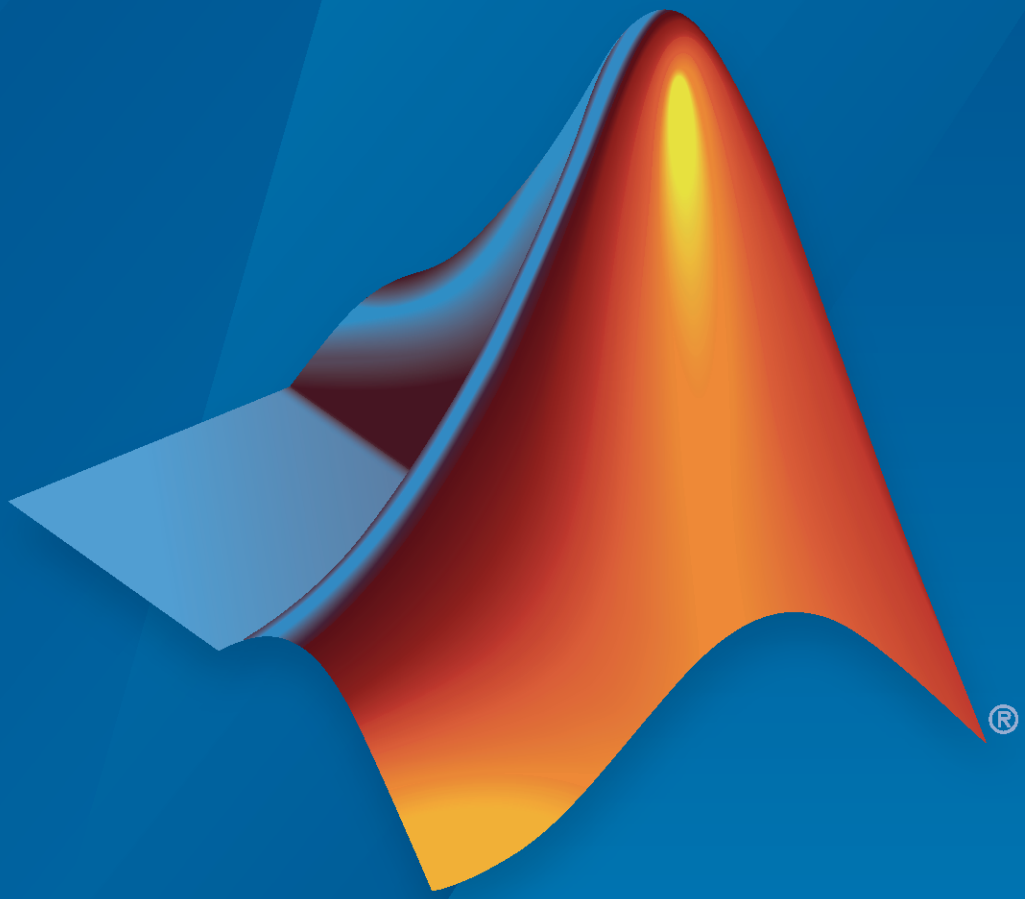
Statistics and Machine Learning Toolbox™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Statistics and Machine Learning Toolbox™ Release Notes

Contents

R2023a

R2022b

R2022a

R2021b

R2021a

R2020b

R2020a

R2019b

R2019a

R2018b

R2018a

R2017b

R2017a

R2015b

R2015a

R2014b

R2014a

R2013b

R2013a

R2012b

R2012a

R2011b

R2011a

R2010b

R2010a

R2009b

R2009a

R2008b

R2008a

R2007b

R2007a

R2006b

R2006a

R14SP3

R14SP2

R2023a

Deployment

Linear Model and ECOC Model Prediction Blocks: Simulate a model and generate code in Simulink

You can now integrate the `predict` function of a linear model object or an error-correcting output codes (ECOC) model object into Simulink® using the new prediction blocks instead of a MATLAB® Function block. You can import a trained model and configure data types using the Block Parameters dialog box.

- RegressionLinear Predict — This block predicts responses for new data using an imported linear regression model object (`RegressionLinear`).
- ClassificationLinear Predict — This block returns classified labels and predicted scores (optional) for new data using an imported linear classification model object (`ClassificationLinear`) for binary classification.
- ClassificationECOC Predict — This block returns classified labels, predicted scores (optional), and positive-class scores (optional) for new data using an imported ECOC classification model object (`ClassificationECOC` or `CompactClassificationECOC`) for multiclass classification. The block supports an ECOC model with linear binary learners and support vector machine (SVM) binary learners.

The blocks support C/C++ code generation and fixed-point conversion. You can generate C and C++ code using Simulink Coder™, and design and simulate fixed-point systems using Fixed-Point Designer™.

Generate C/C++ code for prediction using Gaussian kernel classification and regression models (requires MATLAB Coder)

- You can now generate C/C++ code for the `predict` function for classification, which classifies observations by using a trained Gaussian kernel classification model (`ClassificationKernel`).
- Also, you can generate C/C++ code for the `predict` function for regression, which predicts responses by using a trained Gaussian kernel regression model (`RegressionKernel`).

Generate fixed-point C/C++ code for prediction using a lookup table that approximates the score transformation function of a classifier

You can generate a lookup table that approximates a score transformation function of a trained classifier, and then use the lookup table for fixed-point code generation. The approach requires fewer calculations for score transformation in the generated code than the default approach, which uses the CORDIC-based algorithm. Therefore, using a lookup table yields relatively high-speed performance and relatively low memory requirements.

After creating a fixed-point data type structure using the function generated by `generateLearnerDataTypeFcn`, update the structure to include the lookup table by using a `FunctionApproximation.TransformFunction` object and its function `approximate`. The supported score transformation functions include `"doublelogit"`, `"logit"`, and `"symmetriclogit"`. For an example, see “Use Lookup Table to Approximate Score Transformation”.

Example of verifying and validating machine learning models using model-based design

A new example, “Verify and Validate Machine Learning Models Using Model-Based Design”, shows how to use Simulink to verify and validate machine learning models. The example builds models for the acoustic scene classification (ASC) task, which classifies environments from the sounds they produce, and imports the models to Simulink. Then the example verifies and validates the models using these features:

- Examine performance using Simulink Profiler and SIL/PIL Manager (requires Embedded Coder®)
- Measure runtime memory usage using the Static Code Metrics Report (requires Embedded Coder)
- Perform code coverage analysis using Simulink Coverage™
- Run a back-to-back test using Simulink Test™

Apps

Experiment Manager App: Export machine learning models from Classification Learner or Regression Learner to Experiment Manager

After training a machine learning model in Classification Learner or Regression Learner, you can export the model to **Experiment Manager** to perform multiple experiments. On the **Classification Learner** or **Regression Learner** tab, in the **Export** section, click **Export Model** and select **Create Experiment**. In the dialog box, modify the file names or accept the default values. Then, click **Create Experiment**. The app opens Experiment Manager and a dialog box, in which you can choose to use a new or existing project for your experiment.

In Experiment Manager, use different hyperparameters and hyperparameter search ranges to tune your model. On the tab for your experiment, in the **Hyperparameters** section, click **Add** to add a hyperparameter to the model tuning process. In the table, double-click an entry to adjust its value. When are you ready to run your experiment, click **Run** in the **Run** section of the **Experiment Manager** tab.

(optional) To change the training data set, preprocessing steps, metric used during model tuning, or generated visualizations, you must adjust the experiment training function before running the experiment. To update the training function, click **Edit** in the **Training Function** section on the experiment tab.

For more information, see “Export Model from Classification Learner to Experiment Manager” and “Tune Classification Model Using Experiment Manager” (for classification) and “Export Model from Regression Learner to Experiment Manager” and “Tune Regression Model Using Experiment Manager” (for regression).

Classification Learner App: Train logistic regression and linear SVM models using efficiently trained linear classifiers

The Classification Learner app now contains two new efficiently trained linear classifiers: logistic regression and support vector machine (SVM). These models use the `fitclinear` function (for binary data) and the `fitcecoc` function (for multiclass data), both of which employ techniques that reduce the training computation time at the cost of some accuracy. When training on data with many predictors or many observations, consider using efficiently trained linear classifiers instead of the existing binary GLM logistic regression or linear SVM preset models. Note that "binary GLM logistic regression" is the new name for the logistic regression model preset that uses `fitglm`. For more information, see “Efficiently Trained Linear Classifiers”.

Machine Learning Apps: Compare models according to model size

After you train a model in Classification Learner or Regression Learner, you can check the size of a compact version of the final exported model, trained on the full data set (including training and validation data, but excluding test data). Select the model in the **Models** pane, and then check the **Model size (Compact)** value in the **Training Results** section of the model **Summary** tab.

To compare the size of multiple models, use the results table. On the **Classification Learner** or **Regression Learner** tab, in the **Models** section, click **Results Table**. Click the "Select columns to display" button at the top right of the table. In the **Select Columns to Display** dialog box, select

Model Size (bytes) and click **OK**. In the table, sort the models by clicking the arrows for the **Model Size (bytes)** column.

Note that nearest neighbor classifiers do not have compact versions that omit the data used during training. As a result, these models tend to have larger sizes.

Machine Learning Apps: Save a compact session

In Classification Learner and Regression Learner, you can save a compact version of the current app session to reduce the size of the resulting saved session file. On the **Classification Learner** or **Regression Learner** tab, in the **File** section, select the **Save Compact Session As** option from the **Save** menu. In the Save Learner Session dialog box, specify the session file name and the file location. Click **Save**. The app permanently deletes the training data from all trained models in the current session.

Classification Learner App: Neural network classifiers support misclassification costs

In the Classification Learner app, you can specify misclassification costs for neural network classifiers. Before you train a draft neural network model, select **Costs** in the **Options** section of the **Classification Learner** tab. In the dialog box, specify the misclassification costs (cost matrix). For more information, see “Misclassification Costs in Classification Learner App”.

Regression Learner App: Specify the number of predictors to sample for ensembles

For ensemble models in the Regression Learner app, you can specify the number of predictors to select at random for each split in the tree learners. On the **Summary** tab, under **Model Hyperparameters**, specify the **Number of predictors to sample** value.

- When you set this option to `Select All`, the software uses all available predictors.
- When you set this option to `Set Limit`, you can specify a value by clicking the buttons or entering a positive integer value in the box.

For more information, see “Ensemble Model Hyperparameter Options”.

Regression Learner App: Box plots use the boxchart function

The Regression Learner app now uses the MATLAB `boxchart` function instead of the Statistics and Machine Learning Toolbox `boxplot` function to display box plots. You can use box plots to visualize categorical predictor variables in the response and residual plots. For more information, see “Explore Data and Results in Response Plot” and “Evaluate Model Using Residuals Plot”.

Functionality being removed or changed

Machine Learning Apps: Export a compact model using the Export Model option

Behavior change

In Classification Learner and Regression Learner, you can export a full or compact model using the same **Export Model** option. In previous releases, you had to use the **Export Compact Model** option

to export a compact model. To export a model to the MATLAB workspace, select **Export Model** from the **Export** option in the **Classification Learner** or **Regression Learner** tab, and select **Export Model**. The app opens a dialog box, where you can edit the name of the exported structure and choose whether or not to include the training data in your model. To exclude the training data and export a compact model, clear the check box in the dialog box and click **OK**. The check box will be disabled if the model does not have training data or if it’s not possible to exclude training data from the model. For more information, see “Export Classification Model to Predict New Data” and “Export Regression Model to Predict New Data”.

Machine Learning

Fairness in Machine Learning: Optimize the score threshold for a binary classifier while satisfying fairness bounds

To mitigate the effects of a sensitive attribute on binary classification, you can use the `fairnessThresholder` function to modify your predictions after training a binary classifier. The function searches for an optimal score threshold to maximize accuracy while satisfying fairness bounds. For observations in the critical region below the optimal threshold, the function adjusts the labels so that the fairness constraints hold for the reference and nonreference groups in the sensitive attribute. By default, the function tries to find a threshold that results in a disparate impact value between 0.8 and 1.25. You can change the bias metric and its bounds by using the `BiasMetric` and `BiasMetricRange` name-value arguments, respectively.

After you create a `fairnessThresholder` object, you can use the `predict` and `loss` object functions on new data to predict fairness labels and calculate the classification loss, respectively.

fairnessMetrics Function: Compare fairness metrics across multiple binary classifiers

You can compare fairness metrics across multiple binary classifiers by using the `fairnessMetrics` function. In the call to the function, use the `predictions` argument and specify the predicted class labels for each model. To specify the names of the models, use the `ModelNames` name-value argument.

After you create a `fairnessMetrics` object, use the `report` or `plot` object function.

- The `report` object function returns a fairness metrics table, whose format depends on the value of the `DisplayMetricsInRows` name-value argument. (For more information, see `metricsTbl`.) You can specify a subset of models to include in the report table by using the `ModelNames` name-value argument.
- The `plot` object function returns a bar graph as an array of `Bar` objects. The bar colors indicate the models whose predicted labels are used to compute the specified metric. You can specify a subset of models to include in the plot by using the `ModelNames` name-value argument.

Compatibility Considerations

In previous releases, `fairnessMetrics`, `report`, and `plot` had the following behavior:

- The first and second variables in the `BiasMetrics` and `GroupMetrics` properties of the `fairnessMetrics` object always corresponded to the sensitive attribute name (`SensitiveAttributeNames` column) and the group name (`Groups` column), respectively. For more information on the current behavior, see `BiasMetrics` and `GroupMetrics`.
- Each row of the table returned by `report` contained fairness metrics for a group in a sensitive attribute. The first and second variables in the table always corresponded to the sensitive attribute name (`SensitiveAttributeNames`) and the group name (`Groups`), respectively. The rest of the variables corresponded to the bias and group metrics specified by the `BiasMetrics` and `GroupMetrics` name-value arguments, respectively. For more information on the current behavior, see `metricsTbl`.

- The `b = plot(__)` syntax always returned a single `Bar` object. `plot` displayed blue bars with black edges for the metric values and a solid line for the baseline value. Now, the color of each bar edge matches the color of its bar, and the plot includes a dashed line for the baseline value.

Neural network classifiers support misclassification costs and prior probabilities

`fitcnet` and `fitcauto` support misclassification costs and prior probabilities for neural network classifiers. Use the `Cost` and `Prior` name-value arguments, respectively.

`fitcnet` uses misclassification costs for prediction, but not training. You can change the `Cost` property of a trained neural network classifier object (`ClassificationNeuralNetwork`, `CompactClassificationNeuralNetwork`, or `ClassificationPartitionedModel`) by using dot notation.

Compatibility Considerations

In previous releases, when you specified nondefault misclassification costs or prior probabilities, `fitcauto` omitted neural network models from the model selection process.

gencfeatures and genrfeatures Functions: Use neighborhood component analysis (NCA) to select features automatically generated from a data set with categorical predictors

Before training an SVM model with a Gaussian kernel, you can engineer new features from a data set that contains a mix of numeric and categorical predictors, and use neighborhood component analysis (NCA) as the method for selecting new features (`FeatureSelectionMethod="nca"`).

- To generate features before training a binary SVM classifier, use `gencfeatures`.
- To generate features before training a regression SVM model, use `genrfeatures`.

When using either function, specify `TargetLearner` as `"gaussian-svm"`.

In previous releases, the NCA feature selection method was available for data sets with numeric predictors only.

fitrauto Function: Gaussian Process Regression (GPR) learners can include ARD kernels

The `fitrauto` function includes ARD kernel options for Gaussian process regression (`"gp"`) learners when the `OptimizeHyperparameters` value is `"all"` and the data set has these characteristics:

- The number of observations is less than or equal to 10,000.
- The number of predictors is less than or equal to 32 after the expansion of the categorical predictors (see “Automatic Creation of Dummy Variables”).

When the `KernelFunction` value of a GPR model is an ARD kernel option, `fitrauto` trains the model using these GPR tolerance and initial step size values:

- The `OptimizerOptions` value matches the following `options` structure:

```
options = statset("fitrgp");
options.MaxIter = 1000;
options.TolFun = 1e-3;
options.TolX = 1e-3;
```

- The `InitialStepSize` value is `"auto"`.

Compatibility Considerations

In previous releases, the `fitrauto` function ignored all ARD kernel options for GPR learners, regardless of the size of the data set.

shapley Function: Support for Linear SHAP and Tree SHAP algorithms

`shapley` now supports the Linear SHAP [1] algorithm for linear models and the Tree SHAP [2] algorithm for tree models and ensemble models of tree learners. Before R2023a, `shapley` supported only the Kernel SHAP algorithm to compute interventional Shapley values. The two new algorithms are computationally less expensive than the Kernel SHAP algorithm. For details, see “Interventional Algorithms”.

If you specify the `Method` name-value argument as `"interventional"` when you create a `shapley` object, the software selects an algorithm based on the machine learning model type and other specified options. The `Method` property of the returned `shapley` object stores the name of the selected algorithm.

Compatibility Considerations

The supported values of the `Method` name-value argument have changed from `"interventional-kernel"` and `"conditional-kernel"` to `"interventional"` and `"conditional"`, respectively.

Anomaly Detection: Detect outliers and novelties using the robust random cut forest algorithm

Use the `rrcforest` function, `RobustRandomCutForest` object, and `isanomaly` object function for anomaly detection. This feature uses the robust random cut forest algorithm [1][2].

- Outlier detection (detecting anomalies in training data) — Detect anomalies in training data by passing the data to the `rrcforest` function. The function returns a `RobustRandomCutForest` object, a logical array indicating anomalies, and a numeric array of anomaly scores. You can specify the fraction of anomalies in the data by using the `ContaminationFraction` name-value argument.
- Novelty detection (detecting anomalies in new data with uncontaminated training data) — Create a `RobustRandomCutForest` object by passing uncontaminated training data (data with no outliers) to the `rrcforest` function. Detect anomalies in new data by passing the object and the new data to the object function `isanomaly`. The `isanomaly` function returns a logical array indicating anomalies and a numeric array of anomaly scores. The function identifies observations with scores above the score threshold as anomalies. You can specify the score threshold by using the `ScoreThreshold` name-value argument.

The anomaly score value is between `0` and `inf`. A small positive value indicates a normal observation, and a large positive value indicates an anomaly.

You can perform the computation in parallel by specifying the `UseParallel` name-value argument of `rrcforest` or `isanomaly` as `true`. This option requires Parallel Computing Toolbox™.

GPU Support: regress now accepts gpuArray inputs (requires Parallel Computing Toolbox)

The `regress` function is enhanced to execute on a GPU and fully support GPU arrays.

For a full list of Statistics and Machine Learning Toolbox functions that accept GPU arrays, see Function List (GPU Arrays).

GPU Support: fitrsvm now accepts gpuArray inputs (requires Parallel Computing Toolbox)

- The `fitrsvm` function is enhanced to accept GPU array input arguments so that it can execute on a GPU, with some limitations.
- Most object functions of the `RegressionSVM`, `CompactRegressionSVM`, and `RegressionPartitionedSVM` models now support GPU array input arguments so that the functions can execute on a GPU. The following object functions do not support GPU array input arguments.

Object Function	Model Type
`incrementalLearner`	`RegressionSVM` and `CompactRegressionSVM`
`lime`	`RegressionSVM` and `CompactRegressionSVM`
`shapley`	`RegressionSVM` and `CompactRegressionSVM`
`update`	`CompactRegressionSVM`

- You can now use the `gather` function to create a `RegressionSVM`, `CompactRegressionSVM`, or `RegressionPartitionedSVM` object from an equivalent object fitted with GPU array data. The properties of the new object are stored in the local workspace.

GPU Performance: Improved performance in training a classification support vector machine (requires Parallel Computing Toolbox)

The `fitcsvm` function has improved performance when it executes on a GPU.

This code for `fitcsvm` is about 2x faster than in the previous release.

```
% Generate one class of points inside the unit disk in two dimensions,
% and another class of points in the annulus from radius 1 to radius 2.
gpurng(1); % For reproducibility
N = 100000;
r = sqrt(gpuArray.rand(N,1)); % Radius
t = 2*pi*gpuArray.rand(N,1);  % Angle

r2 = sqrt(3*rand(N,1)+1); % Radius
t2 = 2*pi*rand(N,1);      % Angle
```

```
x1 = [r.*cos(t), r.*sin(t)];
x2 = [r2.*cos(t2), r2.*sin(t2)];
x = [x1;x2];
y = gpuArray.ones(2*N,1);
y(1:N) = -1;

% Create a classification SVM model.
mdl = fitcsvm(x,y,'KernelFunction','rbf', 'BoxConstraint',10);

% Measure the execution time of the fit function.
fitFcn = @() fitcsvm(x,y,'KernelFunction','rbf', 'BoxConstraint',10);
t_gpu = gputimeit(fitFcn)
```

The approximate execution times are:

- R2023a: 4.59s
- R2022b: 9.62s

These codes were timed on a Windows® 10 test system with Intel® Xeon® CPU E5-1650 v3 @ 3.5 GHz and NVIDIA® Titan V with 12 GB of GPU memory.

Example of monitoring equipment state of health using drift-aware learning

A new example, “Monitor Equipment State of Health Using Drift-Aware Learning”, shows how to automate the process of monitoring the state of health for a cooling system using an incremental drift-aware learning model and Streaming Data Framework for MATLAB Production Server™. This example provides a dashboard app that you can use to monitor the state of the cooling system.

Functionality being removed or changed

gencfeatures and genrfeatures Functions: Categorical features with missing values are omitted when TargetLearner is "linear" or "gaussian-svm"
Behavior change

When the `TargetLearner` is `"linear"` or `"gaussian-svm"`, the `gencfeatures` and `genrfeatures` functions always exclude original features that are categorical and include missing values, even when `IncludeInputVariables` is specified as `"include"`. That is, the features are not included in the table of generated features `NewTbl`. Additionally, the `gencfeatures` and `genrfeatures` functions do not generate features from these categorical features with missing values.

To include the original categorical features with missing values in `NewTbl`, you can first remove the observations with missing values from `Tbl` by using the `rmmissing` function.

Statistics

Multinomial Regression: Create a MultinomialRegression object to perform multinomial regression

Create a `MultinomialRegression` object for multinomial regression using the `fitmnr` function. After creating the object, you can predict response values, calculate confidence intervals for the estimated coefficients, and create plots.

- The `coefCI` function calculates confidence intervals for the coefficient estimates.
- The `coefTest` function performs a test to determine if coefficient estimates are statistically different from zero.
- The `feval` function predicts response categories using one input for each predictor variable.
- The `partialDependence` function calculates the partial dependence between the predictor variables and the predicted probabilities for the response.
- The `plotPartialDependence` function calculates and generates a plot of the partial dependence between the predictor variables and the predicted probabilities for the response.
- The `predict` function predicts response categories or category probabilities using one input for all predictor variables.
- The `random` function generates random response category labels.
- The `testDeviance` function performs a test to determine whether the fitted model is statistically different from a model with only intercept coefficients.
- The `plotSlice` function plots the probability for a response category as a function of a predictor variable.
- The `plotResiduals` function plots the residuals for the fitted model.

Multinomial Regression: Specify weights, tolerance, and maximum number of iterations for mnrfit

- Use the `Tolerance` name-value argument with `mnrfit` to specify how close the estimated parameters must be between iterations for the model to converge.
- Use the `IterationLimit` name-value argument to set the maximum number of iterations used by `mnrfit` to fit the model.
- Use the `Weights` name-value argument to set observation weights for the training data.

Euclidean Distance Calculations: Accelerated computations in pdist, pdist2, tsne, createns, ExhaustiveSearcher, and knnsearch

The `pdist`, `pdist2`, `tsne`, `createns`, and `knnsearch` functions, and the `ExhaustiveSearcher` object and its `knnsearch` method, have accelerated Euclidean distance calculations by using the distance metrics `'fasteuclidean'`, `'fastseuclidean'`, or for `pdist` and `pdist2`, `'fastsquaredeuclidean'`. These calculations use a cache to store an intermediate matrix. Control the size of the cache using the `CacheSize` name-value argument in `pdist`, `pdist2`, `knnsearch`, and `tsne`; the `ExhaustiveSearcher` object and its associated `knnsearch` method set the required cache size automatically. You can accelerate Euclidean distance in `knnsearch` only for exhaustive search, not for *K*d-tree search. For an algorithm description, see “Algorithms”.

This code for `pdist2` is about 10x faster than in the previous release.

```
rng default % For reproducibility
N = 10000;
X = randn(N,1000);
Y = randn(N,1000);
tic
D = pdist2(X,Y);
standard = toc
% The next code runs in R2023a or later
tic
D2 = pdist2(X,Y,"fasteuclidean");
accelerated = toc
```

The approximate execution times are:

- R2022b: 14.5s
- R2023a: 1.3s

These codes were timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system.

Example of linear model fitting on a GPU

A new example, “Accelerate Linear Model Fitting on GPU”, shows how to accelerate linear regression model fitting and analysis using a GPU. The workflow compares the execution time needed to fit a linear regression model to a large data set on a CPU with the execution time needed to fit the same model on a GPU. To determine the statistical significance of a model term, the example also uses the GPU to perform an ANOVA.

Visualization

Functionality being removed or changed

gplotmatrix uses the MATLAB default color scheme

Behavior change

Starting in R2023a, the `gplotmatrix` function uses the MATLAB default color scheme to determine the marker colors, following the order specified in the `ColorOrder` property of the axes. In previous releases, the `gplotmatrix` function used the default color scheme `'bgrcmyk'`.

To use the previous behavior, enter:

```
gplotmatrix(X,[],group,'bgrcmyk')
```

If you use the new default color scheme, and the number of unique groups exceeds the number of default colors (7), then `gplotmatrix` cycles through the default values, as needed. To use different colors for different groups, you can specify the marker colors using `hsv(numGroups)`, where `numGroups` is the number of unique group combinations in the grouping variables. For an example, see “Change Marker Colors”.

R2022b

Deployment

Nearest-Neighbor Prediction Block: Simulate a model and generate code in Simulink

You can now integrate the `predict` function of a nearest-neighbor classification object into Simulink using the new prediction block ClassificationKNN Predict instead of a MATLAB function block. You can import a trained model and configure data types using the Block Parameters dialog box.

The ClassificationKNN Predict block returns classified labels for new data using an imported `ClassificationKNN` object. The block can also return the predicted scores and expected classification scores.

The block supports C/C++ code generation and fixed-point conversion. You can generate C and C++ code using Simulink Coder, and you can design and simulate fixed-point systems using Fixed-Point Designer. In addition to fixed-point data types, the block supports all floating-point data types—double-precision, single-precision, and half-precision. Using half-precision data types requires Fixed-Point Designer or MATLAB Coder.

Generate C/C++ code for anomaly detection with a trained isolation forest model (MATLAB Coder)

You can generate C/C++ code for the `isanomaly` function, which detects anomalies in data using a trained isolation forest model (`IsolationForest`). For more information, see Code Generation of `isanomaly` and the example Code Generation for Anomaly Detection.

Example on deployment to FPGA/ASIC platform

A new example, Deploy Neural Network Regression Model to FPGA/ASIC Platform, shows how to train a neural network regression model, use the trained regression model in a Simulink model that estimates the state of charge (SoC) of a battery, and generate HDL code from the Simulink model for deployment to an FPGA/ASIC platform.

Example of model compression workflow for deployment to memory-limited hardware

A new example, Compress Machine Learning Model for Memory-Limited Hardware, shows how to reduce the size of a machine learning model for deployment to memory-limited hardware. The model compression workflow includes feature selection, constrained Bayesian optimization, and parameter quantization. To demonstrate the model compression workflow, this example builds models for the acoustic scene classification (ASC) task, which classifies environments from the sounds they produce.

Apps

Reduce Dimensionality Live Editor Task: Interactively reduce data dimensionality using Principal Component Analysis and generate code

Use the **Reduce Dimensionality** Live Editor task to perform Principal Component Analysis (PCA) interactively in a live script. You can specify the percentage of the variance to explain, or select the number of principal components to retain. The task can create a scree plot (percentage of the variance explained), a 2-D scatter plot, and a 2-D biplot of the principal components. The task automatically generates code that becomes part of your live script.

To use the **Reduce Dimensionality** task in the Live Editor, select **Task > Reduce Dimensionality** on the **Live Editor** tab. Alternatively, in a code block in a live script, begin typing the task name or a related name such as "PCA" or "principal." Then, select the task from the suggested command completions. For more information about Live Editor tasks, see Add Interactive Tasks to a Live Script.

Machine Learning Apps: Interpret trained models using partial dependence plots

Partial dependence plots (PDPs) allow you to visualize the marginal effect of each predictor on the predicted response of a trained model. After you train a model in Classification Learner or Regression Learner, you can view a partial dependence plot for the model. On the **Classification Learner** or **Regression Learner** tab, in the **Plot and Interpret** section, click the arrow to open the gallery. In the **Interpretation Results** section, click **Partial Dependence**. When computing partial dependence values, the app uses the final model, trained on the full data set (including training and validation data, but excluding test data).

For more information, see Interpret Model Using Partial Dependence Plots (for classification) and Interpret Model Using Partial Dependence Plots (for regression). For examples, see Interpret Classifiers Trained in Classification Learner App and Interpret Regression Models Trained in Regression Learner App.

Machine Learning Apps: Compare model information and results in a table view

In the Classification Learner and Regression Learner apps, you can use a table of results to compare models. On the **Classification Learner** or **Regression Learner** tab, in the **Models** section, click **Results Table**. In the **Results Table** tab, you can sort models by their training and test results, as well as by their options (such as model type, selected features, PCA, and so on). For more information, see Compare Model Information and Results in Table View (for classification) and Compare Model Information and Results in Table View (for regression).

Machine Learning Apps: Select individual neural network layers for optimization

Before training an optimizable neural network model in Classification Learner or Regression Learner, you can specify the fully connected layers whose sizes you want to optimize, even when you specify a fixed number of layers. Similarly, you can specify fixed values for the layer sizes and optimize the number of layers. For more information, see Select Hyperparameters to Optimize (for classification) and Select Hyperparameters to Optimize (for regression).

Classification Learner App: Create receiver operating characteristic (ROC) curve

After training a binary or multiclass model in Classification Learner, you can create ROC curves to evaluate the performance of the model.

- Performance on validation data — In the **Plots** section, click the arrow to open the gallery, and then click **ROC Curve (Validation)** in the **Validation Results** group.
- Performance on test data — After computing the test set metrics, in the **Plots** section, click the arrow to open the gallery, and then click **ROC Curve (Test)** in the **Test Results** group.

You can now select multiple classes at the same time and display the corresponding ROC curves on a single plot. The legend in the plot displays the class name and area under the curve (AUC) value for each curve. For more information, see Check ROC Curve.

Compatibility Considerations

Starting in R2022b, the app creates a ROC curve by using the `rocmetrics` function. In previous releases, the app created a ROC curve by using the `perfcurve` function, displayed the area under the curve as a shaded area, and displayed the AUC value as text under the curve.

Machine Learning

Incremental Learning: Specify linear or kernel binary learners for an incremental multiclass ECOC model

The `incrementalClassificationECOC` object, which is an incremental multiclass error-correcting output codes (ECOC) classification model, now supports the `Learners` name-value argument so that you can customize binary learners when you create the object. In previous releases, the object used the default `incrementalClassificationLinear` objects as binary learners.

You can specify `Learners` as either a set of linear binary learners or a set of kernel binary learners by using one of these options:

- `"linear"` or `"kernel"` — Specify the `Learners` value as a string scalar or character vector to use the default linear learners or default kernel learners (default `incrementalClassificationLinear` or `incrementalClassificationKernel` objects, respectively).
- `incrementalClassificationLinear` or `incrementalClassificationKernel` object — Configure binary learner properties (both model-specific properties and incremental learning properties) when you create an incremental learning object, and pass the object to `incrementalClassificationECOC` as the `Learners` value.
- Template object returned by the `templateLinear`, `templateSVM`, or `templateKernel` function — Configure model-specific properties when you create a template object, and pass the object to `incrementalClassificationECOC` as the `Learners` value. Use this approach to specify model properties with a template object and to use the default incremental learning options.
- Cell array of supported incremental learning objects and template objects — Use this approach to customize each learner individually.

Statistics and Machine Learning Toolbox provides several ways of creating `incrementalClassificationLinear` and `incrementalClassificationKernel` objects. You can create an object by calling the object directly or by converting a traditionally trained model. In addition, starting in 2022b, you can create the object by converting a template object.

- Use the `incrementalLearner` function to create an `incrementalClassificationLinear` object by converting a linear template object returned by `templateLinear` or `templateSVM`.
- Use the `incrementalLearner` function to create an `incrementalClassificationKernel` object by converting a kernel template object returned by `templateKernel`.

Incremental Learning: Train an incremental drift-aware classification or regression learner

The `incrementalDriftAwareLearner` function incorporates an incremental classification or regression model and an incremental concept drift detector to provide a self-adjusting incremental machine learning model. Create an incremental drift-aware classification or regression learner as follows:

1. Initiate an incremental classification or regression learner using any incremental learner.
2. Initiate an incremental drift-aware learner by passing the incremental learning model to `incrementalDriftAwareLearner` as an input.

For example:

```
BaseLearner = incrementalClassificationLinear();
IDAMdl = incrementalDriftAwareLearner(BaseLearner);
```

In this case, `incrementalDriftAwareLearner` uses the default concept drift detector. Alternatively, you can initiate the incremental concept drift detector and pass it to `incrementalDriftAwareLearner`.

For example:

```
DDM = incrementalConceptDriftDetector('ddm');
IDAMdl = incrementalDriftAwareLearner(BaseLearner,DriftDetector=DDM);
```

After creating the incremental drift-aware model, you can train it, assess its performance, and predict responses as the model accesses data, either per individual observation or specified batch size, in real time.

- The `fit` function fits the model by updating the base learner and monitoring for drift given an incoming batch of data. When you call `fit`, the software performs the following procedure:
  - Trains the model up to `NumTrainingObservations` observations.
  - After training, the software starts tracking the model loss to see if any concept drift has occurred and updates drift status accordingly.
  - When the drift status is `Warning`, the software trains a temporary model to replace the`BaseLearner` in preparation for an imminent drift.
  - When the drift status is `Drift`, temporary model replaces the `BaseLearner`.
  - When the drift status is `Stable`, the software discards the temporary model.
- The `updateMetrics` function evaluates the performance of the model as it processes incoming observations. The function writes specified metrics, measured cumulatively and within a specified window of processed observations, to the `Metrics` model property.
- The `updateMetricsAndFit` function first evaluates the performance of the model by calling `updateMetrics` on incoming data, and then fits the model to that data by calling `fit`.
- The `predict` function predicts responses given incoming predictor data.
- The `loss` function returns the regression or classification loss given incoming predictor and response data. Unlike `updateMetrics`, the `loss` function does not write the computed loss to the model.
- The `perObservationLoss` function returns the per observation regression or classification error for incremental models.
- The `reset` function resets all learned parameters and estimated hyperparameters of the base learner and all internal states of the drift detector.

