) without modification. The `Prior` and `W` properties store the prior probabilities and observation weights, respectively, after normalization. For model training, the software updates the prior probabilities and observation weights to incorporate the penalties described in the cost matrix. For details, see [Misclassification Cost Matrix, Prior Probabilities, and Observation Weights](#).

- For bagging ('Method', 'Bag'), fitcensemble generates in-bag samples by oversampling classes with large misclassification costs and undersampling classes with small misclassification costs. Consequently, out-of-bag samples have fewer observations from classes with large misclassification costs and more observations from classes with small misclassification costs. If you train a classification ensemble using a small data set and a highly skewed cost matrix, then the number of out-of-bag observations per class can be low. Therefore, the estimated out-of-bag error can have a large variance and can be difficult to interpret. The same phenomenon can occur for classes with large prior probabilities.
- For the RUSBoost ensemble aggregation method ('Method', 'RUSBoost'), the name-value pair argument `RatioToSmallest` specifies the sampling proportion for each class with respect to the lowest-represented class. For example, suppose that there are two classes in the training data: *A* and *B*. *A* has 100 observations and *B* has 10 observations. Suppose also that the lowest-represented class has *m* observations in the training data.
 - If you set 'RatioToSmallest', 2, then $s*m = 2*10 = 20$. Consequently, fitcensemble trains every learner using 20 observations from class *A* and 20 observations from class *B*. If you set 'RatioToSmallest', [2 2], then you obtain the same result.
 - If you set 'RatioToSmallest', [2,1], then $s1*m = 2*10 = 20$ and $s2*m = 1*10 = 10$. Consequently, fitcensemble trains every learner using 20 observations from class *A* and 10 observations from class *B*.
- For dual-core systems and above, fitcensemble parallelizes training using Intel Threading Building Blocks (TBB). For details on Intel TBB, see <https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html>.

References

- [1] Breiman, L. "Bagging Predictors." *Machine Learning*. Vol. 26, pp. 123–140, 1996.
- [2] Breiman, L. "Random Forests." *Machine Learning*. Vol. 45, pp. 5–32, 2001.
- [3] Freund, Y. "A more robust boosting algorithm." *arXiv:0905.2138v1*, 2009.
- [4] Freund, Y. and R. E. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." *J. of Computer and System Sciences*, Vol. 55, pp. 119–139, 1997.
- [5] Friedman, J. "Greedy function approximation: A gradient boosting machine." *Annals of Statistics*, Vol. 29, No. 5, pp. 1189–1232, 2001.
- [6] Friedman, J., T. Hastie, and R. Tibshirani. "Additive logistic regression: A statistical view of boosting." *Annals of Statistics*, Vol. 28, No. 2, pp. 337–407, 2000.
- [7] Hastie, T., R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning* section edition, Springer, New York, 2008.
- [8] Ho, T. K. "The random subspace method for constructing decision forests." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 8, pp. 832–844, 1998.
- [9] Schapire, R. E., Y. Freund, P. Bartlett, and W.S. Lee. "Boosting the margin: A new explanation for the effectiveness of voting methods." *Annals of Statistics*, Vol. 26, No. 5, pp. 1651–1686, 1998.
- [10] Seiffert, C., T. Khoshgoftaar, J. Hulse, and A. Napolitano. "RUSBoost: Improving classification performance when training data is skewed." *19th International Conference on Pattern Recognition*, pp. 1–4, 2008.
- [11] Warmuth, M., J. Liao, and G. Ratsch. "Totally corrective boosting algorithms that maximize the margin." *Proc. 23rd Int'l. Conf. on Machine Learning*, ACM, New York, pp. 1001–1008, 2006.

Extended Capabilities

[expand all](#)

Automatic Parallel Support

Accelerate code by automatically running computation in parallel using Parallel Computing Toolbox™.

GPU Arrays

Accelerate code by running on a graphics processing unit (GPU) using Parallel Computing Toolbox™.

Version History

Introduced in R2016b

[expand all](#)

R2025a: Compute serially when parallel hyperparameter optimization is not available ⚠

R2023b: "auto" option of `OptimizeHyperparameters` includes `Standardize` when weak learners are *k*-nearest neighbor (KNN) classifiers ⚠

See Also

[ClassificationEnsemble](#) | [ClassificationBaggedEnsemble](#) | [ClassificationPartitionedEnsemble](#) | [templateDiscriminant](#) | [templateKNN](#) | [templateTree](#) | [predict](#)

Topics

[Supervised Learning Workflow and Algorithms](#)

[Framework for Ensemble Learning](#)

[Ensemble Algorithms](#)