Fairness in Machine Learning: Evaluate the fairness of a data set or binary classification model using bias and group metrics

The `fairnessMetrics` object computes fairness metrics (bias metrics and group metrics) for a data set or binary classification model with respect to sensitive attributes. The data-level evaluation examines binary, true labels of the data. The model-level evaluation examines the predicted labels returned by the binary classification model, using both true labels and predicted labels. You can use the metrics to determine if your data or model contains bias toward a group within each sensitive attribute.

After creating a `fairnessMetrics` object, you can use the `report` function to generate a fairness metrics report, and the `plot` function to create a bar graph of the metrics.

Fairness in Machine Learning: Reweight observations or remove the disparate impact of a sensitive attribute

To mitigate the effects of a sensitive attribute on binary classification, you can use the `fairnessWeights` function or the `disparateImpactRemover` function to modify your data set before training a binary classifier.

- Use `fairnessWeights` to reweight observations. For every combination of a group in the sensitive attribute and a class label in the response variable, the software computes a weight value. The function then assigns each observation its corresponding weight. The returned weights introduce fairness across the sensitive attribute groups. Pass the weights to an appropriate training function, such as `fitcsvm`, using the `Weights` name-value argument.
- Use `disparateImpactRemover` to remove the disparate impact of the sensitive attribute. The software uses the sensitive attribute to transform the continuous predictors in the data set. The function returns the transformed data set and a `disparateImpactRemover` object that contains the transformation. Pass the transformed data set to an appropriate training function, such as `fitcsvm`, and pass the object to the `transform` object function to apply the transformation to a new data set, such as a test data set.

For examples, see Introduction to Fairness in Binary Classification.

Anomaly Detection: Detect novelties and outliers using one-class support vector machine (SVM)

Use the `ocsvm` function, `OneClassSVM` object, and `isanomaly` object function for anomaly detection. This feature uses Gaussian kernel one-class SVM, which tries to separate data from the origin in the transformed high-dimensional predictor space. The software finds the decision boundary based on the primal form of SVM with the Gaussian kernel approximation method.

- Outlier detection (detecting anomalies in training data) — Detect anomalies in training data by passing the data to the `ocsvm` function. The function returns a `OneClassSVM` object, a logical array indicating anomalies, and a numeric array of anomaly scores. You can specify the fraction of anomalies in the data by using the `ContaminationFraction` name-value argument.
- Novelty detection (detecting anomalies in new data with uncontaminated training data) — Create a `OneClassSVM` object by passing uncontaminated training data (data with no outliers) to the `ocsvm` function. Detect anomalies in new data by passing the object and the new data to the object function `isanomaly`. The `isanomaly` function returns a logical array indicating anomalies and a numeric array of anomaly scores. The function identifies observations with scores above the score threshold as anomalies. You can specify the score threshold by using the `ScoreThreshold` name-value argument.

A negative score value with large magnitude indicates a normal observation, and a large positive value indicates an anomaly.

Anomaly Detection: Detect novelties and outliers using the local outlier factor algorithm

Use the `lof` function, `LocalOutlierFactor` object, and `isanomaly` object function for anomaly detection. This feature uses the local outlier factor algorithm, which detects anomalies based on the relative density of an observation with respect to the surrounding neighborhood.

- Outlier detection (detecting anomalies in training data) — Detect anomalies in training data by passing the data to the `lof` function. The function returns a `LocalOutlierFactor` object, a logical array indicating anomalies, and a numeric array of anomaly scores (local outlier factor values). You can specify the fraction of anomalies in the data by using the `ContaminationFraction` name-value argument.
- Novelty detection (detecting anomalies in new data with uncontaminated training data) — Create a `LocalOutlierFactor` object by passing uncontaminated training data (data with no outliers) to the `lof` function. Detect anomalies in new data by passing the object and the new data to the object function `isanomaly`. The `isanomaly` function returns a logical array indicating anomalies and a numeric array of anomaly scores (local outlier factor values). The function identifies observations with scores above the score threshold as anomalies. You can specify the score threshold by using the `ScoreThreshold` name-value argument.

In this algorithm, anomaly scores represent local outlier factor values. A score value less than 1 or close to 1 indicates a normal observation, and a value greater than 1 can indicate an anomaly.

Time Series: Partition time series data for cross-validation

A `tspartition` object partitions a set of regularly sampled, time series data based on the specified size of the data set. Use this object to define training and test sets for validating a time series regression model with expanding window cross-validation, sliding window cross-validation, or holdout validation.

- Expanding window cross-validation — Specify the number of test sets `t`. The `tspartition` function splits the data set into `t` windows with expanding training sets and fixed-size test sets.
- Sliding window cross-validation — Specify the number of test sets `t`. The `tspartition` function splits the data set into `t` windows with fixed-size training and test sets.
- Holdout validation — Specify `p`, the fraction or number of observations in the test set. The `tspartition` function divides the observations into a training set and a test set, where the test set contains the latest observations.

After you create a `tspartition` object, you can use the `training` object function to extract the training indices and the `test` object function to extract the test indices.

For an example that uses `tspartition` for time series forecasting, see Time Series Forecasting Using Ensemble of Boosted Regression Trees.

GPU Support: Functions that fit decision tree models and ensembles now support categorical predictors (requires Parallel Computing Toolbox)

The functions `fitctree`, `fitrtree`, `fitcensemble`, `fitrensemble`, and `fitensemble` support categorical predictors when you execute the functions on a GPU. The `fitcecoc` function also

supports categorical predictors for classification tree learners when you execute the function on a GPU.

GPU Support: Updated support for fitcsvm (requires Parallel Computing Toolbox)

- The `fitcsvm` function now supports hyperparameter optimization when you execute the function on a GPU.
- When `fitcsvm` fits the SVM model on a GPU, some of the properties of the `ClassificationSVM` model are stored in GPU memory.
- Use `gather` to create a `ClassificationSVM` or `CompactClassificationSVM` object with properties stored in the local workspace from an equivalent object with properties stored in GPU memory.
- The object functions of the `ClassificationSVM` and `CompactClassificationSVM` models execute on a GPU when the model is fitted with GPU arrays.

Compatibility Considerations

GPU array support for `fitcsvm` now matches the behavior of other Statistics and Machine Learning Toolbox functions that fit classification or regression models, such as `fitlm`, `fitctree`, and `fitcknn`. For the full list of Statistics and Machine Learning Toolbox functions that accept GPU arrays, see Function List (GPU Arrays). For more information on how to update your code to avoid compatibility issues, see “ClassificationSVM and CompactClassificationSVM object functions now execute on a GPU when the model is fitted with GPU arrays” on page 2-9.

Battery state of charge (SOC) example using machine learning

A new example, Predict Battery State of Charge Using Machine Learning, shows how to train a Gaussian process regression model to predict the state of charge of a battery.

fitcnet and fitrnet Functions: Specify the initial step size used in model training

The functions `fitcnet` and `fitrnet` now allow you to specify an initial step size. Set the `InitialStepSize` name-value argument to `"auto"` or a positive scalar. By default, the software does not use the initial step size to determine the initial Hessian approximation used to train the neural network model. However, if you specify an initial step size, then the software uses the value to compute the initial inverse-Hessian approximation. For more information, see `InitialStepSize` (for classification) and `InitialStepSize` (for regression).

Functionality being removed or changed

ClassificationSVM and CompactClassificationSVM object functions now execute on a GPU when the model is fitted with GPU arrays

Behavior change

The object functions of the `ClassificationSVM` and `CompactClassificationSVM` models now execute on a GPU when the model is fitted with GPU arrays or when the predictor data you pass to

the object function is a GPU array. Previously, the object functions executed on a GPU only if the predictor data, which you passed to the object function, was a GPU array.

When the object function executes on a GPU, it returns a variable stored in GPU memory or an object with some properties stored in GPU memory. You might want the variables and properties to be stored in the local workspace so you can pass them to functions that do not support GPU arrays. To update your code, use the `gather` function to gather the variables and objects to the local workspace.

A cross-validated Gaussian process regression model is a RegressionPartitionedGP object
Behavior change

Starting in R2022b, a cross-validated Gaussian process regression (GPR) model is a `RegressionPartitionedGP` object. In previous releases, a cross-validated GPR model was a `RegressionPartitionedModel` object.

You can create a `RegressionPartitionedGP` object in two ways:

- Create a cross-validated model from a GPR model object `RegressionGP` by using the `crossval` object function.
- Create a cross-validated model by using the `fitrgp` function and specifying one of the name-value arguments `CrossVal`, `CVPartition`, `Holdout`, `KFold`, or `Leaveout`.

Regardless of whether you train a full or cross-validated GPR model first, you cannot specify an `ActiveSet` value in the call to `fitrgp`.

Statistics

Analysis of Variance: Create an anova object to perform one-, two-, or n-way analysis of variance

Create an `anova` object for one-, two-, or *n*-way analysis of variance (ANOVA) using the `anova` function. After creating the object, you can perform a multiple comparison of the means, compute the ANOVA statistics, and create plots.

- The `boxchart` function creates a box plot of the response for one or two factors.
- The `groupmeans` function computes the mean response and confidence intervals for factor values.
- The `multcompare` function performs a multiple comparison of means for factor values.
- The `plotComparisons` function creates a multiple comparisons plot.
- The `stats` function computes the component ANOVA table and related tables.
- The `varianceComponent` function computes the variance component estimate for random model components.

Probability Distributions: Plot a univariate probability distribution using the plot function

Given a univariate probability distribution object, the `plot` function now plots a probability density function (pdf), a cumulative distribution function (cdf), or a probability plot for the distribution. You can specify the plot type using the `PlotType` name-value argument. When the distribution is fit to data, plot also displays one of the following:

- A histogram of the data if `PlotType` is set to `"pdf"`.
- An empirical cdf of the data if `PlotType` is set to `"cdf"`.

Cox Proportional Hazards Model: Reduce memory usage, select extrapolation method

You can reduce the memory usage of a `CoxModel` object by using `discardResiduals`. For an example, see the function reference page or Cox Proportional Hazards Model Object.

To compute or plot the survival to times outside the training interval when using the `survival` or `plotSurvival` functions, you can choose the extrapolation method in the `ExtrapolationMethod` name-value argument. For example, to choose linear extrapolation to a set of times `T` for `survival`, set

```
s = survival(coxMdl,Time=T,ExtrapolationMethod="linear");
```

For details, see the `survival` or `plotSurvival` reference pages.

GPU Support: Improved performance for pca

The performance of the `pca` function on a GPU is now improved when you use the SVD algorithm. For more information on choosing between the EIG and SVD algorithms, see GPU Arrays.

Descriptive Statistics: Run functions in a thread-based environment

You can now run the following functions in the background using MATLAB `backgroundPool`:

- `cholcov`
- `corr`
- `corrcov`
- `crosstab`
- `grpstats` — Plotting group means is not supported.
- `nearcorr`
- `partialcorr`
- `partialcorri`
- `robustcov`
- `tabulate`
- `tiedrank`

For more information, see Run MATLAB Functions in Thread-Based Environment.

Visualization

Statistical Visualization Functions: Generate MATLAB code to recreate a figure

All statistical visualization functions in Statistics and Machine Learning Toolbox now support enhanced MATLAB code generation that better reflects modifications you make using the plot tools. The following statistical visualization functions are enhanced in this release.

- `andrewsplot` — Andrews plot
- `binScatterPlot` — Scatter plot of bins for tall arrays
- `biplot` — Biplot
- `boxplot` — Box plot
- `gline` — Interactively add line to plot
- `gname` — Add case names to plot
- `glyphplot` — Glyph plot
- `hist3` — Bivariate histogram plot
- `lsline` — Add least-squares line to scatter plot
- `parallelcoords` — Parallel coordinates plot
- `refcurve` — Add reference curve to plot
- `refline` — Add reference line to plot
- `scatterhist` — Scatter plot with marginal histograms

When you select **File > Generate Code** for a figure created by one of the functions, the generated code is displayed in the MATLAB Editor. You can use the code to reproduce the figure. Generated files do not store the data necessary to recreate a figure, so you must supply the data arguments.

tsne Function: Increased performance using Barnes-Hut algorithm

The `tsne` function has improved performance when using the default `'barneshut'` algorithm.

This code for `tsne` is about 3x faster than in the previous release.

```
function timingTest
load HumanActivity.mat
tic
tD = tsne(feat);
toc
```

The approximate execution times are:

- R2022a: 156s
- R2022b: 55.7s

These codes were timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the function `timingTest`.

R2022a

Deployment

Gaussian Process Prediction Block: Simulate a model and generate code in Simulink

You can now integrate the `predict` function of a Gaussian process (GP) regression object into Simulink using the new prediction block RegressionGP Predict instead of a MATLAB function block. You can import a trained model and configure data types using the Block Parameters dialog box.

The RegressionGP Predict block predicts responses for new data using an imported GP regression object (`RegressionGP` or `CompactRegressionGP`). The block can also return the standard deviations and prediction intervals of the responses.

The block supports C/C++ code generation and fixed-point conversion. You can generate C and C++ code using Simulink Coder, and you can design and simulate fixed-point systems using Fixed-Point Designer. In addition to fixed-point data types, the block supports all floating-point data types—double-precision, single-precision, and half-precision. Using half-precision data types requires Fixed-Point Designer or MATLAB Coder.

Generate C/C++ code for prediction using neural network classification and regression models (requires MATLAB Coder)

- You can now generate C/C++ code for the `predict` function, which classifies observations by using a trained neural network classification model (`ClassificationNeuralNetwork` or `CompactClassificationNeuralNetwork`).
- Also, you can generate C/C++ code for the `predict` function, which predicts responses by using a trained neural network regression model (`RegressionNeuralNetwork` or `CompactRegressionNeuralNetwork`).

ecdf Function: Empirical cumulative distribution function enhanced to generate C/C++ code for left-censored, double-censored, and interval-censored data (requires MATLAB Coder)

The `ecdf` function now supports C/C++ code generation for left-censored, double-censored, and interval-censored data, in addition to the previously supported data types (fully observed data and right-censored data).

Apps

Machine Learning Apps: Save and open app sessions

In Classification Learner and Regression Learner, you can save the current app session and open a previously saved app session. To save the current app session, click **Save** in the **File** section of the **Classification Learner** or **Regression Learner** tab. To open a saved app session, click **Open** in the **File** section.

You can also open a saved app session from the command line. For more information, see **Classification Learner** and **Regression Learner**.

Machine Learning Apps: Reserve percentage of imported data for testing

When you import data into Classification Learner or Regression Learner, you can specify to reserve a percentage of the data for testing. If you prefer, you can still choose to import a separate test data set after starting an app session.

Specify the fraction of data to use for testing at the command line or in the app.

- In the command line call to `classificationLearner` or `regressionLearner`, specify the `TestDataFraction` name-value argument. The app populates the New Session from Arguments dialog box with the specified value.

  For example, this code tells Classification Learner to reserve 20% of the imported data for testing.

  ```
  load fisheriris
  classificationLearner(meas,species,"CrossVal","on", ...
    "TestDataFraction",0.20)
  ```

- Alternatively, open the Classification Learner or Regression Learner app, and then create a new session by clicking **New Session** in the **File** section of the **Classification Learner** or **Regression Learner** tab. In the dialog box, click the check box in the **Test** section to set aside a test data set. Specify the percentage of the imported data to use as a test set.

  For examples, see Train Classifier Using Hyperparameter Optimization in Classification Learner App and Train Regression Model Using Hyperparameter Optimization in Regression Learner App.

Machine Learning Apps: Use feature ranking algorithms to select predictors

Before training models in Classification Learner or Regression Learner, you can determine which important predictors to include by using different feature ranking algorithms. After you select a feature ranking algorithm, the app displays a plot of the sorted feature importance scores, where larger scores indicate greater feature importance. The app also displays the ranked features and their scores in a table.

To use feature ranking algorithms in Classification Learner or Regression Learner, click **Feature Selection** in the **Options** section of the **Classification Learner** or **Regression Learner** tab. The app opens a **Default Feature Selection** tab, where you can choose the **MRMR** (minimum redundancy maximum relevance), **Chi2**, **ReliefF**, **ANOVA** (one-way analysis of variance), or **Kruskal Wallis** algorithm for classification, or the **MRMR**, **F Test**, or **RReliefF** algorithm for regression.

For more information, see Select Features to Include (for classification) and Select Features to Include (for regression). For examples, see Train Decision Trees Using Classification Learner App and Train Regression Trees Using Regression Learner App.

Machine Learning Apps: Create several draft models

You can now create a list of draft models in the Classification Learner and Regression Learner apps. Create a draft model using the model gallery or the **Duplicate** button.

- To create a new draft model, use the model gallery. On the **Classification Learner** or **Regression Learner** tab, in the **Models** section, click the arrow to open the gallery. Click the type of model you want to create. You can specify the draft model properties using the model summary.
- To duplicate an existing model, click the model in the **Models** pane. On the **Classification Learner** or **Regression Learner** tab, in the **Models** section, click **Duplicate**. Alternatively, right-click the model in the **Models** pane and select **Duplicate**.

Compatibility Considerations

In previous releases, clicking a model in the model gallery updated the current draft model.

Machine Learning Apps: Use model summary to change model options and view model results

In Classification Learner and Regression Learner, each model has a summary tab that describes the following model options: model hyperparameters, feature selection, PCA options, misclassification costs, and optimizer options. For a draft model, you can specify some of these options before training the model. For a trained model, you can view its specified options, as well as its training and test results.

To examine the summary of a model, click the model in the **Models** pane. On the **Classification Learner** or **Regression Learner** tab, in the **Models** section, click **Summary**. Alternatively, right-click the model in the **Models** pane and select **Summary**.

If you want to change the feature selection, PCA options, misclassification costs, or optimizer options for all new and existing draft models, use the buttons in the **Options** section of the **Classification Learner** or **Regression Learner** tab.

Compatibility Considerations

The model summary replaces the **Advanced** toolstrip button and the **Current Model Summary** pane.

Machine Learning Apps: Train all draft models

In addition to creating several draft models in Classification Learner and Regression Learner, you can also choose to train those models in a batch. On the **Classification Learner** or **Regression Learner** tab, in the **Train** section, click **Train All** and select **Train All**. To train only the currently select draft model, click **Train All** and select **Train Selected**.

For more information, see Train Classification Models in Classification Learner App and Train Regression Models in Regression Learner App.

Regression Learner: Train Gaussian kernel regression models for nonlinear regression of data with many observations

In Regression Learner, you can use kernel regression models to perform nonlinear regression of data with many observations. The Gaussian kernel regression models map predictors in a low-dimensional space into a high-dimensional space, and then fit a linear model to the transformed predictors in the high-dimensional space. Choose between fitting a support vector machine (SVM) model and fitting a least-squares model in the expanded space. For large in-memory data, kernel regression models tend to train and predict faster than SVM regression models with Gaussian kernels.

To create a kernel regression model in Regression Learner, click the arrow in the **Models** section of the **Regression Learner** tab to open the gallery. In the **Kernel Approximation Regression Models** group, click one of the available models. Then, in the **Train** section, click **Train All** and select **Train All** or **Train Selected**.

For more information, see Kernel Approximation Models and Train Kernel Approximation Model Using Regression Learner App.

Classification Learner App: Specify the number of predictors to sample for ensembles

For ensemble classifiers in the Classification Learner app, you can specify the number of predictors to select at random for each split in the tree learners. In the **Summary** tab, under **Model Hyperparameters**, specify the **Number of predictors to sample** value.

- When you set this option to `Select All`, the software uses all available predictors.
- When you set this option to `Set Limit`, you can specify a value by clicking the buttons or entering a positive integer value in the box.

For more information, see Ensemble Model Hyperparameter Options.

Machine Learning Apps: Train draft models in parallel (requires Parallel Computing Toolbox) and interact with app during model training

In Classification Learner and Regression Learner, you can train multiple draft models in parallel, if you have a Parallel Computing Toolbox license, by using the **Use Parallel** button. You can continue to interact with the app while models train in parallel.

If you do not have a Parallel Computing Toolbox license, you can keep the app responsive during model training by ensuring the **Use Background Training** check box is selected. You can continue to interact with the app while models train in the background.

The **Use Parallel** button and the **Use Background Training** check box are available when you train optimizable models. For more information on these options, see Parallel Classifier Training and Parallel Regression Model Training.

Compatibility Considerations

Starting in R2022a, the **Use Parallel** button is on by default. In the previous release, the app toggled the button off by default.

Classification Learner: Anomaly detection example for industrial machinery and manufacturing processes

A new example, Build Condition Model for Industrial Machinery and Manufacturing Processes, shows how to build a condition model that detects anomalies for sensor data collected from an industrial manufacturing machine. The example covers new app features introduced in R2022a, including saving and opening app sessions, reserving a percentage of the imported data for testing, and using feature ranking algorithms to select predictors.

Machine Learning

Incremental Drift Detection: Perform concept drift detection on incoming observations from streaming data

Statistics and Machine Learning Toolbox lets you perform incremental drift detection for continuous or binary data.

Use the `incrementalConceptDriftDetector` function to initiate the drift detector and select one of the three methods: Drift detection method, Hoeffding's bounds drift detection method with moving average test (HDDM-A or HDDMA), or Hoeffding's bounds drift detection method with using exponentially moving average (EWMA) test (HDDM-W or HDDMW). Depending on the method you choose, `incrementalConceptDriftDetector` returns a `DriftDetectionMethod` object or a `HoeffdingDriftDetectionMethod` object.

After creating the object, use it with the `detectdrift` function to monitor the drift status on incoming data. Once a drift is detected, reset the drift detector using the `reset` function.

Incremental Learning: Reset incremental models and compute the per observation loss

The `reset` function resets all learned parameters and any estimated hyperparameters of an incremental model. For classification models, the function also resets the prior class probabilities if they are empirical.

The `perObservationLoss` function computes the per observation regression or classification error for incremental models.

See `reset` and `perObservationLoss` for these regression models: `incrementalRegressionLinear` and `incrementalRegressionKernel`.

See `reset` and `perObservationLoss` for these classification models: `incrementalClassificationLinear`, `incrementalClassificationNaiveBayes`, `incrementalClassificationKernel`, and `incrementalClassificationECOC`.

Incremental Learning: Train kernel regression or binary classification models on incoming observations from streaming data, and assess performance in real time

You can create an incremental learner for kernel regression (`incrementalRegressionKernel`) or binary classification (`incrementalClassificationKernel`) in two ways:

- Convert a trained model to an incremental learner by passing it to the `incrementalLearner` function. Convertible models include:
  - Kernel regression model (`RegressionKernel`), returned by `fitrkernel`
  - Kernel classification model (`ClassificationKernel`), returned by `fitckernel`

  Properties of the converted model reflect the knowledge gained from processing the data.
- Prepare the model for incremental learning by calling `incrementalRegressionKernel` or `incrementalClassificationKernel` directly, before you fit the model to data.

Regardless of how you create the model, you can train it, assess its performance, and predict responses as the model accesses data, either per individual observation or per specified batch size, in real time.

- The `fit` function fits the model to incoming data by updating the kernel model parameters.
- The `updateMetrics` function evaluates the performance of the model as it processes incoming observations. The function writes specified metrics, measured cumulatively and within a specified window of processed observations, to the `Metrics` model property.
- The `updateMetricsAndFit` function first evaluates the performance of the model by calling `updateMetrics` on incoming data, and then fits the model to that data by calling `fit`.
- The `predict` function predicts responses given incoming predictor data.
- The `loss` function returns the classification or regression loss given incoming predictor and response data. Unlike `updateMetrics`, the `loss` function does not write the computed loss to the model.

Incremental Learning: Train a multiclass ECOC model on incoming observations from streaming data, and assess performance in real time

You can create an error-correcting output codes (ECOC) incremental learner (`incrementalClassificationECOC`) for multiclass classification in two ways:

- Convert a trained ECOC model (`ClassificationECOC` or `CompactClassificationECOC`), trained by `fitcecoc`, to an incremental learner by passing it to the `incrementalLearner` function. You can convert an ECOC model with support vector machine (SVM) binary learners or linear classification model binary learners. Properties of the converted model reflect the knowledge gained from processing the data.
- Prepare the model for incremental learning by calling `incrementalClassificationECOC` directly, before you fit the model to data.

Regardless of how you create the model, you can train it, assess its performance, and predict responses as the model accesses data, either per individual observation or specified batch size, in real time.

- The `fit` function fits the model by updating the binary learners given an incoming batch of data.
- The `updateMetrics` function evaluates the performance of the model as it processes incoming observations. The function writes specified metrics, measured cumulatively and within a specified window of processed observations, to the `Metrics` model property.
- The `updateMetricsAndFit` function first evaluates the performance of the model by calling `updateMetrics` on incoming data, and then fits the model to that data by calling `fit`.
- The `predict` function predicts responses given incoming predictor data.
- The `loss` function returns the classification loss given incoming predictor and response data. Unlike `updateMetrics`, the `loss` function does not write the computed loss to the model.

Compute and plot classification performance metrics including receiver operating characteristic (ROC) curves

Use `rocmetrics` to evaluate the performance of binary or multiclass classification models with performance metrics. You can create a `rocmetrics` object by passing true labels, classification

scores, and class names. By default, `rocmetrics` computes the true positive rates (TPR), false positive rates (FPR), and area under the ROC curve for each class. Additionally, you can specify the following supported performance metrics by using the `AdditionalMetrics` name-value argument:

- Number of true positives (TP)
- Number of false negatives (FN)
- Number of false positives (FP)
- Number of true negatives (TN)
- Sum of TP and FP
- Rate of positive predictions (RPP)
- Rate of negative predictions (RNP)
- Accuracy
- False negative rate (FNR), or miss rate
- True negative rate (TNR), or specificity
- Positive predictive value (PPV), or precision
- Negative predictive value (NPR)
- Expected cost
- Custom metric, specified as a function handle

After creating a `rocmetrics` object, you can use the following object functions:

- `plot` — Plot ROC or other classifier performance curves. `plot` returns a `ROCCurve` graphics object for each curve. You can modify the properties of the objects to control the appearance of each curve. For details, see `ROCCurve Properties`.
- `average` — Compute performance metrics for an average ROC curve for multiclass problems.
- `addMetrics` — Compute additional classification performance metrics.

You can also compute the confidence intervals of performance curves by providing cross-validated inputs or by bootstrapping the input data. To bootstrap in parallel, specify the `BootstrapOptions` name-value argument in the call to `rocmetrics` and set the `UseParallel` field of the options structure to `true` using `statset`. For example, specify `BootstrapOptions=statset(UseParallel=true)`. This option requires Parallel Computing Toolbox.

fitcauto and fitrauto Functions: Include neural network learners in model selection

`fitcauto` and `fitrauto` now include neural network models in the list of available learners. To include neural network models as part of the model selection and hyperparameter tuning process, specify `Learners` as `"all"`, `"all-nonlinear"`, or a string containing `"net"` (for example, `["ensemble","net"]`).

Compatibility Considerations

Starting in R2022a, `fitcauto` and `fitrauto` include neural network models when you specify `"all"` or `"all-nonlinear"` for the `Learners` name-value argument. The functions also consider

neural network models when you specify `Learners` as `"auto"`, depending on the characteristics of your data set.

To omit neural network models from the model selection process, you can explicitly specify the models you want to include. For example, to use tree and ensemble models only, specify `"Learners",["tree","ensemble"]`.

gencfeatures and genrfeatures Functions: Automatically create new features before training a support vector machine (SVM) model

You can engineer new features before training an SVM model with a Gaussian kernel.

- To generate features before training a binary SVM classifier, use `gencfeatures`.
- To generate features before training a regression SVM model, use `genrfeatures`.

When using either function, specify `TargetLearner` as `"gaussian-svm"`. By default, the function uses neighborhood component analysis (NCA) as the method for selecting new features (`FeatureSelectionMethod="nca"`). Note that this feature selection method is supported only when the predictor features are all numeric. If `TargetLearner` is `"gaussian-svm"` and the table of original features includes categorical predictors, specify `FeatureSelectionMethod` as `"mrmr"`.

For examples, see Generate New Features to Train SVM Classifier and Generate New Features to Train SVM Regression Model.

Feature Selection: Rank features using fsrmrmr

`fsrmrmr` ranks features using the minimum redundancy maximum relevance (MRMR) algorithm for regression problems. The algorithm tries to find an optimal set of features that minimizes the redundancy of the feature set and maximizes the relevance of the feature set to the response variable. This algorithm applies to data sets that include both continuous and categorical features.

The function returns the indices of the features, ordered by feature importance, and the score for each feature. A large score value indicates that the corresponding feature is important. Also, a drop in the feature importance score represents the confidence of feature selection. For example, if the software is confident of selecting a feature x, then the score value of the next most important feature is much smaller than the score value of x. You can use this information to perform feature selection.

partialDependence and plotPartialDependence Functions: Compute and plot partial dependencies for custom models

The functions `partialDependence` and `plotPartialDependence` now support a custom model, specified as a function handle. The custom model function represented by the function handle must accept predictor data and return a column vector or matrix with one row for each observation. Use `partialDependence` and `plotPartialDependence` to analyze relationships between predictors and outputs of a custom model. The custom model can be a custom supervised learning model (regression or classification) or another type of model, such as a semi-supervised learning model or an anomaly detection model. For examples, see Specify Model Using Function Handle (`partialDependence`) and Specify Model Using Function Handle (`plotPartialDependence`).

loss Functions of Classification Model Objects: Compute observed misclassification cost

The `loss`, `resubLoss`, and `kfoldLoss` functions of classification model objects now support computation of the observed misclassification cost. Specify the `LossFun` name-value argument as `"classifcost"`. The functions return a weighted average misclassification cost of the input data, training data, and data for cross-validation, respectively. You can use the `"classifcost"` option to compare the results of cost-sensitive learning across classification models.

Compatibility Considerations

Starting in R2022a, the `Cost` property of all classification model objects stores the user-specified cost matrix, so that you can compute the observed misclassification cost using the specified value. The software stores prior probabilities (`Prior`) and observation weights (`W`) that do not reflect the penalties described in the cost matrix. For more details, see “Cost property of classification model objects stores the user-specified cost matrix” on page 3-14.

fit function of lime uses the NumImportantPredictors property

The object function `fit` of `lime` now uses the `NumImportantPredictors` property of the input `lime` object. If you specify the number of important predictors to use in a simple model when you create a `lime` object, you do not need to specify it again when you call the `fit` function.

Parallel Inference for Tree Ensembles: Accelerate prediction, loss calculation, and more (requires Parallel Computing Toolbox)

Tree ensembles can now perform inference calculations in parallel. Parallel computing can be faster than serial, especially for large datasets. To compute in parallel, set the `UseParallel` name-value argument to `true`. For example,

```
Yfit = predict(ens,X,UseParallel=true)
```

The following methods can compute in parallel:

Parallel Inference Methods

Method	Classes
`loss`	`CompactClassificationEnsemble`, `ClassificationEnsemble`, `ClassificationBaggedEnsemble`, `CompactRegressionEnsemble`, `RegressionEnsemble`, `RegressionBaggedEnsemble`
`predict`	`CompactClassificationEnsemble`, `ClassificationEnsemble`, `ClassificationBaggedEnsemble`, `CompactRegressionEnsemble`, `RegressionEnsemble`, `RegressionBaggedEnsemble`
`margin`	`CompactClassificationEnsemble`, `ClassificationEnsemble`, `ClassificationBaggedEnsemble`
`edge`	`CompactClassificationEnsemble`, `ClassificationEnsemble`, `ClassificationBaggedEnsemble`
`resubLoss`	`ClassificationEnsemble`, `ClassificationBaggedEnsemble`, `RegressionEnsemble`, `RegressionBaggedEnsemble`
`resubPredict`	`ClassificationEnsemble`, `ClassificationBaggedEnsemble`, `RegressionEnsemble`, `RegressionBaggedEnsemble`
`resubMargin`	`ClassificationEnsemble`, `ClassificationBaggedEnsemble`
`resubEdge`	`ClassificationEnsemble`, `ClassificationBaggedEnsemble`
`oobLoss`	`ClassificationBaggedEnsemble`, `RegressionBaggedEnsemble`
`oobPredict`	`ClassificationBaggedEnsemble`, `RegressionBaggedEnsemble`
`oobMargin`	`ClassificationBaggedEnsemble`
`oobEdge`	`ClassificationBaggedEnsemble`

For example, this code for `RegressionEnsemble.predict` uses parallel computing, and is about 2x faster than the serial computation in the previous release.

```
function timingTest4
rng(1,"twister")
Xtrain = rand(1e3,2);
Ytrain = rand(1e3,1);
mdl = fitrensemble(Xtrain,Ytrain);
Xtest = rand(1e6,2);
Ytest = rand(1e6,1);
tic
predict(mdl,Xrtest,'UseParallel',true);
toc
```

The approximate execution times are:

- R2022a: 4.2s
- R2021b: 8.2s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the function `timingTest4`.

GPU Support: fitrensemble and fitensemble now accept gpuArray inputs (requires Parallel Computing Toolbox)

- The functions `fitrensemble` and `fitensemble` are enhanced to accept `gpuArray` (Parallel Computing Toolbox) input arguments, so that the functions can execute on a GPU. The functions support GPU arrays with some limitations.
- The object functions of the models `RegressionEnsemble`, `CompactRegressionEnsemble`, and `RegressionPartitionedEnsemble` now support `gpuArray` (Parallel Computing Toolbox) input arguments, so that the functions can execute on a GPU.
- You can now use `gather` to create a `RegressionEnsemble`, `CompactRegressionEnsemble`, or `RegressionPartitionedEnsemble` object with properties stored in the local workspace from the equivalent object fitted using data stored as a GPU array.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

GPU Performance: Improved performance in fitting a classification ensemble on a GPU (requires Parallel Computing Toolbox)

The `fitcensemble` function has improved performance when it executes on a GPU. This code for fitting a `ClassificationEnsemble` model on a GPU is about 2.5x faster than in the previous release.

```
% Generate a data set
rng("default")
N = 1e6;
X = [mvnrnd([-1 -1],eye(2),N);...
    mvnrnd([1 1],eye(2),N)];
y = [zeros(N,1); ones(N,1)];

% Transfer the predictor data to the GPU
X = gpuArray(X);

% Create a regression ensemble model
template = templateTree(MaxNumSplits=30,....
    Reproducible=true);
mdl = fitcensemble(X,y,...
    NumLearningCycles=50,...
    Learners=template);

% Measure the execution time of the creation function
creationFunction = @()fitcensemble(X,y,...
    NumLearningCycles=50,...
    Learners=template);
t = gputimeit(creationFunction);
```

The approximate execution times are:

- R2021b: 52.4s
- R2022a: 21.4s

The code was timed on a Windows 10 test system with Intel Xeon CPU E5-1650 v3 @ 3.5 GHz and NVIDIA Titan V with 12 GB of GPU memory.

Functionality being removed or changed

Cost property of classification model objects stores the user-specified cost matrix
Behavior change

Starting in R2022a, the `Cost` property of all classification model objects stores the user-specified cost matrix, so that you can compute the observed misclassification cost using the specified cost value. The software stores the normalized prior probabilities (`Prior`) and observation weights (`W`) that do not reflect the penalties described in the cost matrix. Some classification model objects already had this behavior. The following list shows the objects that changed in R2022a:

- `ClassificationSVM`, `CompactClassificationSVM`, and `ClassificationPartitionedModel`, trained by the `fitcsvm` function
- `ClassificationEnsemble`, `CompactClassificationEnsemble`, `ClassificationBaggedEnsemble`, and `ClassificationPartitionedEnsemble`, trained by the `fitcensemble` function
- `ClassificationLinear` and `ClassificationPartitionedLinear`, trained by the `fitclinear` function
- `ClassificationKernel` and `ClassificationPartitionedKernel`, trained by the `fitckernel` function
- `TreeBagger` and `CompactTreeBagger`, trained by the `TreeBagger` function

Note that model training has not changed and, therefore, the decision boundaries between classes have not changed.

For training, the fitting functions update the specified prior probabilities by incorporating the penalties described in the specified cost matrix, and then normalize the prior probabilities and observation weights. This behavior has not changed. In previous releases, the software stored the default cost matrix in the `Cost` property and stored the prior probabilities and observation weights used for training in the `Prior` and `W` properties, respectively. Starting in R2022a, the software stores the user-specified cost matrix as is, and stores the normalized prior probabilities and observation weights that do not reflect the cost penalties. For more details, see Misclassification Cost Matrix, Prior Probabilities, and Observation Weights.

Some object functions use the `Cost`, `Prior`, and `W` properties:

- The `loss`, `resubLoss`, `kfoldLoss`, and `oobLoss` functions use the cost matrix stored in the `Cost` property if you specify the `LossFun` name-value argument as `"classifcost"` or `"mincost"`.
- The `loss` and `edge` functions use the prior probabilities stored in the `Prior` property to normalize observation weights of the input data.
- The `resubLoss`, `resubEdge`, `kfoldLoss`, `kfoldEdge`, `oobLoss`, `oobEdge`, `oobError`, and `oobMeanMargin` functions use the observation weights stored in the `W` property.

If you specify a nondefault cost matrix when you train a classification model, the object functions return a different value compared to previous releases.

If you want the software to handle the cost matrix, prior probabilities, and observation weights as in previous releases, adjust the prior probabilities and observation weights for the nondefault cost matrix, as described in Adjust Prior Probabilities and Observation Weights for Misclassification Cost Matrix. Then, when you train a classification model, specify the adjusted prior probabilities and observation weights by using the `Prior` and `Weights` name-value arguments, respectively, and use the default cost matrix.

Default LossFun value of the loss functions for ClassificationGAM and ClassificationNeuralNetwork has changed
Behavior change

The default value for the `LossFun` name-value argument has changed for generalized additive models and neural network models for classification. Starting in R2022a, the functions use the `"mincost"` option (minimal expected misclassification cost) as the default when a classification object uses posterior probabilities for classification scores. The following table shows the affected functions, previous default values, and new default values.

Function	Default Value in R2021b and Earlier	Default Value Starting in R2022a

`loss` of `ClassificationGAM` and `CompactClassificationGAM`
 	
`resubLoss` of `ClassificationGAM`
 	
`kfoldLoss` of `ClassificationPartitionedGAM`
	`"classiferror"`	`"mincost"` if the `ScoreTransform` property of the object is `'logit'`, and `"classiferror"` otherwise

`loss` of `ClassificationNeuralNetwork` and `CompactClassificationNeuralNetwork`
 	
`resubLoss` of `ClassificationNeuralNetwork`
 	
`kfoldLoss` of `ClassificationPartitionedModel` for a neural network model
	`"classiferror"`	`"mincost"`

You do not need to make any changes to your code if you use the default cost matrix (whose element value is 0 for correct classification and 1 for incorrect classification). The `"mincost"` option is equivalent to the `"classiferror"` option for the default cost matrix.

loss for regression models can return NaN for predictor data with missing values
Behavior change

The `loss` function no longer omits an observation with a NaN loss when computing the weighted average regression loss. Therefore, `loss` can now return NaN when the predictor data that you pass to the function contains any missing values. In most cases, if the test set observations do not contain missing predictors, the `loss` function does not return NaN.

This change improves the automatic selection of a regression model when you use `fitrauto`. Before this change, the software might select a model (expected to best predict the responses for new data) with few non-NaN predictors.

The regression models for which the `loss` object function might return NaN are presented in the following table. For more details, see the Compatibility Considerations for each `loss` function.

Model Type	Full or Compact Model Object	`loss` Object Function
Gaussian process regression (GPR) model	`RegressionGP`, `CompactRegressionGP`	`loss`
Gaussian kernel regression model	`RegressionKernel`	`loss`
Linear regression model	`RegressionLinear`	`loss`
Neural network regression model	`RegressionNeuralNetwork`, `CompactRegressionNeuralNetwork`	`loss`
Support vector machine (SVM) regression model	`RegressionSVM`, `CompactRegressionSVM`	`loss`

loss and edge for classification models can return NaN for predictor data with missing values
Behavior change

The `loss` and `edge` functions no longer omit an observation with a NaN score when computing the weighted average of classification losses and margins. Therefore, the functions can now return NaN when the predictor data that you pass to the functions contains any missing values. In most cases, if the test set observations do not contain missing predictors, the functions do not return NaN.

This change improves the automatic selection of a classification model when you use `fitcauto`. Before this change, the software might select a model (expected to best classify new data) with few non-NaN predictors.

The classification models for which the `loss` and `edge` functions might return NaN are presented in the following table. For more details, see the Compatibility Considerations for each function.

Model Type	Full or Compact Model Object	`loss` Object Function	`edge` Object Function
Discriminant analysis classification model	`ClassificationDiscriminant`, `CompactClassificationDiscriminant`	`loss`	`edge`
Ensemble of learners for classification	`ClassificationEnsemble`, `CompactClassificationEnsemble`	`loss`	`edge`

Model Type	Full or Compact Model Object	loss Object Function	edge Object Function
Gaussian kernel classification model	`ClassificationKernel`	`loss`	`edge`
Linear classification model	`ClassificationLinear`	`loss`	`edge`
k-nearest neighbor classification model	`ClassificationKNN`	`loss`	`edge`
Neural network classification model	`ClassificationNeuralNetwork`, `CompactClassificationNeuralNetwork`	`loss`	`edge`
Support vector machine (SVM) classification model	`ClassificationSVM`, `CompactClassificationSVM`	`loss`	`edge`

Default Cost value of the perfcurve function has changed
Behavior change

Starting in R2022a, the default value for the `Cost` name-value argument of the `perfcurve` function is `[0 1; 1 0]`, which is the same as the default misclassification cost matrix value for the new feature `rocmetrics` and the classifier training functions, such as `fitcsvm`, `fitctree`, and so on. In previous releases, the default `Cost` value of `perfcurve` is `[0 0.5; 0.5 0]`.

If you specify the `XCrit` or `YCrit` name-value argument as `'ecost'` (expected cost) and use the default `Cost` value, the function returns values in the output argument `X` or `Y` that are doubled compared to the values in previous releases.

If you specify the `XCrit` or `YCrit` name-value argument as a custom metric and use the default `Cost` value, the corresponding output argument value can be different depending on how the custom metric uses a cost matrix.

fitrlinear, fitclinear, fitcecoc, and fitcauto Functions: Regularization method determines the linear learner solver used during the optimization process
Behavior change

Starting in R2022a, when you specify to optimize hyperparameters for a linear learner and do not specify the solver to use, the functions `fitrlinear`, `fitclinear`, `fitcecoc`, and `fitcauto` (for multiclass classification) use either a Limited-memory BFGS (LBFGS) solver or a Sparse Reconstruction by Separable Approximation (SpaRSA) solver, depending on the regularization type selected during that iteration of the optimization process.

- When `Regularization` is `'ridge'`, the function sets the `Solver` value to `'lbfgs'` by default.
- When `Regularization` is `'lasso'`, the function sets the `Solver` value to `'sparsa'` by default.

In previous releases, the default solver selection during the optimization process depended on various factors, including the regularization type, learner type, and number of predictors.

The functions `fitrauto` and `fitcauto` (for binary classification) also use this updated solver selection process (since R2021a).

fitcauto and fitrauto Functions: Automatic selection of learners includes linear models when data is wide after categorical expansion
Behavior change

Starting in R2022a, if you specify `Learners` as `"auto"` and the data has more predictors than observations after the expansion of the categorical predictors (see Automatic Creation of Dummy Variables), then `fitcauto` and `fitrauto` include linear learners (`"linear"`) along with other models during the hyperparameter optimization. In previous releases, linear learners were not considered.

Statistics

Batch Drift Detection:Test and detect potential drifts between baseline and target data

The `detectdrift` function uses permutation testing to identify drift for each variable in baseline and target data. The function determines the drift status for each variable —stable, warning, and drift— and stores the results in a `DriftDiagnostics` object.

After performing permutation testing using `detectdrift`, you can display and visualize the test results.

- `summary` displays a summary of the test results.
- `ecdf` computes the empirical cumulative distribution function (ecdf) for a given list of variables.
- `histcounts` computes the histogram bin counts for a given list of variables.
- `plotDriftStatus` visualizes the test results and drift status.
- `plotEmpiricalCDF` visualizes the ecdf for a given variable.
- `plotHistogram` visualizes the histogram of the baseline and target data for a given variable.
- `plotPermutationResults` visualizes the histogram of permutation results for a given variable.

Estimation Statistics: Compute one- or two-sample effect sizes and visualize the two-sample effect size using the Gardner-Altman plot

`meanEffectSize` computes one- or two-sample effect sizes and exact or bootstrap confidence intervals.

`gardnerAltmanPlot` visualizes the effect size for two-sample data along with the exact or bootstrap confidence intervals.

Supported effect sizes are:

- Single-sample data — Cohen’s D, Robust Cohen’s D, and mean difference
- Two-sample data — Cohen’s D, Robust Cohen’s D, Cliff’s delta, Glass’s delta, Kolmogorov-Smirnov statistic, mean difference, and median difference

lassoglm Regularization: Accelerate fitting

The `lassoglm` function gains speed improvements for all response data distributions. This improvement is greatest for a normal response distribution.

`lassoglm` also gains two name-value arguments that can further accelerate the process of fitting data sets.

- `UseCovariance` allows you to use an algorithm that typically speeds training when the number of fitted features is less than the number of data points and the response distribution `distr = 'normal'`. The default setting of `UseCovariance`, `'auto'`, chooses the new algorithm in this case.
- `CacheSize` allows you to set a memory allocation for the new fitting algorithm.

This code for `lassoglm` assumes the response data is normally distributed, and is about 25x faster than in the previous release.

```
function timingTest1
rng(0,'twister');
N = 3e3; % number of samples
D = 3e2; % number of features
beta = randn(D,1);
beta0 = randn;
X = randn(N,D);
Y = beta0 + X*beta + randn(N,1);
tic
B = lassoglm(X,Y,'normal');
toc
```

The approximate execution times are:

- R2022a: 0.19s
- R2021b: 4.6s

This code for `lassoglm` assumes that the response data has a Poisson distribution, and is about 1.6x faster than in the previous release.

```
function timingTest2
rng default % For reproducibility
N = 1e4; % Number of samples
p = 1e2; % Number of features
X = randn(N,p);
beta = 3*rand(p,1); % Multiplicative coefficients
activep = randperm(p,p/10); % 10% nonzero coefficients
mu = exp(X(:,activep)*beta(activep));
y = poissrnd(mu);
tic
B2 = lassoglm(X,y,"poisson")
toc
```

The approximate execution times are:

- R2022a: 2.6s
- R2021b: 4.2s

These codes were timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the functions `timingTest1` and `timingTest2`.

For details, see the `lassoglm` reference page and its Coordinate Descent Algorithm section. For an example, see Use Correlation Matrix for Fitting `lassoglm`.

Compatibility Considerations

To use the same fitting algorithm as in previous releases, set the `UseCovariance` argument to `false`.

The returned `FitInfo` structure contains a new field, `UseCovariance`, which specifies the fitting algorithm used by the function.

lasso Regularization: Accelerate fitting

The `lasso` function gains speed improvements.

This code for `lasso` is about 8x faster than in the previous release.

```
function timingTest3
rng default % For reproducibility
X = randn(10e6,10);
coef = randi(100,10,1);
y = X*coef + randn(10e6,1)*0.1;
tic
B = lasso(X,y)
toc
```

The approximate execution times are:

- R2022a: 0.7s
- R2021b: 5.4s

This code was timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the function `timingTest3`.

multcompare Function: Perform Dunnett's test

The `multcompare` function now supports Dunnett's test for multiple comparisons against a control group. Specify the `CriticalValueType` name-value argument as `dunnet`, and specify the control group by using the `Control` name-value argument.

To speed up the computation, you can specify `Approximate=true` (default option for the results of the `anovan` function). `multcompare` then uses an approximation method when integrating the multivariate *t* distribution to find a critical value for Dunnett's test.

Compatibility Considerations

The `CType` name-value argument has been renamed to `CriticalValueType` to better indicate its functionality.

grpstats Function: Expanded support for additional data types

The `grpstats` function computes group summary statistics for the variables in input data (table or array), where you can specify the group by using one or more grouping variables. The function now supports computing summary statistics for a variable in a categorical, datetime, duration, or calendar duration vector, a character or string array, or a cell array of character vectors, in addition to the previously supported data types (numeric and logical vector). You can use the built-in summary statistics `gname` (group name) and `numel` (group count) for the newly supported data types. For any other types of summary statistics, you must provide a custom function. For an example, see Compute Custom Summary Statistics.

GPU Support: Additional distribution functions now accept gpuArray inputs (requires Parallel Computing Toolbox)

The functions `nbincdf`, `nbininv`, `nbinpdf`, `nbinstat`, `gpcdf`, `gpinv`, `gplike`, `gppdf`, `gpstat`, `gevcdf`, `gevinv`, `gevlike`, `gevpdf`, and `gevstat` are enhanced to accept `gpuArray` input arguments, so that the functions can execute on a GPU. The functions fully support GPU arrays.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

Descriptive Statistics: Run functions in a thread-based environment

You can now run the following functions in the background using MATLAB `backgroundPool`:

- `geomean`
- `harmmean`
- `kurtosis`
- `mad`
- `moment`
- `range`
- `skewness`
- `trimmean`
- `zscore`

For more information, see Run MATLAB Functions in Thread-Based Environment.

Visualization

gplotmatrix and gscatter Functions: Generate MATLAB code to recreate a figure

The `gplotmatrix` and `gscatter` functions are enhanced to generate MATLAB code that better captures modifications you make using the plot tools.

When you click **File > Generate Code** for a figure created by one of the functions, the generated code is displayed in the MATLAB Editor. You can use the code to reproduce the figure. Generated files do not store the data necessary to recreate a figure, so you must supply the data arguments.

dendrogram Function: Specify axes to plot into

You can now specify the axes for the `dendrogram` function to plot into. Use either the `AX` input argument or the `'Parent'` name-value argument in the call to `dendrogram`.

For example, `dendrogram(AX,tree)` or `dendrogram(tree,'Parent',AX)` plot into axes with handle `AX` instead of the current axes.

For more information on creating an axes object, see `axes` and `uiaxes`.

Functionality being removed or changed

gscatter uses the MATLAB default color scheme

Behavior change

Starting in R2022a, the `gscatter` function uses the MATLAB default color scheme to determine marker colors, following the order specified in the `ColorOrder` property of the axes.

In previous releases, the `gscatter` function uses the colormap returned by the `hsv` function. If you want to determine marker colors using the `hsv` colormap, specify marker colors (fourth input argument of `gscatter`) as `hsv(numGroup)`, where `numGroup` is the number of unique group combinations in the grouping variables.

If you use the new default color scheme and the number of unique groups exceeds the number of default colors (7), then `gscatter` cycles through the default values as needed. If you want to use different colors for different groups, specify the marker colors as `hsv(numGroups)`. For an example, see Specify Marker Colors.

R2021b

Deployment

Neural Network Prediction Blocks: Simulate models and generate code in Simulink

You can now integrate the `predict` function of neural network classification and regression objects into Simulink using the new prediction blocks. You can import a trained model and configure data types using the Block Parameters dialog box.

- ClassificationNeuralNetwork Predict — This block returns classified labels for new data using an imported neural network classification object (`ClassificationNeuralNetwork` or `CompactClassificationNeuralNetwork`).
- RegressionNeuralNetwork Predict — This block predicts responses for new data using an imported neural network regression object (`RegressionNeuralNetwork` or `CompactRegressionNeuralNetwork`).

The blocks support C/C++ code generation and fixed-point conversion. You can generate C and C++ code using Simulink Coder, and you can design and simulate fixed-point systems using Fixed-Point Designer.

The blocks also support a half-precision data type for parameters for simulation and code generation. Half-precision data types occupy only 16 bits of memory, but their floating-point representation enables them to handle wider dynamic ranges than integer or fixed-point data types of the same size. Using half-precision data types requires Fixed-Point Designer or MATLAB Coder.

Export models from Classification Learner or Regression Learner for deployment to MATLAB Production Server (requires MATLAB Compiler SDK)

After you train a model in the Classification Learner or Regression Learner app, you can export the model for deployment to MATLAB Production Server.

- Select the trained model in the **Models** pane. On the **Classification Learner** or **Regression Learner** tab, in the **Export** section, click **Export Model** and select **Export Model for Deployment**.
- In the Select Project File for Model Deployment dialog box, select a location and name for your project file.
- In the autogenerated `predictFunction.m` file, inspect and amend the code as needed.
- Use the Production Server Compiler app to package your model and prediction function. You can simulate the model deployment to MATLAB Production Server by clicking the **Test Client** button in the **Test** section of the **Compiler** tab, and then package your code by clicking the **Package** button in the **Package** section.

For an example, see Deploy Model Trained in Classification Learner to MATLAB Production Server or Deploy Model Trained in Regression Learner to MATLAB Production Server.

Functionality being removed or changed

saveCompactModel and loadCompactModel functions have been removed
Errors

`saveCompactModel` and `loadCompactModel` have been removed. Use `saveLearnerForCoder` and `loadLearnerForCoder` instead.

`saveLearnerForCoder` and `loadLearnerForCoder` provide broader functionality, including fixed-point code generation for supported models.

This table shows typical usage of `saveCompactModel` and `loadCompactModel` and how to update your code to use `saveLearnerForCoder` and `loadLearnerForCoder`.

Removed	**Recommended**
`saveCompactModel(Model,'MyModel');`	`saveLearnerForCoder(Model,'MyModel');`
`Mdl = loadCompactModel('MyModel');`	`Mdl = loadLearnerForCoder('MyModel');`

Apps

Cluster Data Live Editor Task: Interactively cluster data using k-means clustering and generate code

Use the new **Cluster Data** Live Editor Task to interactively perform *k*-means clustering in a live script. You can select the number of clusters manually or select to evaluate the optimal number of clusters. You also can specify optional clustering parameters, including the distance metric and the number of replicates. The task plots the cluster regions, allowing you to interactively explore the effects of changing parameter values and options. The task also automatically generates code that becomes part of your live script.

To use the **Cluster Data** task in the Live Editor, on the **Live Editor** tab, select **Task > Cluster Data**. Alternatively, in a code block in a live script, begin typing the task name and select the task from the suggested command completions. For more information about Live Editor tasks, see Add Interactive Tasks to a Live Script.

Machine Learning Apps: Optimize hyperparameters of neural network models

In Classification Learner and Regression Learner, you can tune the hyperparameters of a neural network model using hyperparameter optimization.

- On the **Classification Learner** or **Regression Learner** tab, in the **Model Type** section, click the arrow to open the gallery. Click the **Optimizable Neural Network** model.
- In the **Model Type** section, click **Advanced** and select **Advanced**. In the dialog box, select the **Optimize** check boxes for the hyperparameters that you want to optimize. Under **Values**, select the fixed values for the hyperparameters that you do not want to optimize. Click **OK**.

  When the software optimizes the number of fully connected layers, it also optimizes the size of each layer. You cannot specify the number of fully connected layers and optimize the layer sizes. Similarly, you cannot specify the layer sizes and optimize the number of layers.
- (Optional) In the **Model Type** section, click **Advanced** and select **Optimizer Options**. Select options to specify how the optimization is performed, and click **OK**.

After selecting options, train the optimizable model. The app tries different combinations of hyperparameter values based on your selections and returns a model with the optimized hyperparameter values. You cannot use the **Use Parallel** button to run the optimization in parallel.

When the optimizable model stops training, the **Current Model Summary** pane in the bottom left of the app shows the optimized hyperparameter values. The exported optimizable model and generated MATLAB code treat these values as fixed model hyperparameters.

For more details, see Hyperparameter Optimization in Classification Learner App or Hyperparameter Optimization in Regression Learner App.

Machine Learning Apps: Compare plots across models by changing the plot layout

Visualize the results of models trained in Classification Learner or Regression Learner by using the plot options in the **Plots** section of the **Classification Learner** or **Regression Learner** tab. You can

rearrange the layout of the plots to compare results across multiple models: use the options in the **Layout** button, drag and drop plots, or select the options provided by the Document Actions arrow located to the right of the model plot tabs.

For more information, see Compare Model Plots by Changing Layout for classification or Compare Model Plots by Changing Layout for regression.

Classification Learner: Train Gaussian kernel classifiers for nonlinear classification of data with many observations

In Classification Learner, you can use kernel classifiers to perform nonlinear classification of data with many observations. The Gaussian kernel classification models map predictors in a low-dimensional space into a high-dimensional space, and then fit a linear model to the transformed predictors in the high-dimensional space. Choose between fitting an SVM linear model and fitting a logistic regression linear model in the expanded space. For large in-memory data, kernel classifiers tend to train and predict faster than SVM classifiers with Gaussian kernels.

To create a kernel classifier in Classification Learner, click the arrow in the **Model Type** section of the **Classification Learner** tab to open the gallery. In the **Kernel Classifiers** group, click one of the available models. Then, in the **Training** section, select **Train**.

For more information, see Kernel Approximation Classifiers and Train Kernel Approximation Classifiers Using Classification Learner App.

Machine Learning Apps: Export models for deployment to MATLAB Production Server (requires MATLAB Compiler SDK)

After you train a model in Classification Learner or Regression Learner, you can export the model for deployment to MATLAB Production Server.

- Select the trained model in the **Models** pane. On the **Classification Learner** or **Regression Learner** tab, in the **Export** section, click **Export Model** and select **Export Model for Deployment**.
- In the Select Project File for Model Deployment dialog box, select a location and name for your project file.
- In the autogenerated `predictFunction.m` file, inspect and amend the code as needed.
- Use the Production Server Compiler app to package your model and prediction function. You can simulate the model deployment to MATLAB Production Server by clicking the **Test Client** button in the **Test** section of the **Compiler** tab, and then package your code by clicking the **Package** button in the **Package** section.

For an example, see Deploy Model Trained in Classification Learner to MATLAB Production Server or Deploy Model Trained in Regression Learner to MATLAB Production Server.

Functionality being removed or changed

Each model has its own set of plots

Behavior change

Starting in R2021b, each model in Classification Learner and Regression Learner has its own set of plots. As a result, you can create the same plot for multiple models and compare the results side-by-

side. In previous releases, you can create only one of each type of plot, and the software updates the plot results based on the selected model.

To view a plot for a specific model, select the model in the **Models** pane. On the **Classification Learner** or **Regression Learner** tab, in the **Plots** section, click the arrow to open the gallery. Click the button for the plot you want to see.

Note that after you train a model in Classification Learner, the app automatically opens the validation confusion matrix for that model. Similarly, after you train a model in Regression Learner, the app automatically opens the response plot for that model. In either app, if you train an "All" preset model (for example, **All Trees**), the app opens a plot for the first model only.

Machine Learning

Anomaly Detection: Detect anomalies in data using the isolation forest algorithm

Use the `iforest` function, `IsolationForest` object, and `isanomaly` object function for anomaly detection. This feature uses the isolation forest algorithm [1].

- Outlier detection (detecting anomalies in training data) — Detect anomalies in training data by passing the data to the `iforest` function. The function returns an `IsolationForest` object, a logical array indicating anomalies, and a numeric array of anomaly scores. The score values in the numeric array are between 0 and 1, and a value close to 1 indicates an outlier. You can specify the fraction of anomalies in the data by using the `ContaminationFraction` name-value argument.
- Novelty detection (detecting anomalies in new data with uncontaminated training data) — Create an `IsolationForest` object by passing uncontaminated training data (data with no outliers) to the `iforest` function. Detect anomalies in new data by passing the object and the new data to the object function `isanomaly`. The `isanomaly` function returns a logical array indicating anomalies and a numeric array of anomaly scores. You can specify the score threshold by using the `ScoreThreshold` name-value argument.

You can perform the computation in parallel by specifying the `UseParallel` name-value argument of `iforest` or `isanomaly` as `true`. This option requires Parallel Computing Toolbox.

Feature Engineering: Automatically create new features before training a regression model

Like `gencfeatures`, the `genrfeatures` function enables you to automate the feature engineering process in the context of a machine learning workflow. Before passing tabular training data to a regression model, you can create new features from the predictors in the data by using `genrfeatures`. Use the returned data to train the model.

`genrfeatures` generates new features based on the `TargetLearner` type, either `"linear"` or `"bag"`. That is, the function creates and selects new features assuming that they will be used to train a linear model or a bagged ensemble model.

- To generate features for an interpretable model, specify `TargetLearner="linear"` in the call to `genrfeatures`. You can then use the returned data to train a linear regression model. For example, use `fitrlinear`.
- To generate features that can lead to better model accuracy, specify `TargetLearner="bag"` in the call to `genrfeatures`. You can then use the returned data to train a bagged ensemble regression model. For example, use `fitrensemble` with `Method="bag"`.

To better understand the generated features, use the `describe` function of the returned `FeatureTransformer` object. To apply the same training set feature transformations to a test or validation set, use the `transform` function of the `FeatureTransformer` object.

Incremental Learning: Naive Bayes incremental learner supports multinomial or multivariate multinomial predictor variables, and custom prediction and loss options

The naive Bayes classification model for incremental learning (`incrementalClassificationNaiveBayes`) supports numeric data representing categorical predictor variables, which enables you to specify a multinomial naive Bayes model (for example, a bag-of-tokens model) or a model containing normal or multivariate multinomial predictor variables. During incremental training with `fit` or `updateMetricsAndFit`, the model adapts to new category levels.

For computing weighted predictions and losses in real time, such as cost-sensitive predictions, `predict` and `loss` enable you to override the misclassification cost (`Cost` property) and prior class probability distribution (`Prior` property) of the incremental model by specifying the `Cost` and `Prior` name-value arguments. Similarly, you can override the score transform function (`ScoreTransform` property) of the incremental model by using the `ScoreTransform` name-value argument of the functions.

Automated Model Selection: Automatically select a model with tuned hyperparameters using ASHA optimization

The `fitcauto` and `fitrauto` functions try a selection of model types with different hyperparameters and return a final model that is expected to perform well on new data. By default, the functions use Bayesian optimization to select and assess models. You can now use an asynchronous successive halving algorithm (ASHA) instead. ASHA optimization often finds good solutions faster than Bayesian optimization for data sets with many observations.

When you use `fitcauto` or `fitrauto` with ASHA optimization, the function randomly chooses several models with different hyperparameter values and trains them on a small subset of the training data. If the performance of a particular model is promising, the model is promoted and trained on a larger amount of the training data. This process repeats, and successful models are trained on progressively larger amounts of data. By default, at the end of the optimization, `fitcauto` chooses the model that has the lowest cross-validation classification error, and `fitrauto` chooses the model that has the lowest cross-validation mean squared error (MSE). For more details, see ASHA Optimization for classification or ASHA Optimization for regression.

To use ASHA optimization, specify `"HyperparameterOptimizationOptions",struct("Optimizer","asha")` in the call to `fitcauto` or `fitrauto`. You can include additional fields in the structure to control other aspects of the optimization.

Machine Learning Using Neural Networks: Optimize hyperparameters using fitcnet and fitrnet

To search for good hyperparameters automatically when creating a neural network, the `fitcnet` and `fitrnet` functions gain the name-value arguments `OptimizeHyperparameters` and `HyperparameterOptimizationOptions`. These arguments cause the fitting function to search for a minimal cross-validation error, exactly as with other fitting functions. For classification examples, see Improve Neural Network Classifier Using `OptimizeHyperparameters` and Customize Neural Network Classifier Optimization. For regression examples, see Minimize Cross-Validation Error in Neural Network and Custom Hyperparameter Optimization in Neural Network.

Generalized Additive Model (GAM): Optimize hyperparameters using fitcgam and fitrgam

To search for optimal hyperparameters automatically for a generalized additive model, use the name-value arguments `OptimizeHyperparameters` and `HyperparameterOptimizationOptions` of the `fitcgam` or `fitrgam` function. Specify hyperparameters to optimize by using `OptimizeHyperparameters`, and specify optimization options by using `HyperparameterOptimizationOptions`. These arguments cause the fitting function to search for a minimal cross-validation error, just as with other fitting functions. For a classification example, see Optimize GAM Using `OptimizeHyperparameters`. For a regression example, see Optimize GAM Using `OptimizeHyperparameters`.

GAM for Regression: Compute the prediction interval of the response

Specify `FitStandardDeviation` as `true` when you train a generalized additive model for regression by using the `fitrgam` function. Then, the function trains an additional model for the standard deviation of the response variable, and you can compute the standard deviation values for given observations by passing the trained model to the `predict`, `resubPredict`, or `kfoldPredict` function. These functions also return the prediction intervals of the response variable. For an example, see Plot Prediction Intervals.

GPU Support: fitctree, fitrtree, fitcensemble, and fitcecoc now accept gpuArray inputs (requires Parallel Computing Toolbox)

The Statistics and Machine Learning Toolbox functions `fitctree`, `fitrtree`, `fitcensemble`, and `fitcecoc` are enhanced to accept `gpuArray` (Parallel Computing Toolbox) input arguments, so that the functions can execute on a GPU. The functions support GPU arrays with some limitations.

The object functions of the models `ClassificationTree`, `RegressionTree`, `ClassificationEnsemble`, `ClassificationECOC`, `ClassificationPartitionedModel`, `RegressionPartitionedModel`, `ClassificationPartitionedEnsemble`, and `ClassificationPartitionedECOC` support `gpuArray` (Parallel Computing Toolbox) input arguments in one of the following ways, so that the functions can execute on a GPU:

- The object function fully supports GPU arrays and model objects fitted with GPU array input arguments.
- The object function offers limited support for GPU arrays and full support for model objects fitted with GPU array input arguments.
- The object function supports only model objects fitted with GPU array input arguments.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

Bayesian Optimization: Create an optimizableVariable object for a nonnegative, integer-valued, log-transformed variable

You can now specify the lower bound of an `optimizableVariable` object as 0 for an integer-valued, log-transformed variable. This feature is useful for a nonnegative integer-valued variable that has a wide range. For example, you can use such a variable for the `Interactions` name-value argument of the `fitcgam` function. This argument specifies the number of interaction terms in a generalized

additive model. When you include 0 in the argument's range, the optimization can include a model with no interaction terms and models with multiple interaction terms. Also, by using a log transformation, the optimization can effectively search a wide range of values for the number of interaction terms.

One-Hot Encoding: Encode and decode categorical data into vectors

Use the `onehotencode` function to transform categorical data labels into one-hot vectors. One-hot encoding expands categorical data into vectors with the same number of elements as the total number of categories. The vectors contain a 1 in the position corresponding to the appropriate category, and 0 otherwise. To encode data labels, you can also use `dummyvar`, which creates dummy variables from grouping variables.

Use the `onehotdecode` function to decode one-hot encoded vectors. The function takes one-hot encoded vectors or dummy variables, and determines the encoded category.

Note that many Statistics and Machine Learning Toolbox functions can operate directly on tables and categorical data, in which case you do not need to use the `onehotencode` and `onehotdecode` functions. One-hot encoding and decoding are useful for functions that accept only numeric inputs.

Functionality being removed or changed

Naive Bayes incremental fitting functions compute biased standard deviations for conditionally normal predictor variables

Behavior change

Starting in R2021b, naive Bayes incremental fitting functions `fit` and `updateMetricsAndFit` compute biased estimates of the standard deviations for conditionally normal predictor variables during training. In other words, for each class k, incremental fitting functions normalize the sum of square deviations of the conditionally normal predictor x_j by the number of training observations in k, n_k. Before R2021b, naive Bayes incremental fitting functions computed the unbiased standard deviation, like `fitcnb`.The currently returned weighted standard deviation estimates differ from those computed before R2021b by a factor of

$$1-\frac{\sum_{\{i:y_i=k\}} w_i^2}{\left(\sum_{\{i:y_i=k\}} w_i\right)^2}.$$

The factor approaches 1 as the sample size increases.

Statistics

ecdf Function: Empirical cumulative distribution function enhanced to include left-censoring, double-censoring, and interval-censoring

The `ecdf` function now supports left-censored, double-censored, and interval-censored data, in addition to the previously supported data types (fully observed data and right-censored data). The function computes the empirical cumulative distribution function (cdf) values using different algorithms, depending on the censorship information.

- If the data contains either right-censored or left-censored observations, `ecdf` computes the Kaplan-Meier estimate.
- If the data is double-censored, containing both left-censored and right-censored observations, `ecdf` uses Turnbull's algorithm [3][4].
- If the data contains interval-censored observations, `ecdf` uses the expectation-maximization iterative convex minorant (EMICM) algorithm [5]. For interval-censored observations, the function returns estimates evaluated at intervals, rather than at points, and visualizes the intervals using shaded rectangles.

mle and mlecov Functions: Maximum likelihood estimation enhanced to include left-censoring, double-censoring, interval-censoring, and truncation

The `mle` and `mlecov` functions now support left-censored, double-censored, interval-censored, and truncated data, in addition to the previously supported data types (fully observed data and right-censored data).

Loguniform Distributions: Evaluate distributions and generate random samples using the LoguniformDistribution object

Create a probability distribution object `LoguniformDistribution` for a loguniform distribution by using the `makedist` function. Pass the distribution name (`'loguniform'`) and, optionally, the distribution parameters (`Lower` and `Upper` name-value arguments for the lower limit and upper limit, respectively) to `makedist`. Once you create the object, you can use the object functions to:

- Evaluate the cumulative distribution function (`cdf`) or the inverse cumulative distribution function (`icdf`).
- Evaluate the probability density function (`pdf`).
- Generate random samples from the distribution (`random`).
- Compute summary statistics, including mean (`mean`), median (`median`), interquartile range (`iqr`), variance (`var`), and standard deviation (`std`).
- Truncate the distribution to specified lower and upper limits (`truncate`).

lasso Regularization: Accelerate fitting

The `lasso` function gains speed improvements. `lasso` also gains two name-value arguments that can further accelerate the process of fitting data sets.

- `'UseCovariance'` allows you to use an algorithm that typically speeds training when the number of fitted features is less than the number of data points. The default setting of this argument, `'auto'`, chooses the new algorithm for fewer fitted features than data points.
- `'CacheSize'` allows you to set a memory allocation for the new fitting algorithm.

For details, see the `lasso` reference page and its Coordinate Descent Algorithm section.

This code for `lasso` is about 20 times faster than in the previous release, and does not use the new algorithm.

```
rng default
N = 1e2;
p = 1e3;
X = randn(N,p);
beta = randn(p,1);
beta0 = randn;
y = beta0 + X*beta + randn(N,1);
tic
B = lasso(X,y);
toc
```

The approximate execution times are:

- R2021a: 3.8s
- R2021b: 0.19s

This code was timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system.

Compatibility Considerations

To use the same fitting algorithm as in previous releases, set the `UseCovariance` argument to `false`.

The returned `FitInfo` structure contains a new field, `UseCovariance`, which specifies the fitting algorithm used by the function.

GPU Support: fitdist, mle, betafit, gevfit, gpfit, and nbinfit now accept gpuArray inputs (requires Parallel Computing Toolbox)

The Statistics and Machine Learning Toolbox functions `fitdist`, `mle`, `betafit`, `gevfit`, `gpfit`, and `nbinfit` are enhanced to accept `gpuArray` (Parallel Computing Toolbox) input arguments, so that the functions can execute on a GPU. `betafit`, `gevfit`, `gpfit`, and `nbinfit` fully support GPU arrays. `fitdist` and `mle` support GPU arrays with some limitations.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

gather Function: Expanded support for additional objects

The `gather` function gathers all properties of an object fitted using data stored as a GPU array, and returns an output object with properties stored in the local workspace. You can now call `gather` on additional Statistics and Machine Learning Toolbox objects. For more information on the supported

regression model, classification model, and probability distribution objects, see the input argument `obj` of the `gather` function.

Visualization

shapley Enhancements: Display query point prediction and average prediction values in a Shapley value plot

The object function `plot` of `shapley` now creates a plot with a title and subtitle. The title shows that the plot is for the Shapley explanation, and the subtitle shows the query point prediction value and the average prediction value, averaged over the given predictor data. These two prediction values are helpful in understanding the Shapley values, because the Shapley value of a feature for a query point is the contribution of the feature to the deviation from the average prediction. You can also find the average value from the new property `Intercept` of a `shapley` object.

Compatibility Considerations

Starting in R2021b, the default `TickLabelInterpreter` value of the axes is `'none'`, enabling the `shapley` plots to label the *y*-axis by using the predictor names from the `PredictorNames` field of the `BlackboxModel` property. In the previous release, the default value is `'tex'`, which interprets labels using a subset of the TeX markup.

R2021a

Deployment

Tree Prediction Blocks and Ensemble of Trees Prediction Blocks: Simulate models and generate code in Simulink

You can now integrate the `predict` function of decision tree objects and ensemble objects of tree weak learners into Simulink using the new prediction blocks instead of a MATLAB Function block. You can import a trained model and configure data types using the Block Parameters dialog box.

- ClassificationTree Predict — This block returns classified labels and predicted scores (optional) for new data using an imported decision tree object (`ClassificationTree` or `CompactClassificationTree`) for classification.
- RegressionTree Predict — This block predicts responses for new data using an imported decision tree object (`RegressionTree` or `CompactRegressionTree`) for regression.
- ClassificationEnsemble Predict — This block returns classified labels and predicted scores (optional) for new data using an imported ensemble object (`ClassificationEnsemble`, `CompactClassificationEnsemble`, or `ClassificationBaggedEnsemble`) for classification. The block supports an ensemble object with tree weak learners. You can train an ensemble by using a boosting algorithm (for example, adaptive boosting, adaptive logistic regression, and robust boosting) or bootstrap aggregation (bagging, for example, the random forest algorithm).
- RegressionEnsemble Predict — This block predicts responses for new data using an imported ensemble object (`RegressionEnsemble`, `CompactRegressionEnsemble`, or `RegressionBaggedEnsemble`) for regression. The block supports an object with tree weak learners. You can train an ensemble by using least-squares boosting or by using bootstrap aggregation (bagging, for example, the random forest algorithm).

The blocks support C/C++ code generation and fixed-point conversion. You can generate C and C++ code using Simulink Coder, and you can design and simulate fixed-point systems using Fixed-Point Designer.

The blocks also support a half-precision data type for parameters for simulation and code generation. Half-precision data types occupy only 16 bits of memory, but their floating-point representation enables them to handle wider dynamic ranges than integer or fixed-point data types of the same size. Using half-precision data types requires Fixed-Point Designer or MATLAB Coder.

Generate C/C++ code for performing incremental learning using linear regression or binary classification model functions (requires MATLAB Coder)

You can generate C/C++ code that trains a linear regression or binary classification model and assesses its performance in real time as batches of observations become available. You can save an `incrementalClassificationLinear` or `incrementalRegressionLinear` model by using `saveLearnerForCoder`, and then load the model in the entry-point function for code generation by using `loadLearnerForCoder` (see Introduction to Code Generation). These incremental learning functions support code generation:

- The `fit` function fits the model to incoming data by updating the coefficients.
- The `updateMetrics` function evaluates the performance of the model as it processes incoming observations. The function writes specified metrics, measured cumulatively and within a specified window of processed observations, to the `Metrics` model property.

- The `updateMetricsAndFit` function first evaluates the performance of the model by calling `updateMetrics` on incoming data, and then fits the model to that data by calling `fit`.
- The `predict` function predicts responses given incoming predictor data.
- The `loss` function returns the classification or regression loss given incoming predictor and response data. Unlike `updateMetrics`, `loss` does not write the computed loss to the model.

Generate C/C++ code for prediction by using a machine learning model with heterogeneous data (requires MATLAB Coder)

Code generation for prediction of a machine learning model now supports heterogeneous predictors and class labels with the `categorical` data type.

- You can now train a model with heterogeneous predictors containing numeric and categorical variables, and then generate C/C++ code for prediction using the model. Specify categorical predictors in the predictor data using the `'CategoricalPredictors'` or `'CategoricalVars'` name-value argument when you train a model. You do not need to create dummy variables for categorical predictors to generate code.
- You can now train a classification model for class labels with the `categorical` data type, in addition to `single`, `double`, `logical`, `char`, and `cell`, and generate code for prediction using the model.

Fixed-Point Deployment Example

A new example, Human Activity Recognition Simulink Model for Fixed-Point Deployment, shows how to prepare a Simulink model that classifies human activity based on sensor signals for code generation and deployment to low-power hardware. The example provides a classification model that is ready for deployment to a BBC micro:bit device.

Functionality being removed or changed

Default parameter values and supported inherited parameter options have been changed for the ClassificationSVM Predict block and RegressionSVM Predict block
Behavior change

Starting in R2021a, in the ClassificationSVM Predict block, the default value of the parameters **Score data type** and **Raw score data type** is `Inherit: auto`. The default value of the **Label data type** parameter is `Inherit: Inherit via back propagation` for numeric and logical labels, and `Inherit: auto` for nonnumeric labels. If you specified **Label data type** as `Inherit: Inherit via back propagation` for nonnumeric labels or `Inherit: Inherit from 'Constant value'`, then change the value to `Inherit: auto`.

In the ClassificationSVM Predict block and RegressionSVM Predict block, the **Kernel data type** parameter does not support inherited options. You can specify **Kernel data type** as a supported data type name or data type object.

Apps

Machine Learning Apps: Train neural network models to predict responses

You can train fully connected, feedforward neural networks for classification or regression in Classification Learner or Regression Learner, respectively. Select neural networks of various sizes, with one, two, or three fully connected layers, excluding the final connected layer for classification or regression. You can adjust the size of each fully connected layer, excluding the last, and choose the activation function for the layers.

To create a neural network model in Classification Learner or Regression Learner, click the arrow in the **Model Type** section of the **Classification Learner** or **Regression Learner** tab to open the gallery. In the **Neural Network Classifiers** or **Neural Networks** group, click one of the available models. Then, in the **Training** section, select **Train**.

For more information, see Neural Network Classifiers and Neural Networks.

Machine Learning Apps: Import test data directly into apps

To evaluate how models trained in Classification Learner or Regression Learner perform on new data, you can import a test data set and compute test metrics inside the apps. You can also visualize the test results using plots.

For example, after training one or more models in Classification Learner or Regression Learner, click **Test Data** and select **From Workspace** in the **Testing** section of the **Classification Learner** or **Regression Learner** tab. In the Import Test Data dialog box, select the test data set from the **Test Data Set Variable** list, and click **Import**.

In the **Models** pane, select the trained model for which you want to compute test metrics. In the **Testing** section, click **Test All** and select **Test Selected**. The app displays the test results in the **Current Model Summary** pane.

For more information, see Evaluate Test Set Model Performance (classification) and Evaluate Test Set Model Performance (regression).

Machine Learning Apps: Sort and delete models

After training models in Classification Learner or Regression Learner, you can sort the models based on different metrics. For example, in the Classification Learner app, you can sort models by model accuracy or total misclassification cost. In the Regression Learner app, you can sort models using the root mean squared error (RMSE), mean squared error (MSE), R-squared value, or mean absolute error (MAE). To select a metric for model sorting, use the **Sort by** list at the top of the **Models** pane.

You can delete models listed in the **Models** pane of Classification Learner or Regression Learner. Select the model you want to delete and click the **Delete selected model** button in the upper right of the pane, or right-click the model and select **Delete model**.

Machine Learning Apps: Load data into the apps from the command line

In the command line call to `classificationLearner` or `regressionLearner`, you can specify the predictor data, response variable, and validation type to use in Classification Learner or Regression Learner, respectively.

For example, this code opens the Classification Learner app and populates the New Session from Arguments dialog box with the predictor data `meas`, response variable `species`, and default 5-fold cross-validation.

```
load fisheriris
classificationLearner(meas,species,'CrossVal','on')
```

For more information, see **Classification Learner** and **Regression Learner**.

Classification Learner: Specify new scaling options for a parallel coordinates plot

In the Classification Learner app, you can adjust the predictor scaling used in a parallel coordinates plot. On the **Classification Learner** tab, in the **Plots** section, click **Parallel Coordinates**. To the right of the plot, under the **Plot** heading, select from the **Scaling** list options: `None`, `Range`, `Z-Score`, `Zero Mean`, `Unit Variance`, and `L2 Norm`. The `Z-Score` option replaces the previous `Standardized` option. For more information, see Investigate Features in the Parallel Coordinates Plot.

Machine Learning Apps: Import data from a file, generate code, and train models in parallel in MATLAB Online

When you use Classification Learner or Regression Learner in MATLAB Online™, you can now import data from a file and generate MATLAB code to train a model with new data. You can also train models in parallel using a Cloud Center cluster (requires Parallel Computing Toolbox). For more information, see Use Parallel Computing Toolbox with Cloud Center Cluster in MATLAB Online (Parallel Computing Toolbox).

Functionality being removed or changed

Use Parallel button is off by default

Behavior change

Starting in R2021a, the **Use Parallel** button is off by default in the Classification Learner and Regression Learner apps. In previous releases, if you had Parallel Computing Toolbox, the app automatically selected the **Use Parallel** button by default. Now, if you have Parallel Computing Toolbox and want to use parallel training, click the **Use Parallel** button in the **Training** section of the **Classification Learner** or **Regression Learner** tab.

Classification Learner selects last nonnumeric variable as response by default

Behavior change

Starting in R2021a, when you open the Classification Learner app and specify a data set variable in the New Session dialog box, the app selects the last nonnumeric variable in the data set as the

response by default. In previous releases, the app selected the first nonnumeric variable in the data set as the response by default. You can still choose a different response variable within the **Response** section of the New Session dialog box.

View tab has been removed

Starting in R2021a, the **View** tab is no longer available in the Classification Learner and Regression Learner apps. In previous releases, you could use the options on the **View** tab to control the layout of the plots in the apps.

Machine Learning

Machine Learning Using Neural Networks: Create fully connected, feedforward neural networks for classification and regression using fitcnet and fitrnet

Use `fitcnet` and `fitrnet` to train fully connected, feedforward neural networks for classification and regression, respectively. By default, a returned neural network includes a fully connected layer of size 10, followed by a rectified linear unit (ReLU) activation. You can adjust the size and activation function of each fully connected layer, excluding the last, by specifying the `LayerSizes` and `Activations` name-value arguments. Each classification neural network has a final fully connected layer followed by a softmax activation function. Each regression neural network has a final fully connected layer that does not have an activation function. The software uses a limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton solver (LBFGS) to train neural networks.

By default, `fitcnet` and `fitrnet` return `ClassificationNeuralNetwork` and `RegressionNeuralNetwork` model objects, respectively. These objects have functions that resemble those of other classification and regression model objects in Statistics and Machine Learning Toolbox. For example, to predict class labels or responses for new data, you can pass a trained classification or regression model object to `predict`.

Interpretability: Fit a generalized additive model (GAM) for binary classification and regression using the fitcgam and fitrgam functions

Use `fitcgam` and `fitrgam` to fit a generalized additive model for binary classification and regression, respectively.

A generalized additive model is an interpretable model that explains a response variable using a sum of univariate and bivariate shape functions. The `fitcgam` and `fitrgam` functions use a nonparametric shape function for each predictor and, optionally, each pair of predictors; therefore, the functions can capture a nonlinear relation between a predictor and the response variable. Because the contributions of individual shape functions to the prediction (classification score for classification and response value for regression) are well separated, the model is easy to interpret.

You can fit a model that includes both linear terms for predictors and pairwise interaction terms for predictors. Alternatively, you can fit a model with linear terms first and then add interaction terms to the model by using the `addInteractions` function. When you use the fitted model, you can specify whether to include interaction terms for analysis.

`fitcgam` and `fitrgam` return:

- `ClassificationGAM` and `RegressionGAM` model objects, respectively, by default
- `ClassificationPartitionedGAM` and `RegressionPartitionedGAM` model objects, respectively, when you specify to cross-validate

You can predict the response of a new observation by using the `predict` function, and plot the effect of each shape function on the prediction for a query point by using the `plotLocalEffects` function. For all object functions and properties of the new objects, see the object reference pages.

Interpretability: Compute Shapley values for features of a machine learning model using shapley

Create a `shapley` object for a machine learning model with a specified query point by using the `shapley` function. The software computes Shapley values of all features in the model for the query point. A Shapley value of a feature explains the deviation of the prediction for the query point from the average prediction due to the feature.

- Use the `plot` function to visualize Shapley values in a bar graph.
- Use the `fit` function to compute Shapley values for another query point.

You can specify the Shapley value computation options by using name-value arguments of `shapley` and `fit`.

- `'Method'` — Specify the Shapley value algorithm to use by setting the `'Method'` name-value argument. `shapley` supports the kernelSHAP algorithm [88] (`'Method','interventional-kernel'`) and the extension to the kernelSHAP algorithm [1] (`'Method','conditional-kernel'`).
- `'UseParallel'` — Specify `'UseParallel',true` to compute Shapley values in parallel. This option requires Parallel Computing Toolbox.

For the full list of supported options, see the object and function reference pages.

Feature Engineering: Automatically create new features before training a classification model

The `gencfeatures` function enables you to automate the feature engineering process in the context of a machine learning workflow. Before passing tabular training data to a classifier, you can create new features from the predictors in the data by using `gencfeatures`. Use the returned data to train the classifier.

`gencfeatures` generates new features based on the `TargetLearner` type, either `'linear'` or `'bag'`. That is, the function creates and selects new features assuming that they will be used to train a linear model or a bagged ensemble model.

- To generate features for an interpretable model, specify `'TargetLearner','linear'` in the call to `gencfeatures`. You can then use the returned data to train a linear classifier. For example, you can use `fitclinear`.
- To generate features that can lead to better model accuracy, specify `'TargetLearner','bag'` in the call to `gencfeatures`. You can then use the returned data to train a bagged ensemble classifier. For example, you can use `fitcensemble` with `'Method','bag'`.

To better understand the generated features, use the `describe` function of the returned `FeatureTransformer` object. To apply the same training set feature transformations to a test or validation set, use the `transform` function of the `FeatureTransformer` object.

Incremental Learning: Train multiclass a naive Bayes model on incoming observations from streaming data, and assess performance in real time

Statistics and Machine Learning Toolbox enables you to create a naive Bayes incremental learner (`incrementalClassificationNaiveBayes`) for multiclass classification in two ways:

- Convert a trained, full naive Bayes model (`ClassificationNaiveBayes`), returned by `fitcnb`, to an incremental learner by passing it to the `incrementalLearner` function. Properties of the converted model reflect the knowledge gained from processing the data.
- Before you fit the model to data, prepare the model for incremental learning by calling `incrementalClassificationNaiveBayes` directly.

Regardless of how you create the model, you can train it, assess its performance, and predict responses as the model accesses data, either per individual observation or specified batch size, in real time.

- The `fit` function fits the model by updating the posterior class distribution given an incoming batch of data, and choosing the class that maximizes the fitted posterior.
- The `updateMetrics` function evaluates the performance of the model as it processes incoming observations. The function writes specified metrics, measured cumulatively and within a specified window of processed observations, to the `Metrics` model property.
- The `updateMetricsAndFit` function first evaluates the performance of the model by calling `updateMetrics` on incoming data, and then fits the model to that data by calling `fit`.
- The `predict` function predicts responses given incoming predictor data.
- The `loss` function returns the classification or regression loss given incoming predictor and response data. Unlike `updateMetrics`, `loss` does not write the computed loss to the model.
- The `logp` function returns the log unconditional probability density of incoming predictor data marginalized over all classes.

Boosted Ensembles: Improved performance fitting classification and regression decision trees

The `fitcensemble` function has improved performance when you use `'LogitBoost'`, `'AdaBoostM1'`, `'AdaBoostM2'`, or `'GentleBooost'` as the ensemble method and decision tree as the learner. The `fitrensemble` function has improved performance when you use `'LSBoost'` as the ensemble method and decision tree as the learner. The default `'Learners'` value is `'tree'` when `'Method'` is any boosting method.

This code for `fitcensemble` is about 3x faster than in the previous release.

```
function timingTest1
% Generate a data set
N = 1e6;
X = [mvnrnd([-1 -1],eye(2),N);...
    mvnrnd([1 1],eye(2),N)];
y = [zeros(N,1); ones(N,1)];

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

The approximate execution times are:

- R2020b: 71.4s
- R2021a: 23.8s

This code for `fitrensemble` is about 2.3x faster than in the previous release.

```
function timingTest2
% Generate a data set
N = 1e6;
X1 = randi([-1,5],[N,1]);
X2 = randi([5,10],[N,1]);
X3 = randi([0,5],[N,1]);
X4 = randi([1,10],[N,1]);
X = [X1 X2 X3 X4];
y = X1 + X2 + X3 + X4 + normrnd(0,1,[N,1]);

tic
mdl = fitrensemble(X,y,'NumBins',50);
toc
```

The approximate execution times are:

- R2020b: 25.4s
- R2021a: 11.1s

These codes were timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the functions `timingTest1` and `timingTest2`.

Parallel Bagged Ensembles: Improved performance fitting regression and classification ensembles (requires Parallel Computing Toolbox)

The `fitcensemble` and `fitrensemble` functions can fit bagged tree ensembles in parallel. The `resume` function of the `ClassificationBaggedEnsemble`, `RegressionBaggedEnsemble`, `ClassificationPartitionedEnsemble`, and `RegressionPartitionedEnsemble` objects also allows parallel fitting. For a documentation example, see Train Classification Ensemble in Parallel.

This code for `fitcensemble` uses parallel computing and is about 1.5x faster than the serial computation in the previous release.

```
function timingTest3
% Generate a data set
N = 1e5;
X = [mvnrnd([-1 -1],eye(2),N);...
    mvnrnd([1 1],eye(2),N)];
y = [zeros(N,1); ones(N,1)];

tic
Mdl2 = fitcensemble(X,y,'Method','Bag',...
    'Options',statset('UseParallel',1));
toc
```

The approximate execution times are:

- R2020b: 29.6s (serial computation)

- R2021a: 19.4s (parallel computation)

This code for `fitrensemble` uses parallel computing and is about 1.5x faster than the serial computation in the previous release.

```
function timingTest4
% Generate a data set
N = 1e5;
X1 = randi([-1,5],[N,1]);
X2 = randi([5,10],[N,1]);
X3 = randi([0,5],[N,1]);
X4 = randi([1,10],[N,1]);
X = [X1 X2 X3 X4];
y = X1 + X2 + X3 + X4 + normrnd(0,1,[N,1]);

tic
mdl = fitrensemble(X,y,'Method','Bag',...
    'Options',statset('UseParallel',1));
toc
```

The approximate execution times are:

- R2020b: 6.2s (serial computation)
- R2021a: 4.1s (parallel computation)

These codes were timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the functions `timingTest3` and `timingTest4`.

Statistics

Cox Proportional Hazards Regression: Fit and analyze lifetime or survival models using object-oriented interface

The `fitcox` function creates a `CoxModel` Cox proportional hazards regression model object for lifetime or survival analysis. The resulting model has properties giving statistics such as a table of model residuals, estimated model coefficients, and their covariances. The model has associated functions for analyzing the fit and predicting survival for new data:

- `coefci` to create confidence intervals
- `hazardratio` to estimate the hazard relative to the baseline
- `linhyptest` to create an ANOVA-type table of statistics
- `plotSurvival` to plot the estimated probability of survival
- `survival` to estimate the survival probability

For an example, see Cox Proportional Hazards Model Object.

GPU Support: gamfit, pca, randsample, and fitcknn now accept gpuArray (requires Parallel Computing Toolbox)

The Statistics and Machine Learning Toolbox functions `gamfit`, `pca`, `randsample`, and `fitcknn` are enhanced to accept `gpuArray` (Parallel Computing Toolbox) input arguments so that they execute on a GPU. `gamfit` and `randsample` fully support GPU arrays. `pca` and `fitcknn` support GPU arrays with some limitations.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

GPU Support: Object functions of ClassificationKNN now support GPU arrays (requires Parallel Computing Toolbox)

The object functions of the `ClassificationKNN` classifier support `gpuArray` (Parallel Computing Toolbox) input arguments in one of the following ways so that they execute on a GPU:

- The object function fully supports GPU arrays and model objects fitted with GPU array input arguments.
- The object function supports model objects fitted with GPU array input arguments.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

gather Function: Enhanced functionality (requires Parallel Computing Toolbox)

The `gather` function gathers all properties of a model fitted using data stored as a GPU array, and returns an output model with properties stored in the local workspace. `gather` has enhanced functionality:

- `gather` can now gather properties of a `ClassificationKNN` object, in addition to `LinearModel`, `CompactLinearModel`, `GeneralizedLinearModel`, and `CompactGeneralizedLinearModel` objects.
- `gather` can now gather properties from multiple models, and return the corresponding output models.

Functionality being removed or changed

statset('coxphfit') Sets TolBnd to []
Behavior change

Starting in R2021a, the `statset` function for `coxphfit` no longer sets a default entry of `1e-6` for the `TolBnd` tolerance. This tolerance, which relates to bounds on variables, never had an effect on `coxphfit` because there are no bounds in `coxphfit`.

Visualization

qqplot Function: Specify target axes for a quantile-quantile plot

You can specify the target axes for a quantile-quantile plot by passing an `Axes` object as the first input argument to the `qqplot` function. For details, see the function reference page.

R2020b

Deployment

SVM Prediction Blocks: Simulate and generate code for support vector machine (SVM) models in Simulink

You can now integrate the `predict` function of `ClassificationSVM` and `RegressionSVM` into Simulink using the ClassificationSVM Predict block and the RegressionSVM Predict block, respectively, instead of using a MATLAB Function block.

- The ClassificationSVM Predict block takes predictor data and returns classified labels and predicted scores (optional).
- The RegressionSVM Predict block takes predictor data and returns predicted responses.

You can import a trained SVM model and configure data types using the block dialog box. The blocks support C/C++ code generation and fixed-point conversion. That is, you can generate C and C++ code using Simulink Coder, and you can design and simulate fixed-point systems using Fixed-Point Designer.

Generate single-precision C/C++ code for the prediction of machine learning models (requires MATLAB Coder)

You can generate single-precision C/C++ code for the prediction of machine learning models for classification and regression. To generate single-precision code, use `saveLearnerForCoder`, `loadLearnerForCoder`, and `codegen`. After training a machine learning model, save the model by using `saveLearnerForCoder`. Define an entry-point function that loads the model by using `loadLearnerForCoder` and calls `predict`. For single-precision code generation, specify the name-value pair argument `'DataType','single'` as an additional input to `loadLearnerForCoder`. Then use `codegen` or the MATLAB Coder app to generate C/C++ code.

Functionality being removed or changed

saveCompactModel and loadCompactModel functions will be removed
Warns

`saveCompactModel` and `loadCompactModel` will be removed in a future release. Use `saveLearnerForCoder` and `loadLearnerForCoder` instead.

`saveLearnerForCoder` and `loadLearnerForCoder` provide broader functionality, including fixed-point code generation for supported models.

This table shows typical usage of `saveCompactModel` and `loadCompactModel` and how to update your code to use `saveLearnerForCoder` and `loadLearnerForCoder`.

Not Recommended	Recommended
`saveCompactModel(Model,'MyModel');`	`saveLearnerForCoder(Model,'MyModel');`
`Mdl = loadCompactModel('MyModel');`	`Mdl = loadLearnerForCoder('MyModel');`

Machine Learning

fitrauto Function: Choose a regression model automatically, across a selection of model types and hyperparameter values

Given predictor and response data, `fitrauto` automatically tries a selection of regression model types with different hyperparameter values. The function uses Bayesian optimization to select models and their hyperparameter values, and computes the following for each model: log(1 + *valLoss*), where *valLoss* is the cross-validation mean squared error (MSE). `fitrauto` returns the optimization results, including the final, chosen model. Use `fitrauto` when you are uncertain which model types best suit your data.

Interpretability: Obtain local interpretable model-agnostic explanations (LIME)

You can explain a prediction of a machine learning model (classification or regression) using LIME. Create a `lime` object for a machine learning model with a specified query point and a specified number of important predictors. The software generates a synthetic data set, and fits a simple interpretable model of important predictors that effectively explains the predictions for the synthetic data around the query point. The simple model can be a linear model (default) or decision tree model.

Use the fitted simple model to explain a prediction of the machine learning model locally, at the specified query point. Use the `plot` function to visualize the LIME results. Based on the local explanations, you can decide whether or not to trust the machine learning model.

Fit a new simple model for another query point by using the `fit` function.

Incremental Learning: Train linear regression or binary classification models on incoming observations from streaming data, and assess performance in real time

An incremental learning, or online learning, algorithm updates a model by training it on a new observation, or a batch of new observations, as soon as the data is available. Optionally, the algorithm can assess the performance of the model before training by treating new data as a test set.

Statistics and Machine Learning Toolbox enables you to create an incremental learner for linear regression (`incrementalRegressionLinear`) or binary classification (`incrementalClassificationLinear`) in two ways:

- Convert a traditionally trained model (one fit to a batch of data) to an incremental learner by passing it to the `incrementalLearner` function. The converted model is warm, meaning its property values reflect the knowledge gained from the traditionally trained model.

  Convertible models include:

  - Linear support vector machine (SVM) for regression, both full and compact (`RegressionSVM` and `CompactRegressionSVM`), see `incrementalLearner`
  - Linear regression model (`RegressionLinear`), see `incrementalLearner`
  - Linear support vector machine (SVM) for binary classification, both full and compact (`ClassificationSVM` and `CompactClassificationSVM`), see `incrementalLearner`

    - Linear classification model (`ClassificationLinear`), see `incrementalLearner`
- If you do not have a supported, traditionally trained model, or you want to prepare an incremental learner to fit to data, call `incrementalRegressionLinear` or `incrementalClassificationLinear` directly. A model created in this way is cold, but you can specify parameters from prior knowledge, such as coefficient values.

Regardless of how you create the model, you can train it, assess its performance, and predict responses as it accesses data, either per individual observation or per specified batch size, in real time.

- The `fit` function fits the model to incoming data by updating the coefficients based on an iteration of stochastic gradient descent (SGD).
- The `updateMetrics` function evaluates the performance of the model as it processes incoming observations. The function writes specified metrics, measured cumulatively and within a specified window of processed observations, to the `Metrics` model property.
- The `updateMetricsAndFit` function first evaluates the performance of the model by calling `updateMetrics` on incoming data, and then fits the model to that data by calling `fit`.
- The `predict` function predicts responses given incoming predictor data.
- The `loss` function returns the classification or regression loss given incoming predictor and response data. Unlike `updateMetrics`, `loss` does not write the computed loss to the model.

Adaptive Scale-Invariant Solver for Incremental Learning

The standard and average stochastic gradient descent (SGD and ASGD) solvers require the regularization and learning rate parameters to be tuned. Also, these parameters are sensitive to the scales of the predictor variables. For traditional machine learning, the data is available, which enables you to tune parameters by applying cross-validation, and to standardize the predictors. For incremental learning, the data might not be available, and the distribution of the predictors is unknown, which makes parameter tuning and predictor standardization difficult or impossible.

Without data or the distribution of the predictors, the losses of the incremental learning models `incrementalClassificationLinear` and `incrementalRegressionLinear` are minimized, by default, through the adaptive scale-invariant solver presented in Kempa et al. [73].

Semi-Supervised Learning: Train machine learning models on partially labeled data using the fitsemigraph and fitsemiself functions

- `fitsemigraph` creates a semi-supervised, graph-based classification model given labeled data, labels, and unlabeled data.
- `fitsemiself` creates a semi-supervised, self-learning classification model given labeled data, labels, and unlabeled data.

The models returned by both functions can predict on unseen data. Each function supports data sets that include any mix of continuous and categorical predictors. Use these functions to create labels for unlabeled data or to train new classification models from both labeled and unlabeled data.

partialDependence Function: Analyze relationships between features and predicted responses through marginalization

The new `partialDependence` function computes partial dependencies of the responses of trained regression models or the scores of trained classification models on their predictors. This function

supports both matrices and tables, accepts new predictor data to compute the partial dependencies, and can be run in parallel.

Categorical and table support for kernel and linear functions

The following functions now support categorical predictors and predictor data in a table:

- `fitckernel`
- `fitclinear`
- `fitcecoc` (with linear or kernel learners)
- `fitcauto` (with linear or kernel learners)
- `fitrkernel`
- `fitrlinear`

The categorical and table support extends to the object functions (such as `predict` and `loss`) of the models created by these functions. The categorical and table support applies to in-memory data only and, therefore, excludes `tall` data.

fitcsvm and fitrsvm Functions: Improved performance with the sequential minimal optimization (SMO) solver

The `fitcsvm` and `fitrsvm` functions show improved performance when you use the SMO solver for an optimization routine. This performance improvement is for all kernel functions.

`fitcsvm` and `fitrsvm` use the SMO solver by default (that is, the default value of the `'Solver'` name-value pair argument is `'SMO'`) in these cases:

- You use `fitcsvm` for one-class learning.
- You use `fitcsvm` for binary classification and the default value `0` for the `'OutlierFraction'` name-value pair argument.
- You use `fitrsvm` and the default value `0` for the `'OutlierFraction'` name-value pair argument.

This code for `fitcsvm` is about 4x faster than in the previous release.

```
function timingTest1
% Generate a data set
N = 2e4;
X = [mvnrnd([-1 -1],eye(2),N); mvnrnd([1 1],eye(2),N)];
y = [zeros(N,1); ones(N,1)];

% Train an SVM model
tic
Mdl = fitcsvm(X,y);
toc
end
```

The approximate execution times are:

- R2020a: 16.0s
- R2020b: 4.1s

This code for `fitrsvm` is about 1.8x faster than in the previous release.

```
function timingTest2
% Load a data set
load carbig
X = [Horsepower Weight];
y = MPG;

% Train an SVM model
tic
Mdl = fitrsvm(X,y);
toc
end
```

The approximate execution times are:

- R2020a: 8.3s
- R2020b: 4.6s

These codes were timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the functions `timingTest1` and `timingTest2`.

fitctree, fitrtree, fitcensemble, and fitrensemble Functions: Improved performance with the binning technique

The `fitctree`, `fitrtree`, `fitcensemble`, and `fitrensemble` functions show improved performance when growing decision trees using the binning technique. The functions use binning if you set the `'NumBins'` name-value pair argument to a positive integer. For example, this code for `fitcensemble` is about 1.3x faster than in the previous release.

```
function timingTest
% Generate a data set
N = 1e4;
D = 100;
mu1 = ones(D,1);
mu2 = zeros(D,1);
sigma = diag(ones(D,1));
X = [mvnrnd(mu1,sigma,N); mvnrnd(mu2,sigma,N)];
y = [ones(N,1); zeros(N,1)];

tic
mdl = fitcensemble(X,y,'Method','Bag','NumBins',50);
toc
end
```

The approximate execution times are:

- R2020a: 4.7s
- R2020b: 3.6s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the function `timingTest`.

lasso Function: Improved performance

The `lasso` function shows improved performance. For example, this code is about 1.3x faster than in the previous release.

```
function timingLasso
% Generate a data set
N = 1e6;
X = randn(N,10);                  % Predictor data
coeff = [0 2 0 -3 0 0 5 0 0 -1]'; % Coefficients with 4 nonzero values
y = X*coeff + randn(N,1)*0.1;     % Response data with small noise

tic
B = lasso(X,y);
toc
end
```

The approximate execution times are:

- R2020a: 4s
- R2020b: 3s

The code was timed on a Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.5 GHz test system by calling the function `timingLasso`.

Stacked Ensemble Example

A new featured example, Combine Heterogeneous Models into Stacked Ensemble, shows how to build multiple machine learning models for a given training data set, and the combine the models using a technique called *stacking* to improve the accuracy on a test data set compared to the accuracy of the individual models.

Statistics

GPU Support: grp2idx, fitlm, fitglm, glmfit, and glmval now accept gpuArray (requires Parallel Computing Toolbox)

These Statistics and Machine Learning Toolbox functions are enhanced to accept `gpuArray` input arguments so that they execute on a GPU. The GPU support extends to all object functions of the models `LinearModel` and `GeneralizedLinearModel` created by the functions `fitlm` and `fitglm`, respectively. This support requires Parallel Computing Toolbox.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

Functionality being removed or changed

nancov, nanmax, nanmean, nanmedian, nanmin, nanstd, nansum, and nanvar functions are not recommended
Still runs

`nancov`, `nanmax`, `nanmean`, `nanmedian`, `nanmin`, `nanstd`, `nansum`, and `nanvar` are not recommended. These functions omit `NaN` values for calculations. Use the MATLAB functions `cov`, `max`, `mean`, `median`, `min`, `std`, `sum`, and `var` instead, respectively. With these functions, you can specify whether to include or omit `NaN` values in calculations. There are no plans to remove the Statistics and Machine Learning Toolbox functions.

The MATLAB functions have several advantages over the Statistics and Machine Learning Toolbox functions.

- The MATLAB functions offer more extended capabilities for supporting tall arrays, GPU arrays, distribution arrays, and C/C++ code generation. `max`, `mean`, `min`, `std`, `sum`, and `var` also support GPU code generation.
- When you specify the `'linear'` option, `max` and `min` return a linear index into the input array corresponding to the maximum and minimum value, respectively.
- `mean` and `sum` return an output value with a specified data type.

You can update your code as follows:

- `nanmax` and `nanmin` — Change instances of the function names `nanmax` and `nanmin` to `max` and `min`, respectively. You do not need to change the input arguments.
- `nanmean`, `nanmedian`, `nanstd`, `nansum`, and `nanvar` — Change instances of these function names to `mean`, `median`, `std`, `sum`, and `var`, respectively. Then specify the `'omitnan'` option for the `nanflag` input argument.
- `nancov` — Change instances of the function name `nancov` to `cov`. Then specify the `'omitrows'` option for the `nanflag` input argument. The `'pairwise'` option of `nancov` corresponds to the `'partialrows'` option of `cov`.

combnk is not recommended
Still runs

`combnk` is not recommended. Use the MATLAB function `nchoosek` instead. There are no plans to remove `combnk`.

To update your code, change instances of the function name `combnk` to `nchoosek`. You do not need to change the input arguments. For example, use `C = nchoosek(v,k)`. The output `C` contains all possible combinations of the elements of vector `v` taken `k` at a time. Note that `C` from `nchoosek` can have a different order compared to the output from `combnk`.

The `nchoosek` function has several advantages over the `combnk` function.

- `nchoosek` also returns the binomial coefficient when the first input argument is a scalar.
- `nchoosek` has extended functionality using MATLAB Coder.
- `nchoosek` is faster than `combnk`.

Unused categories in census1994 have been removed
Behavior change

The `census1994` data set contains the tables `adultdata` and `adulttest`, which have the same set of variables, including the categorical variable `salary`. The values in `salary` are `'<=50K'` or `'>50K'`. However, `salary` in `adulttest` had two unused categories, each with a period in its name: `'<=50K.'` and `'>50K.'`. The unused categories have been removed and the categories have been reordered so that the `salary` variables in `adultdata` and `adulttest` have the same categories in the same order.

Visualization

plotPartialDependence Function: Plot partial dependencies for classification models

The `plotPartialDependence` function shows relationships between selected features and predicted responses for a trained model. This function now supports trained classification models in addition to the previously supported regression models. You can create a partial dependence plot (PDP), individual conditional expectation (ICE) plot, and centered ICE plot. You can also provide additional predictor data to plot and evaluate.

R2020a

Deployment

Generate fixed-point C/C++ code for the prediction of a decision tree model and an ensemble of decision trees (requires MATLAB Coder and Fixed-Point Designer)

You can generate fixed-point C/C++ code for the prediction of a decision tree model and an ensemble of decision trees for classification and regression. To generate fixed-point code, create a structure that defines fixed-point data types by using generateLearnerDataTypeFcn. Then use the structure as an input argument of loadLearnerForCoder in an entry-point function.

Specify score transform of SVM classifiers for fixed-point C/C++ code generation (requires MATLAB Coder and Fixed-Point Designer)

You can generate fixed-point C/C++ code for the prediction of a support vector machine (SVM) classifier that uses one of these score transforms: a built-in function other than `'invlogit'` or a score transformation for fitting posterior probabilities (generated by using `fitPosterior` or `fitSVMPosterior`). For the list of available built-in functions, see `ScoreTransform`. For more information on fitting posterior probabilities for code generation, see Code Generation.

When you train an SVM classifier with a score transform other than `'none'` or `'identity'`, the `generateLearnerDataTypeFcn` function generates a data type function whose output structure `T` contains two fields related to the scores: `ScoreDataType` and `TransformedScoreDataType`. Use these fields to influence the precision of the output scores before and after their transformation, respectively. For more details, see Data Type Function.

Compatibility Considerations

In R2019b, you could train an SVM classifier for fixed-point code generation with some nondefault score transforms (`'ismax'`, `'sign'`, `'symmetric'`, or `'symmetricismax'`). The `generateLearnerDataTypeFcn` function generated a data type function whose output structure contained only one field related to the scores, `ScoreDataType`. This field influenced the precision of the output scores both before and after their transformation.

Use numeric predictors in a table to generate C/C++ code for prediction (requires MATLAB Coder)

When you train a classification or regression model by using numeric predictor variables in a table, you can pass a table with the same variables to `predict` or `random` inside your entry-point function. For an example, see Generate Code to Classify Numeric Data in Table.

kmeans, knnsearch, pdist2, and rangesearch Functions: Return integer-type indices in generated standalone C/C++ code (requires MATLAB Coder)

The `kmeans`, `knnsearch`, `pdist2`, and `rangesearch` functions now return integer-type indices in generated standalone C/C++ code to allow for stricter single-precision support. For MEX code generation, the functions still return double-precision indices to match the MATLAB behavior.

This new behavior also applies to the `knnsearch` and `rangesearch` object functions of nearest neighbor searcher objects.

Compatibility Considerations

In previous releases, these functions returned double-precision indices in all generated code.

Apps

Machine Learning Apps: Import predictor data and the response variable as separate variables

When you select data from the MATLAB workspace to import into Classification Learner or Regression Learner, you can choose separate variables for the predictor data and the response variable.

In Classification Learner or Regression Learner, on the **Classification Learner** or **Regression Learner** tab, click **New Session > From Workspace** in the **File** section. In the New Session dialog box, under **Data Set Variable**, select the predictor data from the list of workspace variables. Under **Response**, click the **From workspace** option button and then select the response variable from the list. For more details, see Select Data from Workspace (Classification Learner) and Select Data from Workspace (Regression Learner).

Classification Learner: Assess model performance with the updated confusion matrix and parallel coordinates plot

The confusion matrix and parallel coordinates plot in Classification Learner now use the `confusionchart` and `parallelplot` functions, respectively.

Machine Learning

fitcauto Function: Choose a classification model automatically, across a selection of classifier types and hyperparameter values

Given predictor and response data, `fitcauto` automatically tries a selection of classification model types with different hyperparameter values. The function uses Bayesian optimization to select models and their hyperparameter values, and computes the cross-validation classification error for each model. `fitcauto` returns the optimization results, including the final, chosen model. Use `fitcauto` when you are uncertain which classifier types best suit your data.

Feature Selection: Rank features using fscchi2 and fsrftest

- `fscchi2` ranks features using chi-square tests for classification problems.
- `fsrftest` ranks features using *F*-tests for regression problems.

Both functions support a data set that includes both continuous and categorical features. Each function returns the indices of features ordered by feature importance, and the score for each feature. A large score value indicates that the corresponding feature is important. You can use the outputs for feature selection.

fscmrmr Function: Specify 'UseMissing',true to use missing values in predictors for ranking

You can now specify whether to use or discard missing values in predictors for ranking by using the `'UseMissing'` name-value pair argument of `fscmrmr`. The default value of `'UseMissing'` is `false` because most classification training functions in Statistics and Machine Learning Toolbox do not use missing values for training. The new feature selection functions `fscchi2` and `fsrftest` also support `'UseMissing'`.

Compatibility Considerations

In R2019b, `fscmrmr` used missing values in predictors by default. To update your code, specify `'UseMissing',true`.

lasso Function: Specify whether to omit the intercept term from the model

In the call to `lasso`, specify `'Intercept',false` to omit the intercept term from the model.

Statistics

GPU Support: corr, random, and 32 probability distribution functions now accept gpuArray (requires Parallel Computing Toolbox)

The following Statistics and Machine Learning Toolbox functions are enhanced to accept `gpuArray` input arguments so that they execute on the GPU. This support requires Parallel Computing Toolbox.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Function List (GPU Arrays).

`betalike`	`evlike`	`hygecdf`	`random`
`betarnd`	`evrnd`	`hygeinv`	`trnd`
`binocdf`	`fpdf`	`hygepdf`	`tiedrank`
`binopdf`	`frnd`	`hygernd`	`unifcdf`
`chi2pdf`	`gampdf`	`nbinrnd`	`unifpdf`
`chi2rnd`	`gamrnd`	`ncfrnd`	`wblfit`
`corr`	`geornd`	`nctrnd`	`wblrnd`
`ecdf`	`gevrnd`	`ncx2rnd`	
`evfit`	`gprnd`	`poissrnd`	

Visualization

Specify target axes for visualization functions

You can now specify target axes for plots created by these functions:

- `addedvarplot`
- `andrewsplot`
- `biplot`
- `histfit`
- `refcurve`
- `plot`, `plotAdded`, and `plotResiduals` of `LinearModel`

Specify an `Axes` object as the first input argument of the function. For details, see the function reference pages.

R2019b

Big Data

Find pairwise distances, all neighbors within a specified distance, and nearest neighbors for input data in a tall array

- Use `pdist2` to find the pairwise distances between input data in a tall array `X` and an in-memory array `Y`.
- Find *k*-nearest neighbors for input data in a tall array by using `knnsearch`.
- Find all neighbors within a specified distance for input data in a tall array by using `rangesearch`.

Generalize exact tall prctile to allow multiple percentiles

`prctile` and `quantile` now allow simultaneous calculation of multiple exact percentiles and quantiles, respectively, using a sorting-based algorithm.

Deployment

Update a deployed tree or linear model without regenerating code (requires MATLAB Coder)

After training a model, use the `learnerCoderConfigurer` function to create a coder configurer object. A coder configurer offers convenient features to configure code generation options, generate C/C++ code, and update model parameters in the generated code.

- Configure code generation options and specify the coder attributes of model parameters by using object properties.
- Generate C/C++ code for the `predict` and `update` functions of the model by using `generateCode`. This step requires MATLAB Coder.
- Update model parameters in the generated C/C++ code without having to regenerate the code. This feature reduces the effort required to regenerate, redeploy, and reverify C/C++ code when you retrain the model with new data or settings. Before updating model parameters, use `validatedUpdateInputs` to validate and extract the model parameters to update.

This workflow now supports the models listed in this table.

Model	Coder Configurer Object
Binary decision tree for multiclass classification	`ClassificationTreeCoderConfigurer`
Linear model for binary classification	`ClassificationLinearCoderConfigurer`
Multiclass model with linear binary learners	`ClassificationECOCCoderConfigurer`
Binary decision tree for regression	`RegressionTreeCoderConfigurer`
Linear regression	`RegressionLinearCoderConfigurer`

Generate C/C++ code for probability distribution functions (requires MATLAB Coder)

You can generate C/C++ code that fits a probability distribution to sample data and evaluates the fitted distribution object. These probability distribution functions support code generation:

- `fitdist` — Fit a probability distribution object to sample data for beta, exponential, extreme value, lognormal, normal, and Weibull distributions.
- Probability distribution object functions (`cdf`, `icdf`, `iqr`, `mean`, `median`, `pdf`,`std`, `truncate`, `var`) — Use the probability distribution object functions to evaluate the fitted distribution object created by `fitdist`.
- `evfit`, `expfit`, `lognfit`, `normfit`, `wblfit` — Estimate probability distribution parameters using distribution-specific functions.

For an example that generates C/C++ code using a probability distribution object, see Code Generation for Probability Distribution Objects.

Generate fixed-point C/C++ code for the prediction of an SVM model (requires MATLAB Coder and Fixed-Point Designer)

You can generate fixed-point C/C++ code for the prediction of a support vector machine (SVM) classification or regression model. To generate fixed-point code, create a structure that defines fixed-point data types by using `generateLearnerDataTypeFcn`. Then use the structure as an input argument of `loadLearnerForCoder` in an entry-point function. For an example, see Fixed-Point Code Generation for Prediction of SVM.

saveLearnerForCoder and loadLearnerForCoder Functions: Save and load machine learning models for code generation

`saveLearnerForCoder` prepares a machine learning model for code generation and saves it as a MAT-file. `loadLearnerForCoder` then reconstructs the saved model.

Additionally, `loadLearnerForCoder` supports the fixed-point code generation workflow for support vector machine (SVM) models.

The `saveLearnerForCoder` and `loadLearnerForCoder` functions replace the `saveCompactModel` and `loadCompactModel` functions.

Functionality being removed or changed

saveCompactModel and loadCompactModel functions will be removed
Still runs

`saveCompactModel` and `loadCompactModel` will be removed in a future release. Use `saveLearnerForCoder` and `loadLearnerForCoder` instead.

`saveLearnerForCoder` and `loadLearnerForCoder` provide broader functionality, including fixed-point code generation for supported models.

This table shows typical usage of `saveCompactModel` and `loadCompactModel` and how to update your code to use `saveLearnerForCoder` and `loadLearnerForCoder`.

Not Recommended	Recommended
`saveCompactModel(Model,'MyModel');`	`saveLearnerForCoder(Model,'MyModel');`
`Mdl = loadCompactModel('MyModel');`	`Mdl = loadLearnerForCoder('MyModel');`

Apps

Machine Learning Apps: Optimize hyperparameters of machine learning models in Classification Learner and Regression Learner

After you choose a machine learning model type, you can select the hyperparameters of the model that you want to tune using hyperparameter optimization.

- On the **Classification Learner** or **Regression Learner** tab, in the **Model Type** section, click the arrow to open the gallery. Click the optimizable model of your choice.
- In the **Model Type** section, select **Advanced > Advanced**. In the dialog box, select the **Optimize** check boxes for the hyperparameters that you want to optimize. Under **Values**, select the fixed values for the hyperparameters that you do not want to optimize.
- (Optional) In the **Model Type** section, select **Advanced > Optimizer Options**. Select options to specify how the optimization is performed.

After selecting options, train the optimizable model. The app tries different combinations of hyperparameter values based on your selections and returns a model with the optimized hyperparameter values. You cannot use the **Use Parallel** button to run the optimization in parallel.

When the optimizable model stops training, the **Current Model** pane in the bottom left of the app shows the optimized hyperparameter values. The exported optimizable model and generated MATLAB code treat these values as fixed model hyperparameters.

For details on hyperparameter optimization in Classification Learner, see Hyperparameter Optimization in Classification Learner App and Train Classifier Using Hyperparameter Optimization in Classification Learner App. For details on hyperparameter optimization in Regression Learner, see Hyperparameter Optimization in Regression Learner App and Train Regression Model Using Hyperparameter Optimization in Regression Learner App.

Classification Learner: Specify misclassification costs when training classifiers

Before training any classification models, specify the costs associated with misclassifying the observations of one class into another. On the **Classification Learner** tab, in the **Options** section, click **Misclassification Costs**. In the dialog box, specify the misclassification costs directly, or import the costs from a workspace variable. For more details, see Misclassification Costs in Classification Learner App and Train and Compare Classifiers Using Misclassification Costs in Classification Learner App.

Machine Learning Apps: Explore data in app plots by using the axes toolbar

Plots in Classification Learner and Regression Learner have an axes toolbar that appears above the top right of the plot for quick access to data exploration tools. The buttons available on the toolbar depend on the contents of the plot. The toolbar can include buttons to export the plot as an image, add data tips, pan or zoom the data, and restore the view.

Machine Learning

Spectral Clustering: Perform spectral clustering using spectralcluster

Use the `spectralcluster` function to implement spectral clustering on data or a similarity matrix derived from the data. `spectralcluster` requires you to specify the number of clusters `k` but also enables you to estimate the correct number of clusters in your data. The example Estimate Number of Clusters and Perform Spectral Clustering shows how to estimate the number of clusters in the data and perform spectral clustering on the data and on the similarity matrix derived from the data.

Feature Selection: Rank features using fscmrmr and fsulaplacian

- `fscmrmr` ranks features using the minimum redundancy maximum relevance (MRMR) algorithm for classification problems. The function can be faster than other feature ranking functions in Statistics and Machine Learning Toolbox. Also, the function supports a data set that includes both continuous and categorical features.
- `fsulaplacian` ranks features using Laplacian scores for unsupervised learning problems.

Both functions return the indices of features ordered by feature importance, and the score for each feature. A large score value indicates that the corresponding feature is important. You can use the outputs for feature selection.

Model selection example using Bayesian optimization

A new featured example, Moving Towards Automating Model Selection Using Bayesian Optimization, shows how you can use Bayesian optimization to tune hyperparameters for several different models, select models that perform well on the first few iterations of Bayesian optimization, and continue to tune hyperparameters of the selected models.

Statistics

nearcorr Function: Compute the nearest correlation matrix by minimizing the Frobenius distance

Given a nonpositive semidefinite matrix, use `nearcorr` to compute the nearest correlation matrix by minimizing the Frobenius distance.

Visualization

gscatter Function: Specify axes for a scatter plot of grouped data

You can now specify the axes for a scatter plot of grouped data by passing an `Axes` or `UIAxes` object to the `gscatter` function. For details, see the function reference page.

R2019a

Big Data

Perform binary regression using decision trees on tall arrays, and optimize hyperparameters for binary regression decision trees and multiclass classification models

- Use `fitrtree` to perform binary regression with decision trees on data in tall arrays, and specify the maximum depth of the output tree by using the `'MaxDepth'` name-value pair argument.
- Perform hyperparameter optimization on tall arrays by using the `'OptimizeHyperparameters'` and `'HyperparameterOptimizationOptions'` name-value pair arguments in the calls to `fitrtree` and `fitcecoc`.

Specify cost, prior, and weights for bagged decision trees

`TreeBagger` supports cost, prior, and weight values for tall arrays. Use the `'Cost'`, `'Prior'`, and `'Weights'` name-value pair arguments in the call to the function.

The `error` method of the `CompactTreeBagger` object supports the `'Weights'` name-value pair argument for tall arrays.

Specify the number of times to repeat k-means clustering for new initial cluster centroid positions, control the level of output to display, and set the maximum number of iterations

- Use the `'Replicates'` name-value pair argument to specify the number of times to repeat *k*-means clustering on out-of-memory data for new initial cluster centroid positions. The `'Start'` name-value pair argument now accepts a numeric array, and `kmeans` infers a value for `'Replicates'` from the third dimension of the array.
- Control the level of output to display by using the `'Display'` name-value pair argument.
- Set the maximum number of iterations by using the `'MaxIter'` name-value pair argument.

Create a confusion matrix chart for tall arrays using confusionchart

`confusionchart` creates a confusion matrix chart for the tall arrays of true labels and predicted labels. You can also pass a tall array of a confusion matrix to plot the confusion matrix chart.

Operate on multiple dimensions of tall arrays simultaneously for selected functions

The following functions now accept a vector dimension argument to specify multiple operating dimensions simultaneously, as well as the option `'all'` to specify all dimensions of a tall array.

`geomean`	`quantile`
`harmmean`	`range`
`kurtosis`	`skewness`

prctile	zscore

Deployment

Update a deployed multiclass SVM model without regenerating code (requires MATLAB Coder)

After training an error-correcting output codes (ECOC) multiclass model that uses binary support vector machine (SVM) learners, use the `learnerCoderConfigurer` function to create a coder configurer object, `ClassificationECOCCoderConfigurer`. A coder configurer offers convenient features to configure code generation options, generate C/C++ code, and update model parameters in the generated code.

- Configure code generation options and specify the coder attributes of model parameters using object properties.
- Generate C/C++ code for the `predict` and `update` functions of the model by using `generateCode`. This step requires MATLAB Coder.
- Update model parameters in the generated C/C++ code without having to regenerate the code. This feature reduces the effort required to regenerate, redeploy, and reverify C/C++ code when you retrain the model with new data or settings. Before updating model parameters, use `validatedUpdateInputs` to validate and extract the model parameters to update.

Generate C/C++ code for prediction using naive Bayes classification models, prediction using multiclass ECOC models with a custom binary loss function, and for kernel density estimates (requires MATLAB Coder)

- You can generate C/C++ code that predicts responses by using trained naive Bayes models.
  - `predict` — Classify observations, estimate posterior probabilities, or compute misclassification costs by applying a naive Bayes classification model to new data.
- You can specify a custom binary loss function for the `predict` function of `CompactClassificationECOC`.
- These functions support code generation:
  - `ksdensity` — Find kernel smoothing function estimates for univariate and bivariate data.
  - `mvksdensity` — Find kernel smoothing function estimates for multivariate data.
  - `ecdf` — Estimate empirical cumulative distribution function values.

Generate optimized CUDA code for pdist and pdist2 (requires GPU Coder)

You can generate optimized CUDA® code for `pdist` and `pdist2` using GPU Coder™. The supported distance input argument values (`Distance`) for optimized CUDA code are `'euclidean'`, `'squaredeuclidean'`, `'seuclidean'`, `'cityblock'`, `'minkowski'`, `'chebychev'`, `'cosine'`, `'correlation'`, `'hamming'`, and `'jaccard'`. For more information on GPU coder, see Getting Started with GPU Coder (GPU Coder) and Supported Functions (GPU Coder).

Generate an optimized MEX function for the kd-tree search algorithm (requires MATLAB Coder)

You can generate an optimized MEX function using Intel Threading Building Blocks (TBB) for parallel computation on multicore platforms when you use the *k*d-tree search algorithm for these functions:

- `knnsearch` and `rangesearch` functions
- `knnsearch` and `rangesearch` functions of `KDTreeSearcher`
- `predict` function of `ClassificationKNN`

For details, see the Code Generation sections of the function reference pages.

Compatibility Considerations

Starting in R2019a, `codegen` generates code using Intel TBB for parallel computation if these functions use the *k*d-tree search algorithm and the code generation build type is a MEX function. Otherwise, `codegen` generates code using `parfor`. If you generate the MEX function to test the generated code of the `parfor` version, you can disable the usage of Intel TBB. Set the `ExtrinsicCalls` property of the MEX configuration object to `false`.

Deploy models exported from machine learning apps easily (requires MATLAB Compiler)

You can now deploy models exported from Classification Learner or Regression Learner by specifying a single pragma in the function to be deployed. For more information, see Deploy Predictions from Classification Learner or Deploy Predictions from Regression Learner.

Apps

Machine Learning Apps: Export plots to figures and generate MATLAB code to train a model with new data in Classification Learner and Regression Learner

- After you create plots in Classification Learner or Regression Learner, you can export them to MATLAB figures. On the **Classification Learner** or **Regression Learner** tab within the app, in the **Export** section, click **Export Plot to Figure**. You can then copy, save, or customize the new figures. For details, see Export Plots in Classification Learner App or Export Plots in Regression Learner App.
- After you train a model in Classification Learner or Regression Learner, you can generate MATLAB code to train the model with new data. On the **Classification Learner** or **Regression Learner** tab within the app, in the **Export** section, click **Generate Function**. The generated function allows you to pass in new training data; the function then returns a new trained model and the accuracy of that model on validation data. The **Generate Function** button replaces the previous **Export Model > Generate Code** option. For details, see Generate MATLAB Code in Classification Learner or Generate MATLAB Code in Regression Learner.

Classification Learner: Train naive Bayes models to classify data

You can train a Gaussian naive Bayes model or a kernel naive Bayes model individually or train the two models simultaneously using Classification Learner. You can also specify kernel properties for the kernel naive Bayes model. For details, see Train Naive Bayes Classifiers Using Classification Learner App.

Machine Learning

Density-Based Clustering: Perform density-based spatial clustering of applications with noise (DBSCAN) using dbscan

Use the `dbscan` function to identify arbitrarily shaped clusters and noise in data. `dbscan` performs clustering on observations or pairwise distances between observations, and supports the use of custom distance metrics for clustering. The example Determine Values for DBSCAN Parameters shows how to determine values for the `minpts` and `epsilon` arguments of `dbscan`.

fitcecoc Function: Perform hyperparameter optimization while using kernel binary learners

You can perform hyperparameter optimization while training an ECOC model that uses kernel binary learners. Specify the `'Learners'` name-value pair argument as `'kernel'` or a `templateKernel` object, and use the `'OptimizeHyperparameters'` and `'HyperparameterOptimizationOptions'` name-value pair arguments in the call to `fitcecoc`.

Accelerate training tree-based classification and regression models by binning predictor values

You can reduce the training time of a tree-based model by using the binning option. Specify the number of bins by using the `'NumBins'` name-value pair argument when you train a classification model using `fitctree`, `fitcensemble`, and `fitcecoc` or a regression model using `fitrtree` and `fitrensemble`. The software groups every numeric predictor into a specified number of bins and then grows trees on the bin indices instead of the original data.

For an example, see Speed Up Training by Binning Numeric Predictor Values. This example shows how to reduce training time by specifying `'NumBins'` when you use a boosting algorithm with tree learners.

Compatibility Considerations

Due to software changes for this new feature, the software requires an additional argument to reproduce results. To reproduce random predictor selections when `'NumVariablesToSample'` is not `'all'`, set the seed of the random number generator by using `rng` and specify the value `true` for the `'Reproducible'` name-value pair argument of `fitctree`, `fitrtree`, or `templateTree`. If you train a model by using `fitcecoc`, `fitcensemble`, or `fitrensemble`, specify `'Reproducible',true` when you create a tree template using `templateTree`. The default value of `'NumVariablesToSample'` is not `'all'` when you specify a random forest algorithm (`'Method','Bag'`) using `fitcensemble` or `fitrensemble`.

Performance enhancements for knnsearch and rangesearch functions

- The algorithms for these functions show faster performance:
  - `rangesearch` object function of `KDTreeSearcher` and `ExhaustiveSearcher`
  - `knnsearch` object function of `KDTreeSearcher` when `'IncludeTies'` is set to `true`

- When using the `knnsearch` and `rangesearch` functions or the `knnsearch` and `rangesearch` object functions, you can set the new `'SortIndices'` name-value pair argument to `false` for faster performance. For details, see the function and object function reference pages.

Functionality Being Removed or Changed

classregtree has been removed

The `classregtree` function and the `classregtree` object have been removed. Instead, use the `fitctree` function and the `ClassificationTree` object for classification, and use the `fitrtree` function and the `RegressionTree` object for regression.

`ClassificationTree` and `RegressionTree` were introduced in R2011a as new objects for classification and regression trees. You can create these objects by using `fitctree` and `fitrtree`, respectively. These functions and objects provide all the functionality of `classregtree` and more, including:

- Cross-validation
- Hyperparameter optimization
- Resubstitution statistics
- More algorithms for splits on categorical variables for multiclass problems
- Tall array support
- Code generation for prediction

To update your code, replace all instances of `classregtree` with `fitctree` or `fitrtree`. This table shows the name-value pair arguments of `classregtree` and the corresponding arguments of `fitctree` and `fitrtree`.

Name-Value Pair Arguments of `classregtree`	Name-Value Pair Arguments of `fitctree` and `fitrtree`
`'categorical'`	`'CategoricalPredictors'`
`'method'`	Instead of specifying `'method'` as `'classification'` or `'regression'`, use `fitctree` for classification or `fitrtree` for regression
`'names'`	`'PredictorNames'`
`'prune'`	`'Prune'`
`'minparent'`	`'MinParentSize'`
`'minleaf'`	`'MinLeafSize'`
`'mergeleaves'`	`'MergeLeaves'`
`'nvartosample'`	`'NumVariablesToSample'`
`'stream'`	Use `rng` to control random number stream
`'surrogate'`	`'Surrogate'`
`'weights'`	`'Weights'`
`'qetoler'` for regression trees only	`'QuadraticErrorTolerance'` of `fitrtree`

Name-Value Pair Arguments of `classregtree`	Name-Value Pair Arguments of `fitctree` and `fitrtree`
`'cost'` for classification trees only	`'Cost'` of `fitctree`
`'splitcriterion'` for classification trees only	`'SplitCriterion'` of `fitctree`
`'priorprob'` for classification trees only	`'Prior'` of `fitctree`

This table shows the object functions of `classregtree` and the corresponding properties and object functions of `ClassificationTree` and `RegressionTree`.

Object Functions of `classregtree`	Properties and Object Functions of `ClassificationTree` and `RegressionTree`
`catsplit`	`CategoricalSplit` property
`children`	`Children` property
`classcount`	`ClassCount` property
`classname`	`ClassNames` property
`classprob`	`ClassProbability` property
`cutcategories`	`CutCategories` property
`cutpoint`	`CutPoint` property
`cuttype`	`CutType` property
`cutvar`	`CutPredictor` property
`eval`	`predict` and `resubPredict` functions
`isbranch`	`IsBranchNode` property
`meansurrvarassoc`	`surrogateAssociation` function
`nodeclass`	`NodeClass` property
`nodeerr`	`NodeError` property
`parent`	`Parent` property
`prune`	`prune` function
`prunelist`	`PruneList` property
`risk`	`NodeRisk` property
`surrcutcategories`	`SurrogateCutCategories` property
`surrcutflip`	`SurrogateCutFlip` property
`surrcutpoint`	`SurrogateCutPoint` property
`surrcuttype`	`SurrogateCutType` property
`surrcutvar`	`SurrogateCutPredictor` property
`surrvarassoc`	`SurrogatePredictorAssociation` property
`test`	`loss` and `cvloss` functions
`type`	`ClassificationTree` for classification and `RegressionTree` for regression
`varimportance`	`predictorImportance` function
`view`	`view` function

svmtrain and svmclassify have been removed

The `svmtrain` and `svmclassify` functions have been removed. Use the `fitcsvm` function and the `predict` function of `ClassificationSVM` instead.

To update your code, replace all instances of `svmtrain` with `fitcsvm`, and replace all instances of `svmclassify` with `predict`. For more details, see the Compatibility Considerations sections of `svmtrain` and `svmclassify`.

Statistics

sobolset Object: Access subsets of the dimensions of a scrambled Sobol point set

You can access subsets of the dimensions of a scrambled `sobolset` object by using the new `reduceDimensions` object function.

geomean and harmmean Functions: Specify whether to include or omit NaN values

In the call to `geomean` and `harmmean`, you can use the `'nanflag'` argument to specify whether to include or omit `NaN` values in the calculations.

Functionality Being Removed or Changed

princomp has been removed

The `princomp` function has been removed. Use the `pca` function instead.

The `pca` function was introduced in R2012b as a new way to perform principal component analysis. The `pca` function provides the functionality of `princomp` and the following additional features:

- Handling of `NaN` as missing data values
- Weighted principal component analysis with user-specified weights
- Choice of the SVD or EIG algorithm for computing principal components
- Option to specify the number of components to return
- Option to not center data before computing principal components

To update your code, replace all instances of `princomp` with `pca`. This table shows some typical usages of `princomp` and how to update your code to use `pca` instead.

Input Data X	Removed Functionality	Recommended Replacement
Full rank matrix	`[coeff,score,latent,tsquare] = princomp(X)`	`[coeff,score,latent,tsquare] = pca(X)`
Rank deficient matrix	`[coeff,score,latent,tsquare] = princomp(X)`	`[coeff,score,latent,tsquare] = pca(X,'Economy',false)`
Rank deficient matrix	`[coeff,score,latent,tsquare] = princomp(X,'econ')`	`[coeff,score,latent,tsquare] = pca(X)`

Visualization

sortClasses Function: Sort the classes on a confusion matrix chart to cluster similar classes for ease of interpretation

After creating a confusion matrix chart by using the `confusionchart` function, use the `sortClasses` function to sort the classes of the confusion matrix. Specify the input argument `order` as `'cluster'` in the call to `sortClasses` to cluster similar classes in the confusion matrix chart.

R2018b

Version: 11.4

New Features

Bug Fixes

Big Data Algorithms: Fit multiclass classification models, perform hyperparameter optimization, specify cost and priors when fitting classification models, compute approximate quantiles, and expand categorical variables into dummy variables on out-of-memory data

These functions support tall arrays:

- `fitcecoc` performs multiclass classification on data in tall arrays by using linear or kernel models as the binary classifiers.
- `prctile` and `quantile` return the percentiles and quantiles of a tall data set, respectively, as either exact or approximate values. Set the `'Method'` name-value pair argument to either `'exact'` or `'approximate'`.
- `dummyvar` converts categorical variables into numeric dummy variables.

These functions support hyperparameter optimization for tall arrays: `fitcdiscr`, `fitckernel`, `fitclinear`, `fitctree`, `fitrkernel`, and `fitrlinear`. Use the `'OptimizeHyperparameters'` and `'HyperparameterOptimizationOptions'` name-value pair arguments in the call to the function.

`fitcnb` and `fitctree` support cost and prior values for tall arrays. Use the `'Cost'` and `'Prior'` name-value pair arguments in the call to the function.

For a complete list of Statistics and Machine Learning Toolbox functions that support tall arrays, see Tall Array Support, Usage Notes, and Limitations.

Code Generation: Update a deployed SVM model without regenerating code (requires MATLAB Coder)

Generate C/C++ code for the prediction of a support vector machine (SVM) model using a coder configurer, then update model parameters of a deployed SVM model without having to regenerate the code. After training an SVM model, use the `learnerCoderConfigurer` function to create a coder configurer object, `ClassificationSVMCoderConfigurer` for an SVM classification model or `RegressionSVMCoderConfigurer` for an SVM regression model. A coder configurer offers convenient features to configure code generation options, generate C/C++ code, and update model parameters in the generated code.

- Configure code generation options and specify the coder attributes of SVM model parameters using object properties.
- Generate C/C++ code for the `predict` and `update` functions of the SVM model by using `generateCode`. Generating C/C++ code requires MATLAB Coder.
- Update model parameters in the generated C/C++ code without having to regenerate the code. This feature reduces the effort required to regenerate, redeploy, and reverify C/C++ code when you retrain the SVM model with new data or settings. Before updating model parameters, use `validatedUpdateInputs` to validate and extract the model parameters to update.

Nonlinear (Kernel) Classification and Regression: Perform hyperparameter optimization and cross-validation when fitting models using fitckernel and fitrkernel

`fitckernel` and `fitrkernel` can now tune a model by searching for optimal hyperparameters, and train a cross-validated model.

- Use the `'OptimizeHyperparameters'` and `'HyperparameterOptimizationOptions'` name-value pair arguments to minimize a cross-validation error over a preselected choice of hyperparameters.
- Use one of the name-value pair arguments `'CrossVal'`, `'CVPartition'`, `'Holdout'`, `'KFold'`, or `'Leaveout'` to perform cross-validation. If you specify one of these cross-validation name-value pair arguments when using `fitckernel` or `fitrkernel`, then the software returns a `ClassificationPartitionedKernel` or `RegressionPartitionedKernel` model object, respectively.

Multiclass Nonlinear Classification: Perform multiclass learning for nonlinear kernel classification by using fitcecoc

Create a multiclass ECOC model by using `ClassificationKernel` models as binary learners. Specify the kernel binary learners in the call to `fitcecoc` by setting the `'Learners'` name-value pair argument either to the value `'kernel'` or to a template object or cell vector of template objects created using the `templateKernel` function.

If you perform cross-validation when you train an ECOC model that uses kernel binary learners, then `fitcecoc` returns a `ClassificationPartitionedKernelECOC` model object.

Visualization: Display a confusion matrix using confusionchart

A confusion matrix for a classification problem helps you evaluate how well the classifier performs on a data set, and identify the areas where the classifier performs accurately or inaccurately. Use `confusionchart` to compute and plot a confusion matrix from the arrays of true labels and predicted labels. Additionally, you can:

- Pass a nonnegative-integer-valued confusion matrix to plot the confusion matrix.
- View summary statistics about your data, such as the percentages of correctly and incorrectly classified observations for given labels.
- Sort the classes of a confusion matrix according to the class-wise precision (positive predictive value), class-wise recall (true positive rate), or total number of correctly classified observations by using `sortClasses`.
- Control the appearance and behavior of the confusion matrix chart by modifying `ConfusionMatrixChart Properties`.

Code Generation: Generate C code for Cox proportional hazards regression (requires MATLAB Coder)

`coxphfit` supports code generation. You can generate C code that fits a Cox proportional hazard regression model to data.

For a full list of Statistics and Machine Learning Toolbox functions that support code generation, see Code Generation Support, Usage Notes, and Limitations.

GPU Support: gamrnd and randg accept gpuArray (requires Parallel Computing Toolbox)

`gamrnd` and `randg` accept `gpuArray` input arguments so that they execute on the GPU.

For a full list of Statistics and Machine Learning Toolbox functions that accept `gpuArray` input arguments, see Functions with `gpuArray` Arguments.

probplot, normplot, and wblplot Functions: Specify axes to add a probability plot

`probplot`, `normplot`, and `wblplot` can now add a probability plot into the target axes specified as an `Axes` object or a `UIAxes` object. For details, see the function reference pages.

Vector Dimension Argument: Operate on multiple dimensions simultaneously for selected functions

The following functions now accept a vector dimension argument to specify multiple operating dimensions simultaneously, as well as the option `'all'` to specify all dimensions of an array.

`geomean`	`nanstd`
`harmmean`	`nansum`
`iqr`	`nanvar`
`kurtosis`	`prctile`
`mad`	`quantile`
`moment`	`range`
`nanmax`	`skewness`
`nanmean`	`trimmean`
`nanmedian`	`zscore`
`nanmin`	

For example, if `X` is a numeric matrix, then `nanmean(X,'all')` returns the mean of all the elements in `X`, after removing all `NaN` values. Because `X` is an array with two dimensions, `nanmean(X,'all')` is equivalent to `nanmean(X,[1 2])`.

R2018a

Code Generation: Generate C code for distance calculation on vectors and matrices, and for prediction by using k-nearest neighbor with Kd-tree search and nontree ensemble models (requires MATLAB Coder)

The following functions support code generation:

- `grp2idx` — Create an index vector from a grouping variable.
- `pdist` — Find the pairwise distance between pairs of observations.
- `squareform` — Format a distance matrix.

When you train a *k*-nearest neighbor classification model by using `fitcknn` for code generation, you can now use the *K*d-tree search algorithm. For more details, see the Code Generation section of the `ClassificationKNN` class.

When you find nearest neighbors by using the functions `knnsearch` and `rangesearch` and the object functions `knnsearch` and `rangesearch` for code generation, you can now use the *K*d-tree search algorithm.

When you train an ensemble by using `fitcensemble` for code generation, you can now specify `'discriminant'` or `'knn'` as weak learners by using the `'Learners'` name-value pair argument. For more details, see the Code Generation section of the `CompactClassificationEnsemble` class and Code Generation Workflow Using MATLAB Coder App.

Nonlinear Regression for Big Data: Fit kernel SVM regression models by using random feature expansion

For nonlinear regression for big data, use the `fitrkernel` function to train a Gaussian kernel regression model using feature expansion. `fitrkernel` creates an instance of the `RegressionKernel` object.

`RegressionKernel` is a new model object for accessing and performing operations on the training data, and is especially suited for using tall arrays. `RegressionKernel` is compact; that is, it does not store the training data. With some exceptions, the syntax and methods of the `RegressionKernel` model object resemble those for other regression model objects that store the training data. For example, to predict responses of a `RegressionKernel` model, pass the trained model object and new predictor data to `predict`. To continue training, pass a trained model object and the training data to `resume`.

Big Data Algorithms: Compute confusion matrices and create nonstratified partitions for cross-validation on out-of-memory data

These functions support tall arrays:

- `confusionmat` displays confusion matrices.
- `cvpartition` supports nonstratified holdout cross-validation partitions with the `'Stratify'` name-value pair argument.

For a complete list of supported statistics functions, see Tall Array Support, Usage Notes, and Limitations.

Classification Learner App: Visualize and investigate high-density data with improved scatter plots

In the scatter plot of Classification Learner, you can now zoom in and out, pan across the plot, and change the stacking order of the plotted classes. This new functionality makes it easier to:

- Inspect and analyze your data.
- Determine the separation of classes for different input features.
- Inspect model results such as misclassification.
- Visually compare the performance of classification models in different regions of the predictor space.

To zoom or pan, point to the scatter plot and click one of the buttons that appear near the top-right corner of the plot. For examples and more information, see Classification Learner App.

cvpartition Function: Create nonstratified partitions of data for cross-validation

If the first input to `cvpartition` is `group`, then you can create nonstratified random partitions for k-fold and holdout cross-validation by using the `'Stratify'` name-value pair argument.

Bayesian optimization example

A new featured example, Bayesian Optimization with Tall Arrays shows how to use `bayesopt` to select optimal parameters for training a kernel classifier on tall arrays.

Code generation example

A new featured example Human Activity Recognition Simulink Model for Smartphone Deployment shows how to prepare a Simulink model that classifies human activity based on smartphone sensor signals for code generation and smartphone deployment.

Functionality Being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
`classregtree`	Errors	`fitctree` or `fitrtree`	Replace all instances of `classregtree` with `fitctree` or `fitrtree`.
`svmtrain`	Errors	`fitcsvm`	Replace all instances of `svmtrain` with `fitcsvm`.
`svmclassify`	Errors	`predict` of `ClassificationSVM`	Replace all instances of `svmclassify` with `predict`.
`princomp`	Errors	`pca`	Replace all instances of `princomp` with `pca`.

Functionality	Result	Use Instead	Compatibility Considerations
`fitNaiveBayes`	Removed	`fitcnb`	Replace all instances of `fitNaiveBayes` with `fitcnb`.
`ProbDist`	Removed	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistParametric`	Removed	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistKernel`	Removed	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistUnivKernel`	Removed	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistUnivParam`	Removed	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.

R2017b

Version: 11.2

New Features

Bug Fixes

Code Generation: Generate C code for prediction by using discriminant analysis, k-nearest neighbor, SVM regression, regression tree ensemble, and Gaussian process regression models (requires MATLAB Coder)

The following functions support code generation:

- `predict` (`CompactClassificationDiscriminant`) — Classify observations or estimate classification scores and costs by applying a discriminant analysis classification to new data.
- `predict` (`ClassificationKNN`) — Classify observations or estimate classification scores and costs by applying *k*-nearest neighbor classification, based on an exhaustive search, to new data.
- `predict` (`CompactRegressionSVM`) — Predict responses by applying a support vector machine (SVM) regression to new data.
- `predict` (`CompactRegressionEnsemble`) — Predict responses by applying ensembles of regression trees to new data.
- `predict` (`RegressionLinear`) — Predict responses by applying a linear regression to new data.
- `predict` (`CompactRegressionGP`) — Predict responses or estimate confidence intervals on predictions by applying a Gaussian process regression to new data.
- `knnsearch` (`ExhaustiveSearcher`) and `knnsearch`— Identify the *k*-nearest neighbors using the exhaustive search algorithm.
- `rangesearch` (`ExhaustiveSearcher`) and `rangesearch` — Identify all neighbors within a specified distance using the exhaustive search algorithm.
- `pdist2` — Compute the pairwise distance between two sets of observations.

When you train an SVM model by using `fitcsvm` for code generation, you can now specify a score transformation function by using the `'ScoreTransform'` name-value pair argument or by assigning the `ScoreTransform` object property. Therefore, `saveCompactModel` can accept compact SVM models equipped to estimate class posterior probabilities, that is, models returned by `fitposterior` or `fitSVMPosterior`. Also, you can now implement one-class learning.

When you train a linear classification model by using `fitclinear` for code generation, you can now specify either `'svm'` or `'logistic'` for the `'Learner'` name-value pair argument.

Big Data Algorithms: Fit kernel SVM classification models by using random feature expansion, fit linear SVM regression models, grow decision trees, and draw weighted random samples from out-of-memory data

These functions support tall arrays:

- `fitckernel` (new function) for Gaussian kernel classification using feature expansion
- `fitrlinear` for SVM regression
- `fitctree` for classification decision trees

Additionally, `datasample` supports weighted sampling without replacement with the `'Weights'` name-value pair argument.

For a complete list of supported statistics functions, see Tall Array Support, Usage Notes, and Limitations.

Parallel Bayesian Optimization: Tune hyperparameters faster by using parallel function evaluation (requires Parallel Computing Toolbox)

By computing in parallel, you can speed your Bayesian optimization. You can optimize hyperparameters in parallel by using `bayesopt` or any of the fit functions that support Bayesian optimization. For details, see Parallel Bayesian Optimization.

When calling a fit function that supports Bayesian optimization, you can limit the total optimization time by setting the `MaxTime` field of the `HyperparameterOptimizationOptions` structure. For a list of the fit functions that support Bayesian optimization, see Bayesian Optimization Using a Fit Function.

Machine Learning Apps: Select training data more efficiently in the Classification Learner and Regression Learner Apps

The New Session dialog box allows you to easily select a data set and response and predictor variables. This dialog box is well suited for large data sets or data sets with many variables. For more details, see Select Data and Validation for Classification Problem and Select Data and Validation for Regression Problem.

Partial Dependence Plots: Visualize relationships between features and predicted responses through marginalization

The `plotPartialDependence` function shows relationships between selected features and predicted responses for a trained regression model object. You can create a partial dependence plot (PDP), individual conditional expectation (ICE) plots, and centered ICE plots. You can also provide additional predictor data to evaluate and plot.

Nonlinear Classification for Big Data: Use fitckernel to train a Gaussian kernel classifier using feature expansion

For nonlinear classification for big data, use the `fitckernel` function to train a binary, Gaussian kernel classification model using feature expansion. `fitckernel` creates an object of the new class `ClassificationKernel`.

`ClassificationKernel` is a new class for accessing and performing operations on the training data. The `ClassificationKernel` model object does not store the training data in a similar way as `ClassificationLinear` and `RegressionLinear`. However, with some exceptions, the syntax and methods resemble those for the other classification model objects that store the training data. For example, to classify observations or estimate classification scores and costs for new data, pass a trained model object and new predictor data to `predict`. To continue training, pass a trained model object and the training data to `resume`.

For all methods and properties of the new objects, see the `ClassificationKernel` class page.

Gaussian Processes: Supply the initial step size for likelihood optimization, or optimize by using LBFGS

You can use LBFGS as an optimizer while training Gaussian process regression models. Specify this option using the `'Optimizer'` name-value pair argument of `fitrgp`.

If `'Optimizer'` is `'quasinewton'` or `'lbfgs'`, then you can specify the approximate maximum absolute value of the first optimization step using the `'InitialStepSize'` name-value pair argument. This specification can improve training time.

ksdensity and mvksdensity Functions: Specify a boundary correction method

For probability density function estimation, `ksdensity` and `mvksdensity` support the reflection method for boundary correction. You can specify this method by using the `'BoundaryCorrection','Reflection'` name-value pair argument.

regularize Function: Specify the maximum number of iterations allowed

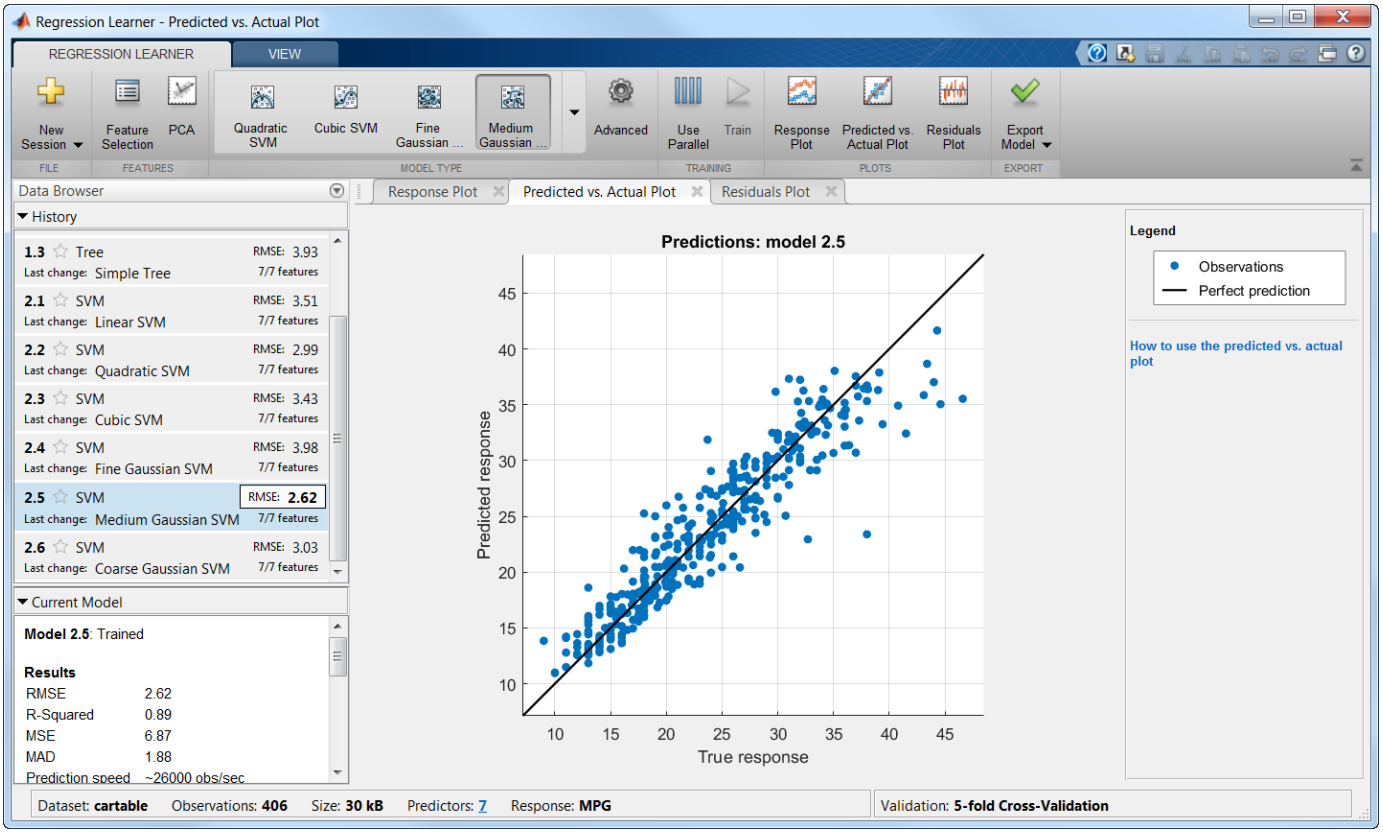
You can specify the maximum number of iterations allowed by using the `'MaxIter'` name-value pair argument in the call to the `regularize` function of a `RegressionEnsemble` object.

R2017a

Regression Learner App: Train regression models using supervised machine learning

Regression Learner is a new app that you can use to train regression models to predict data. Using this app, you can explore your data, select features, specify validation schemes, train models, and assess results. You can perform automated training to search for the best regression model type, including linear regression models, regression trees, Gaussian process models, support vector machines, and ensembles of regression trees.

To use the model with new data, or to learn about programmatic regression, you can export the model to the workspace or generate MATLAB code to recreate the trained model.



For more information, see Regression Learner App.

Big Data Algorithms: Perform support vector machine (SVM) and Naive Bayes classification, create bags of decision trees, and fit lasso regression on out-of-memory data

Several classification and regression functions add support for tall arrays:

- `fitclinear` for support vector machine classification
- `fitcnb` for Naive Bayes classification
- `TreeBagger` for creating bags of decision trees using ADMM algorithm

- `lasso` for fitting lasso regression using ADMM algorithm
- The `loss` and `predict` methods of these regression classes:
  - `CompactRegressionSVM`
  - `CompactRegressionGP`
  - `CompactRegressionTree`
  - `CompactRegressionEnsemble`
  - `RegressionLinear`
- The `predict`, `loss`, `margin`, and `edge` methods of these classification classes:
  - `CompactClassificationEnsemble`
  - `CompactClassificationTree`
  - `CompactClassificationTree`
  - `CompactClassificationDiscriminant`
  - `CompactClassificationNaiveBayes`
  - `CompactClassificationSVM`
  - `CompactClassificationECOC`
  - `ClassificationKNN`
  - `ClassificationLinear`

For a complete list of supported functions, see Tall Array Support, Usage Notes, and Limitations.

Code Generation: Generate C code for prediction by using linear models, generalized linear models, decision trees, and ensembles of classification trees (requires MATLAB Coder)

You can generate C code that predicts responses by using trained linear models, generalized linear models (GLM), decision trees, or ensembles of classification trees. The following prediction functions support code generation:

- `predict` — Predict responses or estimate confidence intervals on predictions by applying a linear model to new predictor data.
- `predict` or `glmval` — Predict responses or estimate confidence intervals on predictions by applying a GLM to new predictor data.
- `predict` or `predict` — Classify observations or estimate classification scores by applying a classification tree or ensemble of classification trees, respectively, to new data.
- `predict` — Predict responses by applying a regression tree to new data.

You can generate C code to simulate responses from a linear model or a generalized linear model using `random` or `random`, respectively.

Bayesian Statistics: Perform gradient-based sampling using Hamiltonian Monte Carlo (HMC) sampler

You can now perform Hamiltonian Monte Carlo (HMC) sampling from a probability density function. Use the `hmcSampler` function to create a `HamiltonianSampler` object for the log probability

density that you specify. The object samples from this density by generating a Markov chain with the corresponding equilibrium distribution using HMC.

After creating a `HamiltonianSampler` object, you can use:

- `tuneSampler` to tune the HMC sampler prior to drawing samples
- `drawSamples` to draw samples from the density
- `estimateMAP` to estimate the maximum of the log probability density
- `diagnostics` to assess the convergence

For a workflow example, see Bayesian Linear Regression Using Hamiltonian Monte Carlo.

Feature Extraction: Perform unsupervised feature learning by using sparse filtering and reconstruction independent component analysis (RICA)

Sparse filtering and reconstruction independent component analysis (RICA) are unsupervised, feature learning techniques for producing informative representations, including overcomplete representations, of high-dimensional predictor data. That is, they combine the raw predictor variables to produce an output feature set containing possibly more variables. Such representations can expose underlying correlations among the raw predictor variables, which can lead to improved predictive accuracy. These techniques are appropriate for image and video processing problems.

`sparsefilt` performs regularized sparse filtering. `rica` performs RICA, a variation of ICA. Whereas ICA enforces the feature weight matrix to be orthonormal during optimization, RICA includes a penalty term in the objective function allowing the feature weight matrix to deviate from orthonormality. Given a dimension for the output feature set, both methods learn predictor weights while imposing sparse/independent representations.

`sparsefilt` returns results as a `SparseFiltering` object and `rica` returns results as a `ReconstructionICA` object. After learning the feature weights, pass the object and predictor data to `transform` to transform the predictor data to the learned representation. You can train any classification or regression model using the new representation. For details, see the reference pages and Feature Extraction Workflow.

t-SNE: Visualize high-dimensional data

Using the `tsne` function you can embed high-dimensional data into two or three dimensions in order to view natural clustering. For details, see the function reference page and Visualize High-Dimensional Data Using t-SNE.

Survival Analysis: Fit Cox proportional hazards models with time-dependent covariates

Use `coxphfit` to fit a Cox proportional hazards model to data with time-dependent covariates. `coxphfit` uses the counting process type input to handle this type of data. You can enter the risk intervals for each subject using the `T` argument in the call to `coxphfit`.

Distribution Fitting App: dfittool Renamed to distributionFitter

To start the Distribution Fitter app at the command line, enter `distributionFitter`. The `dfittool` function also starts the Distribution Fitter app.

lasso and lassoglm Functions: Specify maximum number of iterations allowed

You can specify the maximum number of iterations allowed using the new `'MaxIter'` name-value pair argument in the call to `lasso` and `lassoglm`. For details, see the function reference pages.

Functionality Being Changed

The following functionality is removed or will be removed in a future release. Use the newer functionality instead.

Functionality	What Happens When You Use This Functionality?	Use This Functionality Instead	Compatibility Considerations
`treedisp`	Removed	`view` (`ClassificationTree`) or `view` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace all instances of `treedisp` with `view` (`ClassificationTree`) or `view` (`RegressionTree`).
`treefit`	Removed	`fitctree` or `fitrtree`	Replace all instances of `treefit` with `fitctree` or `fitrtree`.
`treeprune`	Removed	`prune` (`ClassificationTree`) or `prune` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace all instances of `treeprune` with `prune` (`ClassificationTree`) or `prune` (`RegressionTree`) .

Functionality	What Happens When You Use This Functionality?	Use This Functionality Instead	Compatibility Considerations
`treetest`	Removed	• `resubLoss` (`ClassificationTree`) or `resubLoss` (`RegressionTree`) • `loss` (`ClassificationTree`) or `loss` (`RegressionTree`) • `cvloss` (`ClassificationTree`) or `cvLoss` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace all instances of • `treetest(T,'resubstitution')` with `resubLoss` (`ClassificationTree`) or `resubLoss` (`RegressionTree`) • `treetest(T,'test',X,Y)` with `loss` (`ClassificationTree`) or `loss` (`RegressionTree`) • `treetest(T,'crossvalidate',X,Y)` with `cvloss` (`ClassificationTree`) or `cvloss` (`RegressionTree`)
`treeval`	Removed	`predict` (`ClassificationTree`) or `predict` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace all instances of `treeval` with `predict` (`ClassificationTree`) or `predict` (`RegressionTree`).

Functionality	What Happens When You Use This Functionality?	Use This Functionality Instead	Compatibility Considerations
`classregtree`	Warning	`fitctree` or `fitrtree`	Replace all instances of `classregtree` with `fitctree` or `fitrtree`.
`svmtrain`	Warning	`fitcsvm`	Replace instances of `svmtrain` with `fitcsvm`.
`svmclassify`	Warning	`fitcsvm`	Replace instances of `svmclassify` with `fitcsvm`.
`princomp`	Warning	`pca`	Replace all instances of `princomp` with `pca`.
`fitNaiveBayes`	Error	`fitcnb`	Replace all instances of `fitNaiveBayes` with `fitcnb`.
`ProbDist`	Error	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistParametric`	Error	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistKernel`	Error	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.

Functionality	What Happens When You Use This Functionality?	Use This Functionality Instead	Compatibility Considerations
`ProbDistUnivKernel`	Error	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistUnivParam`	Error	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.

R2016b

Big Data Algorithms: Perform dimension reduction, descriptive statistics, k-means clustering, linear regression, logistic regression, and discriminant analysis on out-of-memory data

Tall arrays provide a way to work naturally with out-of-memory data. You can create tall numeric arrays, cell arrays, categoricals, and you can use any of these tall types as variables in a tall table. For more information, see Tall Arrays.

You can perform various statistical and machine learning analyses on tall arrays. For a complete list of supported functions, see Tall Array Support, Usage Notes, and Limitations.

Bayesian Optimization: Tune machine learning algorithms by searching for optimal hyperparameters

Bayesian optimization is an algorithm for optimizing functions that can be nondifferentiable, discontinuous, and take a significant amount of time to evaluate. The algorithm internally maintains a Gaussian process model of the objective function, and uses objective function evaluations to train the model.

Bayesian optimization is well suited to optimizing hyperparameters of classification and regression algorithms. Hyperparameters are internal parameters of classification and regression functions, and can affect the performance of these functions. You can tune hyperparameters using Bayesian optimization in two ways:

- Using `bayesopt` to optimize a custom objective function over the parameters you define for tuning. This option gives you more control over optimization.
- Using the `OptimizeHyperparameters` name-value pair in training functions for classification and nonparametric regression. This option minimizes a cross-validation error over a preselected choice of hyperparameters.

For details, see Bayesian Optimization Workflow.

Feature Selection: Use neighborhood component analysis (NCA) to choose features for machine learning models

Neighborhood component analysis (NCA) is a nonparametric and embedded method for selecting features with the goal of maximizing prediction accuracy of regression and classification algorithms. `fscnca` and `fsrnca` perform NCA with regularization to learn feature weights for minimization of an objective function that measures the average leave-one-out classification or regression loss over the training data. `fscnca` and `fsrnca` support various optimization methods:

- Limited memory BFGS (LBFGS) — Recommended when the number of features is large.
- Stochastic gradient descent (SGD) — Recommended when the number of observations is large.
- Mini-batch LBFGS — Hybrid method that combines LBFGS and SGD. This method might converge faster than LBFGS or SGD.

`fscnca` returns results as a `FeatureSelectionNCAClassification` object, and `fsrnca` returns results as a `FeatureSelectionNCARegression` object.

Using these objects you can:

- Predict continuous responses or class labels using `predict`.
- Compute prediction or classification error using `loss` to estimate the prediction accuracy of selected features or to tune the regularization parameter.
- Refit the model using modified settings or continue iterations starting from the current solution using `refit`.

Code Generation: Generate C code for prediction by using SVM and logistic regression models (requires MATLAB Coder)

You can generate C code that classifies new observations by using trained, binary support vector machine (SVM) or logistic regression models, or multiclass SVM or logistic regression via error-correcting output codes (ECOC).

- `saveCompactModel` compacts and saves the trained model to disk.
- `loadCompactModel` loads the compact model in a prediction function that you declare. The prediction function can, for example, accept new observations and return labels and scores.
- `predict` classifies and estimates scores for the new observations in the prediction function.
  - To classify by using binary SVM models, see `predict`.
  - To classify by using binary logistic regression models, see `predict`.
  - To classify by using multiclass SVM or logistic regression via ECOC, see `predict`.

Classification Learner: Train classifiers in parallel (requires Parallel Computing Toolbox)

If you have Parallel Computing Toolbox, you can train models in parallel using Classification Learner. When you train classifiers, the app automatically starts a parallel pool of workers, unless you turn off the default parallel pool preference.

Parallel training allows you to train multiple classifiers at once and continue working. During training, you can examine results and plots from models, and initiate training of more classifiers.

For details, see Parallel Classifier Training.

Machine Learning Performance: Speed up Gaussian mixture modeling, SVM with duplicate observations, and distance calculations for sparse data

The algorithms for `pdist` and `fitgmdist` show improved performance.

When using `fitrsvm` and `fitcsvm`, you can remove duplicate observations for improved training time. This option replaces duplicate observations with a single observation whose weight is equal to the cumulative weight of these observations. To remove the duplicate observations, use the `RemoveDuplicates` name-value pair argument in the call to `fitrsvm` and `fitcsvm`.

Survival Analysis: Fit Cox proportional hazards models with new options for residuals and handling ties

Use `coxphfit` to fit Cox proportional hazards model to stratified data or data with ties. Use the `Strata` name-value pair argument to specify the stratification in the data. Use the `Ties` name-value pair argument to specify the method (Breslow or Efron) to handle tied failure times.

`coxphfit` also returns various residuals, including Cox-Snell, deviance, martingale, Schoenfeld, and score residuals.

Ensemble Methods Usability: Use simpler functions to train classification or regression ensembles

The `fitcensemble` and `fitrensemble` functions provide simpler interfaces to fit ensemble learners for classification and regression, respectively.

Unlike `fitensemble`, `fitcensemble` and `fitrensemble` provide options for Bayesian optimization.

Compatibility Considerations

For ensembles of boosted decision trees, `fitcensemble` and `fitrensemble` set their default tree depths to allow a maximum of ten splits, whereas `fitensemble` allows one split only by default.

Quantile Regression: Use bagged regression trees (TreeBagger) to implement quantile regression

In addition to predicting the conditional mean of a continuous response by growing a random forest [9], `TreeBagger` can predict conditional quantiles for robust regression and conditional distribution estimation.

- To predict quantiles of observations using a `TreeBagger` regression model, use `quantilePredict`.
- To estimate the quantile loss, use `quantileLoss`.
- To make out-of-bag predictions, use `oobQuantilePredict`.
- To estimate the out-of-bag quantile loss, use `oobQuantileLoss`.

GPU support: pdist, pdist2, and knnsearch accept gpuArray

`pdist`, `pdist2`, and `knnsearch` functions are enhanced to accept `gpuArray` input arguments so that they execute on the GPU. This support requires Parallel Computing Toolbox.

Gaussian Processes: Use additional popular kernel functions

You can use four new kernel (covariance) functions while training Gaussian Process Regression models. The new options are: exponential, rational quadratic, ARD exponential, and ARD rational quadratic kernel. You can specify the kernel function using the `KernelFunction` name-value pair argument in `fitrgp`. For more information on the kernel functions, see Kernel (Covariance) Function Options.

coxphfit Function: Specify coefficient initial values and observation weights

The name of the `'Init'` name-value pair argument is now `B0`. Use `B0` to specify the initial values for the estimated model coefficients. The name `Init` still works.

You can pass observation weights using the `'Frequency'` name-value pair argument, which now can be an array containing nonnegative scalar values.

fitgmdist Function: Set initial values using kmeans++ algorithm by default

The default value for the `Start` name-value pair argument of `fitgmdist` has changed to `'plus'`. Previously it was `'randSample'`.

Compatibility Considerations

To use `'randSample'` as the initial value setting method, specify `'Start','randSample'` in the call to `fitgmdist`.

fitgmdist Function: Specify tolerance for posterior probabilities

You can specify the tolerance for posterior probability estimates using the new `'ProbabilityTolerance'` name-value pair argument in the call to `fitgmdist`. In each iteration, after the estimation of posterior probabilities, `fitgmdist` sets any posterior probability that is not larger than the tolerance value to zero.

The `gmdistribution` object stores the tolerance value in the new `ProbabilityTolerance` property.

fitctree, fitrtree, and templateTree Functions: Unbiased feature selection for decision trees

For more flexibility in growing decision trees, `fitctree`, `fitrtree`, and `templateTree` offer several predictor-splitting algorithms. You can choose these alternatives using the `'PredictorSelection'` name-value pair argument.

- `'allsplits'`: At each node, MATLAB chooses the predictor that maximizes the split-criterion gain over all possible splits. This is the default and the expected algorithm. This algorithm is biased toward choosing predictors with many distinct values.
- `'curvature'`: MATLAB chooses the predictor by minimizing the p-value of a χ^2 test of independence between each predictor and response.
- `'interaction-curvature'`: MATLAB chooses the predictor to split by minimizing the smallest p-value of χ^2 tests of independence between:
  - Each predictor and response.
  - Each pair of predictors and response.

The algorithms that use χ^2 tests to split predictors are unbiased with respect to the number of distinct values in a predictor. Also, you can use these algorithms for feature selection.

R2016a

Machine Learning for High-Dimensional Data: Perform fast fitting of linear classification and regression models with techniques such as stochastic gradient descent and (L)BFGS using fitclinear and fitrlinear functions

For faster training on high-dimensional data sets, use `fitrlinear` and `fitclinear` to fit regularized, linear regression and binary classification models, respectively. For multiclass classification problems, specify parameters by creating a linear classification model template using `templateLinear`, and then pass the template object to `fitcecoc` for training.

Models for linear regression include support vector machine (SVM) and least squares regression, and models for binary classification include SVM and logistic regression. You can additionally include a lasso or ridge penalty to the objective function. The software optimizes the objective function using any of these algorithms:

- Stochastic gradient descent (SGD)
- Average SGD
- Broyden-Fletcher-Goldfarb-Shanno (BFGS)
- Limited-memory BFGS (LBFGS)
- Sparse Reconstruction by Separable Approximation (SpaRSA)

To increase the execution speed when training using `fitclinear`, `fitrlinear`, or `fitcecoc`, orient the predictor data so that columns correspond to observations, and set `'ObservationsIn','columns'`.

`fitrlinear` and `fitclinear` return:

- `RegressionLinear` and `ClassificationLinear` model objects, respectively, by default
- `RegressionPartitionedLinear` and `ClassificationPartitionedLinear` model objects, respectively, when you specify to cross-validate

For multiclass classification problems using linear classification models, `fitcecoc` returns:

- A `CompactClassificationECOC` model composed of `ClassificationLinear` model objects
- A `ClassificationPartitionedLinearECOC` model object when you specify to cross-validate

Unlike other regression and classification model objects in Statistics and Machine Learning Toolbox, these objects do not store the training data. However, with some exceptions, the syntax and methods resemble those for the other regression and classification model objects. For example, to predict responses or classes for new data, pass a trained linear regression or classification model object to `predict`.

Classification Learner: Train multiple models automatically, visualize results by class labels, and perform logistic regression classification

Classification Learner helps you explore methods for training models to classify data using supervised machine learning. In R2016a, new features in the app include:

- Automated classifier training. Get started by automatically training a selection of different classification models on your data with one click. Use automated training to quickly try a selection

of model types, then explore promising models interactively. Models show progress bars while training, and you can interrupt training between models or between validation folds. The app highlights the best accuracy score.

- Results visualized by class. In the scatter plot, show or hide particular classes, or focus only on correct or incorrect predictions.
- Logistic regression classification. Try a popular baseline classification technique on your data.

For details, see Train Classification Models in Classification Learner App.

Performance: Perform clustering using kmeans, kmedoids, and Gaussian mixture models faster when data has a large number of clusters

The algorithms for `kmeans`, `kmedoids`, and `fitgmdist` (for *plus* initialization method) show improved performance, particularly when there is a large number of clusters in the data.

Probability Distributions: Fit kernel smoothing density to multivariate data using the ksdensity and mvksdensity functions

`ksdensity` now supports fitting and plotting a probability density estimate for bivariate sample data. Use the new name-value pair `'PlotFcn'` to select the plot type for bivariate sample data. Choose from a contour plot, 3-D line plot, 3-D shaded surface plot, or contour plot under a 3-D shaded surface plot.

Use `mvksdensity` to fit a probability density estimate to multivariate data.

Stable Distributions: Model financial and other data that requires heavy-tailed distributions

There is a new probability distribution object for the stable distribution. This distribution is commonly used to model financial and other data that requires heavy-tailed distributions. Use `fitdist` to fit this distribution to data. Use `makedist` to specify the distribution parameters directly. Either function produces a probability distribution object that you can use to generate random samples or compute functions such as pdf and cdf.

Half-Normal Distributions: Model truncated data and create half-normal probability plots

There is a new probability distribution object for the half-normal distribution. This distribution is commonly used to model truncated data.

- Use `fitdist` to fit this distribution to data. Use `makedist` to specify the distribution parameters directly. Either function produces a probability distribution object that you can use to generate random samples or compute functions such as pdf and cdf.
- Use `probplot` to create a half-normal probability plot.

Linear Regression: CompactLinearModel object reduces memory footprint of linear regression model

`CompactLinearModel` is a new class for storing configurations of fitted linear regression models without storing fitting data or residuals. Fit a full `LinearModel` object using `fitlm`, then use the new compact method to create a `CompactLinearModel` object that retains only summary information about the model, such as coefficient values.

For all methods and properties of the new objects, see the `CompactLinearModel` and `LinearModel` class pages.

Robust covariance estimation for multivariate sample data using robustcov

The new function `robustcov` estimates a robust covariance matrix for multivariate sample data. Robust covariance estimates are less sensitive to outliers in the sample data than classical estimation methods. Estimation options available using robustcov include the FAST-MCD (Minimum Covariance Determinant) estimate, the Orthogonalized Gnanadesikan-Kettenring (OGK) estimate, and an estimate based on “concentration” techniques.

Squared Euclidean distance measure for pdist and pdist2 functions

`pdist` and `pdist2` support the option of returning the squared Euclidean distance. You can specify this measure by setting the `distance` positional argument to `'squaredeuclidean'`.

Performance enhancements for nearest neighbor search using kd-tree

For dual-core systems and above, the `knnsearch` method of `KDTreeSearcher` parallelizes the k-nearest neighbor search using Intel Threading Building Blocks (TBB). For details on Intel TBB, see https://software.intel.com/tbb.

GPU support for extreme value distribution functions and kmeans

The following Statistics and Machine Learning Toolbox functions are enhanced to accept `gpuArray` input arguments so that they execute on the GPU. This support requires Parallel Computing Toolbox.

```
evcdf
evpdf
evinv
evlike
evrnd
evstat
kmeans
```

Changes to default online update phase for kmeans function

The default value of `OnlinePhase` name-value pair argument for `kmeans` is now `'off'`. Previously, the default value was `'on'`.

Compatibility Considerations

`kmeans` might return different results than when the default value of `OnlinePhase` was `'on'`. To turn the online update phase on, use the `'OnlinePhase','on'` name-value pair argument in the call to `kmeans`.

Name change in ksdensity

The name-value pair `'npoints'` for `ksdensity` has changed to `'NumPoints'`.

Compatibility Considerations

The new name-value pair argument name `'NumPoints'` does not work in previous releases. Please refer to the documentation for the correct version of Statistics and Machine Learning Toolbox you use.

Name change in paretotails

The following property names for the `paretotails` class have changed.

New Property Name	Old Property Name	Class

`NumSegments`
`UpperParameters`
`LowerParameters`
	`nsegments`
	`lowerparams`
	`upperparams`
		`paretotails`

Compatibility Considerations

The new property names do not work in previous releases. Please refer to the documentation for the correct version of Statistics and Machine Learning Toolbox you use.

Functionality Being Changed

Following functionality will be removed in a future release. Use the newer functionality instead.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`treedisp`	Error	`view` (`ClassificationTree`) or `view` (`RegressionTree`)	Use`fitctree` or `fitrtree` to grow a tree. Replace all instances of `treedisp` with`view` (`ClassificationTree`) or `view` (`RegressionTree`).

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`treefit`	Error	`fitctree` or `fitrtree`	Replace all instances of `treefit` with `fitctree` or `fitrtree`.
`treeprune`	Error	`prune` (`ClassificationTree`) or `prune` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace all instances of `treeprune` with `prune` (`ClassificationTree`) or `prune` (`RegressionTree`) .
`treetest`	Error	• `resubLoss` (`ClassificationTree`) or `resubLoss` (`RegressionTree`) • `loss` (`ClassificationTree`) or `loss` (`RegressionTree`) • `cvLoss` (`ClassificationTree`) or `cvLoss` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace all instances of • `treetest(T,'resubstitution')` with `resubLoss` (`ClassificationTree`) or `resubLoss` (`RegressionTree`) • `treetest(T,'test',X,Y)` with `loss` (`ClassificationTree`) or `loss` (`RegressionTree`) • `treetest(T,'crossvalidate',X,Y)` with `cvLoss` (`ClassificationTree`) or `cvLoss` (`RegressionTree`)
`treeval`	Error	`predict` (`ClassificationTree`) or `predict` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace all instances of `treeval` with `predict` (`ClassificationTree`) or `predict` (`RegressionTree`).
`classify`	Warning	`fitcdiscr`	Replace all instances of `classify` with `fitcdiscr`.

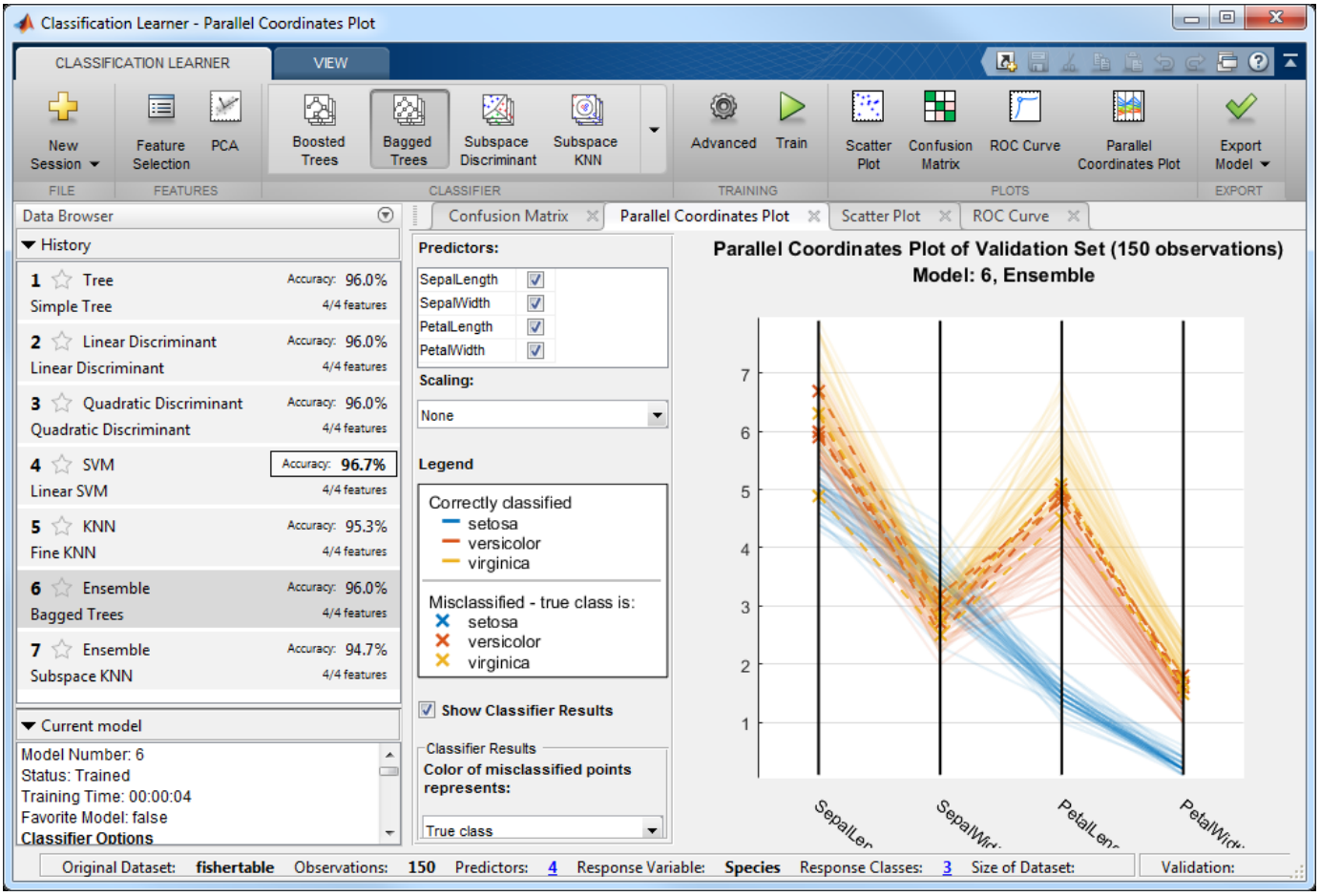
Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`fitNaiveBayes`	Warning	`fitcnb`	Replace all instances of `fitNaiveBayes` with `fitcnb`.
`ProbDist`	Warning	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistParametric`	Warning	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistKernel`	Warning	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistUnivKernel`	Warning	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistUnivParam`	Warning	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.

R2015b

Classification Learner: Train discriminant analysis to classify data, train models using categorical predictors, and perform dimensionality reduction using PCA

Classification Learner helps you explore methods for training models to classify data using supervised machine learning. In R2015b, new features in the app include:

- Discriminant analysis classifier: Train classifiers with fast, accurate, and easy to interpret discriminant analysis, which is good for wide datasets.
- Principal component analysis (PCA): Reduce the dimensionality of the predictor space using PCA to help prevent overfitting.
- Categorical predictors: Train classification models when some or all predictors are categorical variables. Previously you could only have numeric predictors in the app.
- Data import from file: Import spreadsheets, text, csv, and other files into the app. Previously you could only select data from the workspace.
- Parallel coordinates plot: Visualize training data and misclassified points to investigate features to include or exclude. Parallel coordinates can help visualize 3 to 10 dimensions of data on a single plot and see patterns. This can help you understand relationships between features and identify useful predictors for separating classes.
- ROC Threshold: Assess classifier performance by seeing where the threshold for your trained classifier lies on the ROC curve.



For details, see Explore Classification Models Interactively.

Compatibility Considerations

If you exported a classification model from Classification Learner to the workspace and wrote a script to make predictions with new data in R2015a, you must change your code to use models exported from the app in R2015b. Use the new `trainedClassifier.predictFcn`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
In user scripts, `predict` function used on model exported from Classification Learner in R2015a	Errors in R2015b	`trainedClassifier.predictFcn`	Classification models exported from Classification Learner changed from a classification object in R2015a to a structure in R2015b. The new structure contains a classification object and a new predict function, `predictFcn`. The new structure allows you to make predictions for models that include principal component analysis (PCA).

When you export a model from Classification Learner, the app displays information about the exported model in the command window. The message shows how to make predictions using the model.

To fix your code from R2015a to work with classifiers exported in R2015b, change `predict` to `trainedClassifier.predictFcn`, where `trainedClassifier` is the name of your struct variable.

For example, change this R2015a code:

```
yfit = predict(trainedClassifier,T{:,trainedClassifier.PredictorNames})
```

To this R2015b code:

```
yfit = trainedClassifier.predictFcn(T)
```

Supply the data `T` in same data type as your training data used in the app (table or matrix).

- If you supply a table, ensure it contains the same predictor names as your training data. The `predictFcn` ignores additional variables in tables.
- If you supply a matrix, it must contain the same predictor columns or rows as your training data, and no response variable or other unused variables.

Note that the default name `trainedClassifier` increments every time you export to avoid overwriting your classifiers, e.g., `trainedClassifier1`. Make sure your code uses the correct name of your struct variable.

You can also extract the classification object from the exported struct for further analysis (e.g., `trainedClassifier.ClassificationSVM`, `trainedClassifier.ClassificationTree`, etc., depending on your model type). Be aware that if you used feature selection such as PCA in the app, you will need to take account of this transformation by using the information in the PCA fields of the struct.

Nonparametric Regression: Fit models using support vector regression (SVR) or Gaussian processes (Kriging)

You can train nonparametric regression models using support vector machine (SVM) regression or Gaussian process regression (GPR).

- **SUPPORT VECTOR REGRESSION:** The `fitrsvm` function trains a SVM regression model. Using the new functionality, you can:

  - Specify the kernel function
  - Provide observation weights
  - Train a cross-validated model
  - Predict responses using the trained model
  - Compute resubstitution statistics

  `fitrsvm` creates a `RegressionSVM` or `RegressionPartitionedSVM` object. `RegressionSVM` is a new class for accessing and performing operations on the training data. `CompactRegressionSVM` is a new class for storing configurations of trained models without storing training data. `RegressionPartitionedSVM` is a new class for a set of cross-validated SVM regression models trained on cross-validated folds.

  For all methods and properties of the new objects, see the `RegressionSVM`, `CompactRegressionSVM`, and `RegressionPartitionedSVM` class pages.
- **GAUSSIAN PROCESS REGRESSION:** The `fitrgp` function trains a Gaussian process regression (GPR) model. Using the new functionality, you can:

  - Specify the fitting, prediction, and active set selection methods
  - Specify the kernel (covariance) function and provide initial values for the hyperparameters
  - Train a cross-validated model
  - Compute response predictions along with the prediction intervals using the trained model
  - Compute resubstitution statistics
  - Compute post-fit statistics

  `fitrgp` creates a `RegressionGP` or `RegressionPartitionedModel` object. `RegressionGP` is a new class for accessing and performing operations on the training data. `CompactRegressionGP` is a new class for storing configurations of trained models without storing training data. `RegressionPartitionedModel` is an existing class for a set of cross-validated GPR models trained on cross-validated folds.

  For all methods and properties of the new objects, see the `RegressionGP`, `CompactRegressionGP`, and `RegressionPartitionedModel` class pages.

Tables and Categorical Data for Machine Learning: Use table and categorical predictors in classification and nonparametric regression functions and in Classification Learner

The classification functions `fitctree`, `fitcsvm`, `fitcdiscr`, `fitcnb`, and `fitcknn`, non-parametric regression functions `fitrtree`, `fitrsvm`, and `fitrgp`, and the ensemble learner `fitensemble` accept data in `table`. Except for `fitcdiscr`, all of the above listed functions and

Classification Learner accept `categorical` predictors. For more information on these data types, see Tables and Categorical Arrays.

Code Generation: Automatically generate C and C++ code for kmeans and randsample functions (using MATLAB Coder)

`kmeans` and `randsample` are now supported for code generation. For a full list of Statistics and Machine Learning Toolbox functions that are supported by MATLAB Coder, see Statistics and Machine Learning Toolbox.

GPU Acceleration: Speed up computation for over 65 functions including probability distributions, descriptive statistics, and hypothesis testing (using Parallel Computing Toolbox)

The following Statistics and Machine Learning Toolbox functions are enhanced to accept `gpuArray` input arguments so that they execute on the GPU.

betacdf*	expfit*	geomean*	mvncdf*	poisscdf*	tpdf
betainv*	expinv*	geopdf*	mvnpdf*	poissinv*	tstat
betapdf	explike*	geornd*	mvnrnd*	poisspdf	ttest*
betastat	exppdf*	geostat	nanmax*	poisstat,	ttest2*
binofit*	exprnd*	harmmean*	nanmean*	prctile	trimmean
binoinv	expstat	iqr*	nanmedian	quantile	unidcdf
binopdf	fcdf*	kurtosis*	nanmin*	range*	unidinv
binornd*	finv*	logncdf*	nanstd*	raylcdf	unidpdf
binostat	fstat	lognfit*	nansum*	raylfit*	unidrnd*
chi2cdf*	gamcdf*	logninv*	nanvar*	raylinv	unidstat
chi2gof*	gaminv*	lognlike*	normcdf*	raylpdf	unifinv
chi2inv*	gamlike*	lognpdf*	norminv*	raylrnd*	unifrnd*
chi2stat	gampdf	lognrnd*	normlike*	raylstat	unifstat
cholcov*	gamstat	lognstat	normpdf*	skewness*	vartest
corrcov*	geocdf*	mad*	normrnd*	tcdf*	vartest2*
expcdf*	geoinv*	moment*	normstat	tinv*	zscore*
					ztest*

* These functions perform faster on the GPU than the CPU. Other functions in the table show similar performance on the GPU and on the CPU.

Option to turn off clipping of Alpha coefficients in fitcsvm

You can specify not to clip the Alpha coefficient for an observation to zero or the box-constraint value for that observation, using the `'ClipAlphas',false` name-value pair argument of `fitcsvm`, while training a support vector machine for classification.

Name changes in TreeBagger

Some property and method names for `TreeBagger` and `CompactTreeBagger` classes and name-value pair argument names for the methods of these classes have changed as follows.

New Property Name	Old Property Name	Class
InBagFraction TreeArguments ComputeOOBPredictorImportance NumPredictorsToSample NumTrees PredictorNames OOBPermutedPredictorDeltaError OOBPermutedPredictorDeltaMeanMargin OOBPermutedPredictorCountRaiseMargin NumPredictorSplit SurrogateAssociation MinLeafSize DeltaCriterionDecisionSplit	FBoot TreeArgs ComputeOOBVarImp NVarToSample NTrees VarNames OOBPermutedVarDeltaError OOBPermutedVarDeltaMeanMargin OOBPermutedVarCountRaiseMargin NVarSplit VarAssoc MinLeaf DeltaCritDecisionSplit	TreeBagger
NumTrees PredictorNames NumPredictorSplit SurrogateAssociation	NTrees VarNames DeltaCritDecisionSplit VarAssoc	CompactTreeBagger

New Name-Value Pair Name	Old Name-Value Pair Name	Method	Class
InBagFraction NumPredictorsToSample OOBPrediction OOBPredictorImportance NumPredictorsToSample NumPrint MinLeafSize MinParentSize Cost Method Options Prior SampleWithReplacement Weights	FBoot NVarToSample oobpred oobvarimp nvartosample nprint minleaf minparent cost method options prior samplewithreplacement weights	TreeBagger (constructor)	TreeBagger
NumPrint	nprint	growTrees, fillprox	TreeBagger
UseInstanceForTree	useifort	predict, error, margin, meanMargin	TreeBagger
Options	options	growTrees	TreeBagger

New Name-Value Pair Name	Old Name-Value Pair Name	Method	Class
Trees	trees	fillprox, predict, oobPredict, oobError, oobMargin, oobMeanMargin, error, margin, meanMargin,	TreeBagger
TreeWeights	treeweights	predict, oobPredict, oobError, oobMargin, oobMeanMargin, error, margin, meanMargin	TreeBagger
Mode	mode	oobError, oobMargin, oobMeanMargin, error, margin, meanMargin	TreeBagger
Keep Colors MDSCoordinates	keep colors mdscoords	mdsprox	TreeBagger
UseInstanceForTree Trees TreeWeights	useifort trees treeweights	predict, error, margin, meanMargin	CompactTreeBagger
Mode	mode	error, margin, meanMargin	CompactTreeBagger
Weights	weights	error, meanMargin	CompactTreeBagger
Data Labels	data labels	outlierMeasure, mdsprox	CompactTreeBagger
Colors MDSCoordinates	colors mdscoords	mdsprox	CompactTreeBagger

New Method Name	Old Method Name	Class
mdsprox	mdsProx	TreeBagger, CompactTreeBagger
fillprox	fillProximities	TreeBagger, CompactTreeBagger

Compatibility Considerations

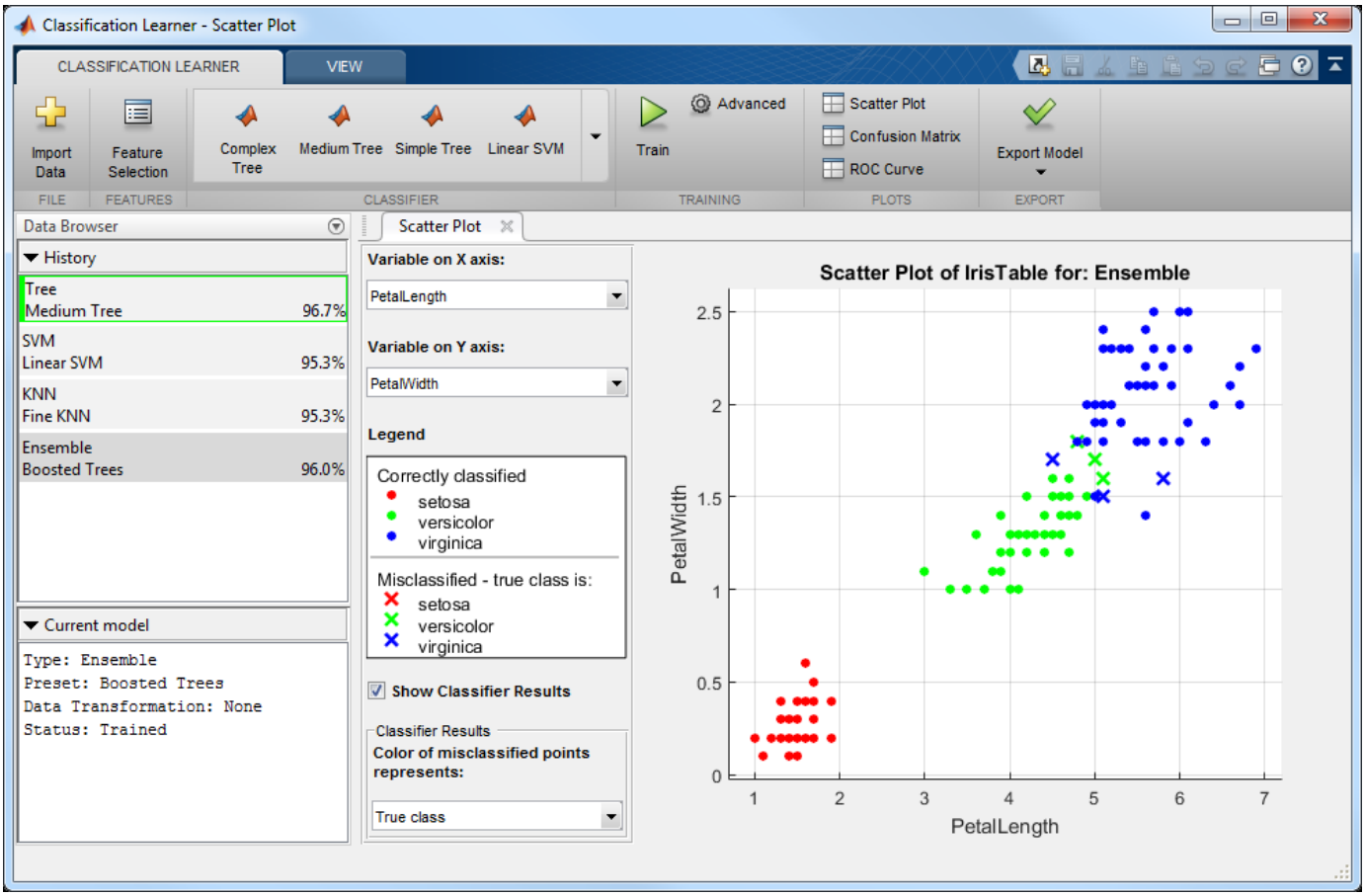
The new property, method, and name-value pair argument names do not work in the previous releases. Please refer to the documentation for the correct version of Statistics and Machine Learning Toolbox you use.

R2015a

Classification app to train models and classify data using supervised machine learning

Classification Learner is a new app that lets you train models to classify data using supervised machine learning. You can explore your data, select features, specify cross-validation schemes, train models, and assess results. You can choose from several classification types including decision trees, support vector machines, nearest neighbors, and ensemble classification.

Perform supervised machine learning by supplying a known set of input data (observations or examples) and known responses to the data (i.e., labels or classes). Use the data to train a model that generates predictions for the response to new data. To use the model with new data, or to learn about programmatic classification, you can export the model to the workspace or generate MATLAB code to recreate the trained model.



For details, see Explore Classification Models Interactively.

Statistical tests for comparing accuracies of two classification models using compareHoldout, testcholdout, and testckfold functions

You can statistically assess the predictive accuracies of two classification models using holdout sample predictions or repeated cross validation.

- The `testcholdout` accepts holdout sample predicted labels from both classification models and the true labels. This function implements the asymptotic, exact, or mid-*p* version of McNemar's test. If you specify misclassification costs, `testcholdout` compares the models using a likelihood ratio or a chi-square test.
- The `compareHoldout` object function accepts any two trained classification model objects in Statistics and Machine Learning Toolbox, sets of holdout predictor data for both models, and corresponding true labels. Like `testcholdout`, this object function implements the asymptotic, exact, or mid-*p* version of McNemar's test. If you specify misclassification costs, `compareHoldout` compares the models using a likelihood ratio or a chi-square test.
- The `testckfold` function accepts any two trained classification model objects or templates in Statistics and Machine Learning Toolbox, and repeatedly applies *k*-fold cross validation using two sets of out-of-sample predictor data and true labels. Then, `testckfold` assesses the resulting accuracies using a *t* or an *F* test.

Speedup of kmedoids, fitcknn, and other functions when using cosine, correlation, or spearman distance calculations

Pairwise distance calculations (by `pdist` and `pdist2`) in `kmedoids` and `fitcknn` use Basic Linear Algebra Subroutines (BLAS) libraries based on the Intel Math Kernel Library (MKL). For details on Intel MKL, see https://software.intel.com/intel-mkl.

Performance enhancements for decision trees and performance curves

- For dual-core systems and above, `fitctree`, `fitrtree`, and `fitensemble` parallelize training decision trees using Intel Threading Building Blocks (TBB). For details on Intel TBB, see https://software.intel.com/tbb.
- You can parallelize computation of pointwise confidence intervals `perfcurve` returns for the *x*- and *y*-coordinates, thresholds, or the area under the curve measure. You need Parallel Computing Toolbox to use this option.

Additional option to control decision tree depth using 'MaxNumSplits' argument in fitctree, fitrtree, and templateTree functions

You can control the depth of a decision tree by choosing the maximal number of splits (branch nodes) rather than choosing the minimum leaf size or minimum parent size. Specify this option using the `'MaxNumSplits'` name-value pair argument in the `fitctree`, `fitrtree`, or `templateTree`. Full trees (`ClassificationTree` or `RegressionTree` classifiers) contain the field `MaxNumSplits` in the property `ModelParameters` to store the specified maximal number of splits.

Code generation for pca and probability distribution functions (using MATLAB Coder)

`pca`, `betafit`, `betalike` and `pearsrnd` are now supported for code generation. For a full list of Statistics and Machine Learning Toolbox functions that are supported by MATLAB Coder, see Statistics and Machine Learning Toolbox.

Power and sample size for two-sample t-test using sampsizepwr function

`sampsizepwr` returns the power, sample size, or alternative hypothesis value for a two-sample *t*-test for populations with equal variances. Specify the two-sample *t*-test using `'t2'` as the `'testtype'` input variable.

Discard support vectors of SVM and ECOC models

You can reduce the memory footprint of a linear support vector machine (SVM) model by discarding their support vectors. Pass a trained SVM model (i.e., a `ClassificationSVM` or `CompactClassificationSVM` object) to `discardSupportVectors` to discard:

- The α coefficients (stored in the `Alpha` property)
- The support vectors (stored in the `SupportVectors` property)
- The support vector labels (stored in the `SupportVectorLabels` property)

By default, `fitcsvm` and `compact` do not discard the α coefficients, support vectors, and the support vector labels.

You can pass a trained error correcting output codes (ECOC) model (i.e., a `ClassificationECOC` or `CompactClassificationECOC` object) to `discardSupportVectors` to similarly discard the α coefficients, support vectors, and the support vector labels from all linear SVM binary learners. To control whether linear SVM binary learners store support vectors, create an SVM template using `templateSVM` and set the `'SaveSupportVectors'` name-value pair argument.

Compatibility Considerations

By default, `fitcecoc` discards the α coefficients, support vectors, and the support vector labels from all linear SVM binary learners. To store these estimates, create an SVM template and specify `'SaveSupportVectors',true`. Then, pass the SVM template to `fitcecoc`.

Minimum leaf size for boosted regression trees

The default minimum leaf size for boosted regression trees is `5`.

Compatibility Considerations

To train boosted regression trees using the previous defaults, construct a regression tree template using `templateTree`, and specify `'MinLeafSize',1` and `'MaxNumSplits',1`. Then, pass the regression tree template to `fitensemble`.

Additional option to plot grouped histograms using the scatterhist and gplotmatrix functions

The `scatterhist` function includes two new name-value pair arguments that allow you to display grouped histograms of the marginal distributions along the *x*- and *y*-axes of the scatter plot:

- `'PlotGroup'` allows you to specify whether to plot the marginal distributions by group or for the entire data set.

- `'Style'` allows you to specify whether to display a stairstep plot, which shows the outline of a histogram without filling in the bars, or a histogram bar plot.

If you specify a grouping variable that contains more than one group, then by default `scatterhist` displays grouped stairstep plots. If you specify a grouping variable that contains only one group, then `scatterhist` displays a histogram bar plot. To display kernel density plots, use the `'Kernel'` name-value pair argument.

The positional argument `'dispopt'` in `gplotmatrix` supports two additional options for controlling the appearance of the plots along the diagonal of the plot matrix:

- `'stairs'` displays a stairstep plot, which shows the outline of the grouped histograms without filling in the bars.
- `'grpbars'` displays a standard grouped histogram bar plot.

Confidence interval computation for residuals using the function regress

`regress` computes the confidence intervals for studentized residuals using the degree of freedom $n - p - 1$, where n is the number of observations and p is the number of predictor variables.

Compatibility Considerations

The degrees of freedom in the computation of confidence intervals for studentized residuals that `regress` returns is $n - p - 1$ rather than $n - p$. Results may differ from those in previous releases.

Functionality Being Changed

Following functionality will be removed in a future release. Use the newer functionality instead.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`princomp`	Warns	`pca`	Replace instances of `princomp` with `pca`.
`treedisp`	Warns	`view` (`ClassificationTree`) or `view` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace instances of `treedisp` with`view` (`ClassificationTree`) or `view` (`RegressionTree`).
`treefit`	Warns	`fitctree` or `fitrtree`	Replace instances of `treefit` with `fitctree` or `fitrtree`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`treeprune`	Warns	`prune` (`ClassificationTree`) or `prune` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace instances of `treeprune` with `prune` (`ClassificationTree`) or `prune` (`RegressionTree`) .
`treetest`	Warns	• `resubLoss` (`ClassificationTree`) or `resubLoss` (`RegressionTree`) • `loss` (`ClassificationTree`) or `loss` (`RegressionTree`) • `cvLoss` (`ClassificationTree`) or `cvLoss` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace instances of • `treetest(T,'resubstitution')` with `resubLoss` (`ClassificationTree`) or `resubLoss` (`RegressionTree`) • `treetest(T,'test',X,Y)` with `loss` (`ClassificationTree`) or `loss` (`RegressionTree`) • `treetest(T,'crossvalidate',X,Y)` with `cvLoss` (`ClassificationTree`) or `cvLoss` (`RegressionTree`)
`treeval`	Warns	`predict` (`ClassificationTree`) or `predict` (`RegressionTree`)	Use `fitctree` or `fitrtree` to grow a tree. Replace instances of `treeval` with `predict` (`ClassificationTree`) or `predict` (`RegressionTree`).
`classify`	Still runs	`fitcdiscr`	Replace instances of `classify` with `fitcdiscr`.
`classregtree`	Still runs	`fitctree` or `fitrtree`	Replace instances of `classregtree` with `fitctree` or `fitrtree`.
`fitNaiveBayes`	Still runs	`fitcnb`	Replace instances of `fitNaiveBayes` with `fitcnb`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`ProbDist`	Still runs	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistParametric`	Still runs	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistKernel`	Still runs	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistUnivKernel`	Still runs	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`ProbDistUnivParam`	Still runs	`makedist` and `fitdist`	To create and fit probability distribution objects, use `makedist` and `fitdist` instead.
`svmclassify`	Still runs	`fitcsvm`	Replace instances of `svmclassify` with `fitcsvm`.
`svmtrain`	Still runs	`fitcsvm`	Replace instances of `svmtrain` with `fitcsvm`.

R2014b

Version: 9.1

New Features

Bug Fixes

Multiclass learning for support vector machines and other classifiers using the fitcecoc function

The `fitcecoc` function fits an error-correcting output code (ECOC) model for multiclass learning. Using training data and a coding scheme, `fitcecoc` combines a set of binary learners, such as SVM classifiers, using a coding design to create a multiclass model. You can use a supported coding scheme, or specify your own using the `designecoc` function. The new functionality also supports fitting posterior probabilities for most methods. `fitcecoc` creates an object of the new class `ClassificationECOC` or `ClassificationPartitionedECOC`.

`ClassificationECOC` is a new class for accessing and performing operations on the training data. `CompactClassificationECOC` is a new class for storing configurations of trained models without storing training data. `ClassificationPartitionedECOC` is a new class for a set of cross-validated ECOC models trained on cross-validated folds.

`ClassificationECOC` is built on the same framework as `ClassificationTree`, `ClassificationDiscriminant`, `ClassificationKNN`, and `ClassificationSVM`, so you have a variety of options and methods, including:

- Cross validation
- Resubstitution statistics
- Generalization statistics
- Weighted classification

For all methods and properties of the new objects, see the `ClassificationECOC`, `CompactClassificationECOC`, and `ClassificationPartitionedECOC` class pages.

Generalized linear mixed-effects models using the fitglme function

`GeneralizedLinearMixedModel` is a new class for fitting generalized linear mixed-effects (GLME) models. Fit GLME models using `fitglme`. You can:

- Specify GLME models using the formula notation.
- Fit GLME models for a response with conditional distribution of normal, binomial, poisson, gamma, or inverse Gaussian.
- Specify the link function using a string or a structure.
- Fit GLME models using maximum pseudo likelihood (MPL), restricted maximum pseudo likelihood (REMPL), maximum likelihood using Laplace approximation, or maximum likelihood using approximate Laplace approximation with fixed effects profiled out.
- Specify a covariance pattern for the random effects.
- Calculate estimates of the empirical Bayes predictors (EBPs) for random effects.
- Perform custom hypothesis tests on fixed effects.
- Compute confidence intervals on fixed effects, random effects, and covariance parameters.
- Examine residuals, diagnostic plots, fitted values, and design matrices.
- Compare two different models using the theoretical likelihood ratio test.
- Make predictions on new data using the fitted GLME model.
- Generate random data using the fitted GLME model at new design points.

- Refit a new GLME model based on the previously fitted model, using a new response vector.

For the properties and methods of this object, see the class page for `GeneralizedLinearMixedModel`.

Clustering that is robust to outliers using the kmedoids function

The `kmedoids` function partitions data into *k* clusters using the k-medoids algorithm. This functionality provides clustering on categorical data, clustering using arbitrary distance metrics, robustness to outliers, and scaling to large data sets.

Speedup of the kmeans and gmdistribution clustering using the kmeans++ algorithm

The `kmeans` and `fitgmdist` functions perform clustering using the k-means++ initialization algorithm. The default initialization algorithm for `kmeans` is now set to k-means++.

Fisher’s exact test for 2-by-2 contingency tables

The `fishertest` function performs Fisher’s exact test on 2-by-2 contingency tables. The new functionality is appropriate for small sample sizes.

templateEnsemble function for creating ensemble learning template

You can use the `templateEnsemble` function to create an ensemble learning template suitable for training error-correcting output code (ECOC) multiclass classifiers. In particular, you can perform multiclass classification by specifying binary learners that use the ensemble methods GentleBoost, LogitBoost, and RobustBoost.

templateSVM function for creating SVM learning template

You can use the `templateSVM` function to create an SVM learning template suitable for training error-correcting output code (ECOC) multiclass classifiers. In particular, you can perform multiclass classification by specifying binary learners that standardize predictor data or use a particular box constraint.

Standardizing training data in k-nearest neighbor classification

You can standardize the training data before fitting the model in k-nearest neighbor classification. The standardization takes into account the observation weights and missing data. You can specify this option using the `'Standardize'` name-value pair argument in the `fitcknn` function.

fitcnb function for naive Bayes classification

You can use the `fitcnb` function to train a multiclass naive Bayes model. `fitcnb` creates an object of the new class `ClassificationNaiveBayes`.

`ClassificationNaiveBayes` is a new class for accessing and performing operations on the training data. `CompactClassificationNaiveBayes` is a new class for storing configurations of trained models without storing training data.

The `fitcnb` function and `ClassificationNaiveBayes` and `CompactClassificationNaiveBayes` classes include the functionality of the `fitNaiveBayes` function and `NaiveBayes` class. `ClassificationNaiveBayes` is built on the same framework as `ClassificationTree`, `ClassificationDiscriminant`, `ClassificationKNN`, and `ClassificationSVM`, so you have a variety of additional options and methods, including:

- Cross validation
- Resubstitution statistics
- Generalization statistics
- Weighted classification

You can also use the `templateNaiveBayes` function to create a naive Bayes classifier template suitable for training error-correcting output code (ECOC) multiclass classifiers.

For all methods and properties of the new objects, see the `ClassificationNaiveBayes` and `CompactClassificationNaiveBayes` class pages.

R2014a

Repeated measures modeling for data with multiple measurements per subject

`fitrm` is a new function for fitting models to repeated measures data, where each subject has multiple response measurements. It produces an object of the new `RepeatedMeasuresModel` class. You can:

- Perform analysis of variance for between-subjects factors using `anova`.
- Perform multivariate analysis of variance using `manova`.
- Perform hypothesis tests on the coefficients using `coeftest`.
- Perform repeated measures analysis of variance using `ranova`.
- Test for sphericity (compound symmetry) with Mauchly's test using `mauchly`.
- Plot data and estimated marginal means with optional grouping using `plot` and `plotprofile`.
- Compute summary statistics organized by group using `grpstats`.
- Perform multiple comparisons of marginal means using `multcompare`.
- Make predictions on new data with the fitted repeated measures model using `predict`.
- Generate random data with the fitted repeated measures model at new design points using `random`.

For the properties and methods of this object, see the `RepeatedMeasuresModel` class page.

fitcsvm function for enhanced performance of support vector machines (SVMs) for binary classification

You can now use the new `fitcsvm` function to train an SVM classifier for one- or two-class learning. `fitcsvm` creates an object of the new class `ClassificationSVM` or existing class `ClassificationPartitionedModel`.

`ClassificationSVM` is a new class for accessing and performing operations on the training data. `CompactClassificationSVM` is a new class for storing configurations of trained models without storing training data. The syntax and methods resemble those in the existing `ClassificationTree` and `CompactClassificationTree` classes.

The new `fitcsvm` function and `ClassificationSVM` and `CompactClassificationSVM` classes include the functionality of the `svmtrain` and `svmclassify` functions. `ClassificationSVM` provides several benefits compared to the `svmtrain` and `svmclassify` functions:

- The new functionality
  - Supports computation of soft classification scores
  - Supports fitting posterior probabilities
  - Has improved training speed, especially on big data with well-separated classes by providing shrinkage
  - Allows a warm restart by accepting an initial α value
  - Allows training to resume after the maximum number of iterations is exceeded
  - Supports robust learning in the presence of outliers

- `ClassificationSVM` is built on the same framework as `ClassificationTree`, `ClassificationDiscriminant`, and `ClassificationKNN`, so you have a variety of options and methods, including:
  - Cross validation
  - Resubstitution statistics
  - Generalization statistics
  - Weighted classification

For all methods and properties of the new objects, see the `ClassificationSVM` and `CompactClassificationSVM` class pages.

evalclusters methods to expand the number of clusters and number of gap criterion simulations

There are two new methods for the objects created using the `evalclusters` function:

- `addK` adds additional number of clusters to be evaluated. This method applies to all classes of cluster evaluation (i.e., `clustering.evaluation.GapEvaluation`, `clustering.evaluation.SilhouetteEvaluation`, `clustering.evaluation.CalinskiHarabaszEvaluation`, and `clustering.evaluation.DaviesBouldinEvaluation`).
- `increaseB` increases the number of reference data sets for gap criterion simulations. This method applies to the `clustering.evaluation.GapEvaluation` class.

The default value of the `'SearchMethod'` name-value pair argument for `clustering.evaluation.GapEvaluation` objects is now always `'globalMaxSE'`.

Compatibility Considerations

The default value of the `'SearchMethod'` name-value pair argument for `clustering.evaluation.GapEvaluation` objects is now always `'globalMaxSE'` and does not change depending on the value of the `'KList'` name-value pair argument.

p-value output from the multcompare function

`multcompare` now returns the *p*-value of each pairwise comparison of group means. `multcompare` returns the *p*-value in the sixth column of its first output argument. The *p*-value is the overall significance level at which the individual comparison is borderline significant.

Compatibility Considerations

The first output argument of `multcompare` now has six columns, instead of five. The sixth column contains the *p*-value.

mnrfit, lassoglm, and fitglm functions accept categorical variables as responses

`mnrfit` now accepts a categorical variable as the response. The `lassoglm`, `fitglm`, and `glmfit` functions now accept a two-level categorical variable as the response. The `random` method for the `GeneralizedLinearModel` class now also returns categorical responses.

Functions accept table inputs as an alternative to dataset array inputs

The following functions and methods now accept table inputs as alternative to dataset array inputs.

Functions and Methods	Class
`fitlm`, `fitglm`, `fitlme`, `fitnlm`, `stepwiseglm`, `stepwiselm`, `grpstats`, `datasample`	N/A
`predict`, `random`, `feval`	`LinearModel`
`devianceTest`, `random`, `predict`, `feval`	`GeneralizedLinearModel`
`random`, `predict`, `feval`	`NonLinearModel`
`random`, `predict`	`LinearMixedModel`

Functions and model properties return a table rather than a dataset array

The following functions, methods, and model properties now return a `table` rather than a `dataset` array.

Functions and Methods	Class
`xptread`, `grpstats`*	N/A
`anova`	`LinearModel`
`devianceTest`	`GeneralizedLinearModel`
`fixedEffects`, `randomEffects`	`LinearMixedModel`

Property	Class
`VariableInfo`, `ObservationInfo`, `Variables`, `Diagnostics`, `Residuals`, `Coefficients`	`LinearModel`
`VariableInfo`, `ObservationInfo`, `Variables`, `Diagnostics`, `Residuals`, `Fitted`, `Coefficients`	`GeneralizedLinearModel`
`VariableInfo`, `ObservationInfo`, `Variables`, `Diagnostics`, `Residuals`, `Coefficients`	`NonLinearModel`
`VariableInfo`, `ObservationInfo`, `Variables`, `Coefficients`, `ModelCriterion`	`LinearMixedModel`

*`grpstats` now matches the output with input type.

Compatibility Considerations

The functions and properties listed now return a `table` instead of a `dataset` array. You can convert them to dataset arrays using the `table2dataset` function.

Default value of 'EmptyAction' on kmeans is now 'singleton'.

The default value of the `'EmptyAction'` name-value pair argument of the `kmeans` function is now `'singleton'`.

Compatibility Considerations

To set the value of `'EmptyAction'` to `'error'`, you must explicitly specify `'EmptyAction','error'`.

Functions for classification methods and clustering

The following are new functions for classification and regression trees, discriminant analysis, nearest neighbors, Naive Bayes classification, and Gaussian mixture models.

New Function	Replacing
`fitcdiscr`	`ClassificationDiscriminant.fit`
`fitcknn`	`ClassificationKNN.fit`
`fitctree`	`ClassificationTree.fit`
`fitrtree`	`RegressionTree.fit`
`fitNaiveBayes`	`NaiveBayes.fit`
`fitgmdist`	`gmdistribution.fit`
`templateDiscriminant`	`ClassificationDiscriminant.template`
`templateKNN`	`ClassificationKNN.template`
`templateTree`	`ClassificationTree.template` or `RegressionTree.template`
`makecdiscr`	`ClassificationDiscriminant.make`

Functionality being changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`ClassificationDiscriminant.fit`	Still runs	`fitcdiscr`	Replace instances of `ClassificationDiscriminant.fit` with `fitcdiscr`.
`ClassificationKNN.fit`	Still runs	`fitcknn`	Replace instances of `ClassificationKNN.fit` with `fitcknn`.

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`ClassificationTree.fit`	Still runs	`fitctree`	Replace instances of `ClassificationTree.fit` with `fitctree`.
`RegressionTree.fit`	Still runs	`fitrtree`	Replace instances of `RegressionTree.fit` with `fitrtree`.
`NaiveBayes.fit`	Still runs	`fitNaiveBayes`	Replace instances of `NaiveBayes.fit` with `fitNaiveBayes`.
`gmdistribution.fit`	Still runs	`fitgmdist`	Replace instances of `gmdistribution.fit` with `fitgmdist`.
`ClassificationDiscriminant.template`	Still runs	`templateDiscriminant`	Replace instances of `ClassificationDiscriminant.template` with `templateDiscriminant`.
`ClassificationKNN.template`	Still runs	`templateKNN`	Replace instances of `ClassificationKNN.template` with `templateKNN`.
`ClassificationTree.template` or `RegressionTree.template`	Still runs	`templateTree`	Replace instances of `ClassificationTree.template` or `RegressionTree.template` with `templateTree`.
`ClassificationDiscriminant.make`	Still runs	`makecdiscr`	Replace instances of `ClassificationDiscriminant.make` with `makecdiscr`.

R2013b

Version: 8.3

New Features

Bug Fixes

Linear mixed-effects models

`LinearMixedModel` is a new class for fitting linear mixed-effects (LME) models. Fit multi-level LME models or LME models with nested and/or crossed random effects using the `fitlme` or `fitlmematrix` function. You can:

- Specify LME models using either the formula notation or via matrix input.
- Fit LME models using maximum likelihood (ML) or restricted maximum likelihood (REML).
- Specify a covariance pattern for the random effects.
- Calculate estimates of best linear unbiased predictors (BLUPs) for random effects.
- Perform custom joint hypothesis tests on fixed and random effects.
- Compute confidence intervals on fixed effects, random effects, and covariance parameters.
- Examine residuals, diagnostic plots, fitted values, and design matrices.
- Compare two different models via theoretical or simulated likelihood ratio tests.
- Make predictions on new data using the fitted LME model.
- Generate random data using the fitted LME model at new design points.

For the properties and methods of this object, see the class page for `LinearMixedModel`.

Code generation for probability distribution and descriptive statistics functions (using MATLAB Coder)

Many probability distribution and descriptive statistics functions are now supported for code generation. For a full list of Statistics Toolbox™ functions that are supported by MATLAB Coder, see Statistics Toolbox Functions.

evalclusters function for estimating the optimal number of clusters in data

The new function `evalclusters` estimates the optimal number of clusters for various criterion values, and returns the clustering solution corresponding to the estimated optimal value.

You can provide clustering solutions, ask `evalclusters` to use one of the built-in clustering algorithms, `'kmeans'`, `'linkage'`, or `'gmdistribution'`, or provide a function handle.

The following criteria are available:

- The Calinski-Harabasz (CH) index
- The Silhouette index
- The Gap statistic
- The Davies-Bouldin (DB) index

mvregress function that now accepts a design matrix even if Y has multiple columns

`mvregress` now accepts an n-by-(p + 1) design matrix X, when the response Y is an n-by-d matrix with $d > 1$, where n is the number of observations, p is the number of predictor variables, d is the

number of dimensions in the response, and X includes a column of ones for the intercept (constant) term.

Upper tail probability calculations for cumulative distribution functions

Statistics Toolbox now provides upper tail probability calculations for cumulative distribution functions. You can compute the upper tail probabilities using a trailing `'upper'` argument in the following functions:

- `cdf` function for probability distribution objects, returned by `pd = makedist(distname)` or `pd = fitdist(X,distname)`:

  ```
  cdf(pd,X,'upper')
  ```

- `cdf` function:

  ```
  Y = cdf('name',X,A,'upper')

  Y = cdf('name',X,A,B,'upper')

  Y = cdf('name',X,A,B,C,'upper')
  ```

- Distribution-specific `cdf` functions:

Distribution	New Syntax
Beta	`p = betacdf(X,A,B,'upper')`
Binomial	`Y = binocdf(X,N,P,'upper')`
Chi-square	`p = chi2cdf(X,V,'upper')`
Extreme Value	`P = evcdf(X,mu,sigma,'upper')` `[P,PLO,PUP] = evcdf(X,mu,sigma,pcov,'upper')`
Exponential	`P = expcdf(X,mu,'upper')` `[P,PLO,PUP] = expcdf(X,mu,pcov,'upper')`
F	`P = fcdf(X,V1,V2,'upper')`
Gamma	`P = gamcdf(X,A,B,'upper')` `[P,PLO,PUP] = gamcdf(X,A,B,pcov,'upper')`
Geometric	`Y = geocdf(X,P,'upper')`
Generalized Extreme Value	`P = gevcdf(X,k,sigma,mu,'upper')`
Generalized Pareto	`P = gpcdf(X,sigma,theta,'upper')`
Hypergeometric	`P = hygecdf(X,M,K,N,'upper')`
Lognormal	`P = logncdf(X,mu,sigma,'upper')` `[P,PLO,PUP] = logncdf(X,mu,sigma,pcov,'upper')`

Distribution	New Syntax
Negative Binomial	`Y = nbincdf(X,R,P,'upper')`
Non-central F	`P = ncfcdf(X,NU1,NU2,DELTA,'upper')`
Non-central t	`P = nctcdf(X,NU,DELTA,'upper')`
Non-central Chi-square	`P = ncx2cdf(X,V,DELTA,'upper')`

Normal	`P = normcdf(X,mu,sigma,'upper')`
	
	`[P,PLO,PUP] = normcdf(X,mu,sigma,pcov,'upper')`

Poisson	`P = poisscdf(X,lambda,'upper')`
t	`P = tcdf(X,V,'upper')`
Rayleigh	`P = raylcdf(X,B,'upper')`
Uniform Discrete	`P = unidcdf(X,N,'upper')`
Uniform Continuous	`P = unidcdf(X,A,B,'upper')`

Weibull	`P = wblcdf(X,A,B,'upper')`
	
	`[P,PLO,PUP] = wblcdf(X,A,B,pcov,'upper')`

partialcorri function for partial correlation with asymmetric treatment of inputs and outputs

The new function `partialcorri` computes linear partial correlation coefficients with internal adjustments. You can compute partial correlation between pairs of variables in `Y` and `X`, adjusting for the remaining variables in `X`, or between pairs of variables in `Y` and `X`, adjusting for the remaining variables in `X`, after first controlling both `X` and `Y` for the variables in `Z`.

You can also:

- Specify whether to use Pearson or Spearman partial correlations.
- Specify how to handle missing values.
- Perform hypotheses test of zero correlation against a one-sided or two-sided alternative.

Fitting functions for linear, generalized linear, and nonlinear models

There are new functions for the fitting and stepwise algorithms of linear and generalized linear models, and the fitting algorithm of nonlinear models. The new functions are as follows.

New Function	Replacing
`fitlm`	`LinearModel.fit`
`stepwiselm`	`LinearModel.stepwise`
`fitglm`	`GeneralizedLinearModel.fit`
`stepwiseglm`	`GeneralizedLinearModel.stepwise`
`fitnlm`	`NonLinearModel.fit`

Functionality being changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`LinearModel.fit`	Still runs	`fitlm`	Replace instances of `LinearModel.fit` with `fitlm`
`LinearModel.stepwise`	Still runs	`stepwiselm`	Replace instances of `LinearModel.stepwise` with `stepwiselm`
`GeneralizedLinearModel.fit`	Still runs	`fitglm`	Replace instances of `GeneralizedLinearModel.fit` with `fitglm`
`GeneralizedLinearModel.stepwise`	Still runs	`stepwiseglm`	Replace instances of `GeneralizedLinearModel.stepwise` with `stepwiseglm`
`NonLinearModel.fit`	Still runs	`fitnlm`	Replace instances of `NonLinearModel.fit` with `fitnlm`

R2013a

Support vector machines (SVMs) for binary classification (formerly in Bioinformatics Toolbox)

Support vector machines are now in Statistics Toolbox. Train support vector machine classifier using `svmtrain` and classify data using `svmclassify`.

Probabilistic PCA and alternating least-squares algorithms for principal component analysis with missing data

Two new features handle missing data in principal component analysis:

- The new function `ppca` uses probabilistic principal components analysis, which is based on an isotropic error model.
- The function `pca` has a new alternating least squares (ALS) algorithm. Use the name-value pair argument `'algorithm'` with the value `'als'`.

Anderson-Darling goodness-of-fit test

The new function `adtest` performs the Anderson-Darling goodness-of-fit test. `adtest` can perform:

- Simple test: Test against a specific distribution with parameters specified. You can test against any continuous univariate parametric distribution.
- Composite test: Test against a specified distribution family (also called an omnibus test). You can test against the normal, exponential, extreme-value, lognormal, or weibull distribution families.

Decision-tree performance improvements and categorical predictors with many levels

- The training speed for decision trees and their ensembles is improved. The improvement is best seen in decision tree ensembles obtained using the `fitensemble` function or `TreeBagger` class.
- Improved efficiency of `TreeBagger` when used in parallel mode.
- You can specify the number of surrogate splits saved in decision trees using the `'surrogate'` name-value pair argument in the `fit` and `template` methods of the `ClassificationTree` and `RegressionTree` classes.
- `ClassificationTree.fit` and `ClassificationTree.template` provide several heuristic methods for splitting on categorical predictors with many levels. Use the `'AlgorithmForCategorical'` name-value pair argument to specify the algorithm to find the best split and the `'MaxCat'` name-value pair argument to specify the maximum number of categories you allow.

Grouping and kernel density options in scatterhist function

The `scatterhist` function has these name-value pair arguments:

- `'Group'` lets you specify a grouping variable and produces a grouped scatter plot.
- `'Kernel'` lets you use grouped kernel density plots instead of overall histograms for the marginal distributions.

- Additional options let you change colors, line properties, legends, and more.

Nonlinear model enhancements

These functions now accept additional error models and fixed or fit-dependent weights.

`NonLinearModel` methods: • `NonLinearModel.fit` • `predict` • `random`	• Use the `'ErrorModel'` and `'ErrorParameters'` name-value pair arguments to define the error models and `'Weights'` to enter weights. • The `NonLinearModel` object has a new property, `WeightedResiduals`.
`nlinfit`	• Use `'ErrorModel'` and `'ErrorParameters'` name-value pair arguments to define the error models and `'Weights'` name-value pair to enter weights. • `nlinfit` returns a structure containing information about the error model you define.
`nlpredci`	• Accepts the error model structure returned by `nlinfit`. • Adjusts Scheffe type simultaneous confidence intervals for weights, error models, and rank deficient Jacobians.

Additional functionality changes are:

- `disp` (`NonLinearModel` method) shows only estimable coefficients, and shows `NaN` for inestimable coefficients.
- Ftest (`NonLinearModel` method) automatically decides whether to compare the full model against an intercept-only model or zero.
- `NonLinearModel` properties such as `Diagnostics`, `Residuals`, `LogLikelihood`, `SSE`, and `SST` account for weights and error models.

Syntax changes in parametric hypothesis test functions

Parametric hypothesis test functions accept optional input arguments as name-value pair arguments.

`adtest`	Anderson-Darling goodness-of-fit test
`ansaribradley`	Ansari-Bradley test
`dwtest`	Durbin-Watson test
`kstest`	One-sample Kolmogorov-Smirnov test
`kstest2`	Two-sample Kolmogorov-Smirnov test
`lillietest`	Lilliefors test
`ttest`	One-sample *t*-test
`ttest2`	Two-sample *t*-test
`vartest`	One-sample variance chi-square test
`vartest2`	Two-sample variance *F*-test

`vartestn`	Variance test across multiple groups
`ztest`	*z*-test

Probability distribution enhancements

New probability distribution objects provide the following new functionality:

- Create a distribution without fitting to data using the new `makedist` function.
- Assign directly to parameter values.
- Create truncated distributions.
- Create and operate on arrays of distribution objects.
- Create custom distributions. To begin, use `dfittool` and select **Edit > Define Custom Distributions**. Use the provided template to define the `'Laplace'` distribution, or modify it to create your own.
- Compute and plot likelihood ratio confidence intervals and profile likelihood for fitted probability distributions.
- Additional distributions in the probability distribution framework:
  - Multinomial
  - Piecewise Linear
  - Triangular
  - Uniform

You can continue fitting distributions to data using the existing `fitdist` function.

Compatibility Considerations

The class names of probability distribution objects returned by `fitdist` are different than in earlier releases.

R2012b

Boosting algorithms for imbalanced data, sparse ensembles, and multiclass boosting, with self termination

There are three new boosting algorithms for classification:

- **RUSBoost** (boosting by random undersampling) for imbalanced data (data in which one class has many more observations than the other).
- **LPBoost** (linear programming) and **TotalBoost** (totally corrective boosting) which self-terminate, can lead to a sparse ensemble, and can be used for multiclass boosting.

Burr distribution for expressing a wide range of distribution shapes while preserving a single functional form for the density

There is a new probability distribution object for the Burr Type XII distribution, a three-parameter family of continuous distributions on the real line. Use `fitdist` to fit this distribution to data. Use `ProbDistUnivParam` to specify the distribution parameters directly. Either function produces a distribution you can use to generate random samples or compute functions such as `pdf` and `cdf`.

Data import to a dataset array with the MATLAB Import Tool

You can now import data from a file directly into a `dataset` array using the MATLAB Import Tool.

Principal component analysis enhancements for handling NaN as missing data, weighted PCA, and choosing between EIG or SVD as the underlying algorithm

The new `pca` function includes additional functionality for principal component analysis. Features of `pca` include:

- Handling of NaN as missing data values.
- Weighted principal component analysis with user-specified weights.
- Choice of SVD or EIG algorithm for computing principal components.
- Option to specify number of components to return.
- Option to not center before computing principal components.

Compatibility Considerations

The new `pca` function replaces the `princomp` function.

Speedup of k-means clustering using Parallel Computing Toolbox

Statistics Toolbox now supports parallel execution for `kmeans`.

One-sided nonparametric hypothesis tests

An option to test one-sided right- or left-tailed alternatives is available for these nonparametric hypothesis tests:

- `signrank`
- `ranksum`
- `signtest`

Reorder nodes in dendrogram plots

- The `dendrogram` function has new options for reordering the nodes of hierarchical binary cluster trees:
  - The `reorder` option allows you to specify a permutation vector for the order of nodes in a dendrogram plot.
  - The `checkcrossings` option checks whether a requested permutation vector leads to crossing branches in a dendrogram plot.
- The function `optimalleaforder` generates an optimal permutation of nodes.

Nonlinear model enhancements

You can add a vector of observation weights, or a handle to a function that returns a vector of observation weights, to these functions:

- `NonLinearModel.fit`.
- `predict` and `random` (`NonLinearModel` methods).
- `nlinfit` and `nlpredci`.

For an example of weighted fitting, see Weighted Nonlinear Regression.

Compatibility Considerations

Use either `Weights` or `RobustWgtFun` when performing weighted nonlinear regression.

Changes to LinearModel diagnostics

The diagnostics in the `Diagnostics` dataset array for `LinearModel` objects are in a new order, and no longer appear in the Variables editor. The new order is:

- `Leverage`
- `CooksDistance`
- `Dffits`
- `S2_i`
- `CovRatio`
- `Dfbetas`
- `HatMatrix`

Compatibility Considerations

To access the correct diagnostics, you should update any code that indexes the diagnostics dataset array columns by number.

Functionality being changed

Functionality	What Happens When You Use This Functionality?	Use This Instead	Compatibility Considerations
`princomp`	Still runs	`pca`	Replace instances of `princomp` with `pca`

R2012a

Linear, Generalized Linear, and Nonlinear Models for Regression

`LinearModel` is a new class for performing linear regression. `LinearModel.fit` creates a model that:

- Lets you fit models with both categorical and continuous predictor variables
- Contains information about the quality of the fit, such as residuals and ANOVA tables
- Lets you easily plot the fit
- Allows for automatic or manual exclusion of unimportant variables
- Enables robust fitting for reduced influence of outliers
- Lets you specify quadratic and other models using a symbolic formula
- Enables stepwise model selection

There are similar improvements for generalized linear and nonlinear modeling using the `GeneralizedLinearModel` and `NonLinearModel` classes. For details, see the class reference pages in the reference material, or Linear Regression, Stepwise Regression, Robust Regression — Reduce Outlier Effects, Generalized Linear Regression, or Nonlinear Regression in the User's Guide.

Variable Editor for Dataset Arrays

You can now edit, sort, plot, and select portions of dataset arrays from the MATLAB Variable Editor. For details, see Using Dataset Arrays in the User's Guide.

Lasso for Generalized Linear Regression

The `lassoglm` function regularizes generalized linear models. Use `lassoglm` to examine model alternatives and to constrain or remove redundant or unimportant variables in generalized linear regression. For details, see the function reference page, or Lasso Regularization of Generalized Linear Models in the User's Guide.

K-Nearest Neighbor Classification

`ClassificationKNN.fit` creates a classification model that performs *k*-nearest neighbor classification. You can check the quality of the model with cross validation or resubstitution. For details, see the `ClassificationKNN` page in the reference material, or Classification Using Nearest Neighbors in the User's Guide.

Random Subspace Ensembles

`fitensemble` can construct random subspace ensembles to improve the classification accuracy of both *k*-nearest neighbor classifiers and discriminant analysis classifiers. For details, see Ensemble Methods or Random Subspace Classification in the User's Guide.

Regularized Discriminant Analysis with Variable Selection

`ClassificationDiscriminant` models now have two parameters, `Gamma` and `Delta`, for regularization and lowering the number of variables. Set `Gamma` to regularize the discriminant. Set `Delta` to eliminate variables. Use `cvshrink` to obtain optimal `Gamma` and `Delta` parameters by

cross validation. For details, see the reference pages, or Regularize a Discriminant Analysis Classifier in the User's Guide.

stepwisefit Coefficient History

The `stepwisefit` function now returns the fitted coefficient history in the `history.B` field.

RobustWgtFun Replaces WgtFun

The `WgtFun` option is now called `RobustWgtFun` in the `nlinfit`, `statget`, and `statset` functions. `RobustWgtFun` also makes the `Robust` option superfluous.

Compatibility Considerations

The `WgtFun` and `Robust` options are currently accepted by all functions. To avoid potential future incompatibilities, update code that uses the `WgtFun` and `Robust` options to use the `RobustWgtFun` option.

ClassificationTree Now Predicts Class with Minimal Misclassification Cost

The `ClassificationTree` `predict` method now chooses the class with minimal expected misclassification cost. Previously, it chose the class with maximal posterior probability. The new behavior is consistent with the `cvLoss` method. Furthermore, both `ClassificationDiscriminant` and `ClassificationKNN` predict using minimal expected misclassification cost. For details, see `predict` and `loss`.

Compatibility Considerations

If you use a nondefault cost matrix, some `ClassificationTree` classification predictions can differ from those in previous versions.

fpdf Improvements

The `fpdf` function now accepts a wider range of parameter values, including `Inf`.

R2011b

Lasso Regularization for Linear Regression

The `lasso` function incorporates both the lasso regularization algorithm and the elastic net regularization algorithm. Use `lasso` to remove redundant or unimportant variables in linear regression. The `lassoPlot` function helps you visualize `lasso` results, with a variety of coefficient trace plots and a cross-validation plot.

For details, see Lasso and Elastic Net.

Discriminant Analysis Classification Object

You can now use the `ClassificationDiscriminant` and `CompactClassificationDiscriminant` classes for classification via discriminant analysis. The syntax and methods resemble those in the existing `ClassificationTree` and `CompactClassificationTree` classes. The `ClassificationDiscriminant` class includes the functionality of the `classify` function. `ClassificationDiscriminant` provides several benefits compared to the `classify` function:

- After you fit a classifier, you can predict without refitting.
- `ClassificationDiscriminant` is built on the same framework as `ClassificationTree`, so you have a variety of options and methods, including:
  - Cross validation
  - Resubstitution statistics
  - A choice of cost functions
  - Weighted classification
- `ClassificationDiscriminant` can fit several models, including linear, quadratic, and linear or quadratic with pseudoinverse.

For details, see Discriminant Analysis.

Nearest Neighbor Searching for Points Within a Fixed Distance

The `rangesearch` function finds all members of a data set that are within a specified distance of members of another data set. As with the `knnsearch` function, you can set a variety of distance metrics, or program your own. `rangesearch` has counterparts that are methods of the `ExhaustiveSearcher` and `KDTreeSearcher` classes.

datasample Function for Random Sampling

The `datasample` function samples with or without replacement from a data set. It can also perform weighted sampling, with or without replacement.

Fractional Factorial Design Improvements

The `fracfactgen` function now allows up to 52 factors, instead of the previous limit of 26 factors. Specify factors as case-sensitive strings, using `'a'` through `'z'` for the first 26 factors, and `'A'` through `'Z'` for the remaining factors.

`fracfact` now checks for an arbitrary level of interaction in confounding, instead of the previous limit of confounding up to products of two factors. Set the `MaxInt` name-value pair to the level of interaction you want. You can also set names for the factors using the `FactorNames` name-value pair.

nlmefit Returns the Covariance Matrix of Estimated Coefficients

The `nlmefit` function now returns the covariance matrix of the estimated coefficients as the `covb` field of the `stats` structure.

signrank Change

The `signrank` test now defines ties to be entries that differ by `2*eps` or less. Previously, ties were entries that were identical to machine precision.

Conversion of Error and Warning Message Identifiers

For R2011b, error and warning message identifiers have changed in Statistics Toolbox.

Compatibility Considerations

If you have scripts or functions that use message identifiers that changed, you must update the code to use the new identifiers. Typically, message identifiers are used to turn off specific warning messages, or in code that uses a `try`/`catch` statement and performs an action based on a specific error identifier.

For example, if you use the `'resubstitution'` method, the `'stats:plsregress:InvalidMCReps'` identifier has changed to `'stats:plsregress:InvalidResubMCReps'`. If you use the `'resubstitution'` method and your code checks for `'stats:plsregress:InvalidMCReps'`, you must update it to check for `'stats:plsregress:InvalidResubMCReps'` instead.

To determine the identifier for a warning, run the following command just after you see the warning:

```
[MSG,MSGID] = lastwarn;
```

This command saves the message identifier to the variable `MSGID`.

To determine the identifier for an error, run the following command just after you see the error:

```
exception = MException.last;
MSGID = exception.identifier;
```

Tip Warning messages indicate a potential issue with your code. While you can turn off a warning, a suggested alternative is to change your code so it runs warning free.

R2011a

Version: 7.5

New Features

Bug Fixes

Boosted Decision Trees for Classification and Regression

The new `fitensemble` function constructs ensembles of decision trees. It provides:

- Several popular boosting algorithms (`AdaBoostM1`, `AdaBoostM2`, `GentleBoost`, `LogitBoost`, and `RobustBoost`) for classification
- Least-squares boosting (`LSBoost`) for regression
- Most `TreeBagger` functionality for ensembles of bagged decision trees

There is also an improved interface for classification trees (`ClassificationTree`) and regression trees (`RegressionTree`), encompassing the functionality of `classregtree`.

For details, see Ensemble Methods.

Memory and Performance Improvements in Linkage Methods

The `linkage` and `clusterdata` functions have a new `savememory` option that can use less memory than before. With `savememory` set to `'on'`, the functions do not build a pairwise distance matrix, so use less memory and, depending on problem size, can use less time. You can use the `savememory` option when:

- The linkage `method` is `'ward'`, `'centroid'`, or `'median'`
- The linkage distance `metric` is `'euclidean'` (default)

For details, see the `linkage` and `clusterdata` function reference pages.

Conditional Weighted Residuals and Derivative Step Control in nlmefit and nlmefitsa

The `nlmefit` and `nlmefitsa` functions now provide the conditional weighted residuals of the fit. Use this information to assess the quality of the model; see Example: Examining Residuals for Model Verification.

The `statset` `Options` structure now includes `'DerivStep'`, which enables you to set finite differences for gradient estimation.

Detecting Ties in k-Nearest Neighbor Search

`knnsearch` now optionally returns all *k*th nearest neighbors of points, instead of just one. The `knnsearch` methods for `ExhaustiveSearcher` and `KDTreeSearcher` also have this option.

Distribution Fitting Tool Uses fitdist Function

MATLAB functions generated with the Distribution Fitting Tool now use the `fitdist` function to create fitted probability distribution objects. The generated functions return probability distribution objects as output arguments.

Speed and Accuracy Improvements in Noncentral Chi-Square CDF

`ncx2cdf` is now faster and more accurate for large values of the noncentrality parameter.

Perfect Separation in Binomial Regression

If the two categories in a binomial regression model (such as `logit` or `probit`) are perfectly separated, the best-fitting model is degenerate with infinite coefficients. In this case, the `glmfit` function is likely to exceed its iteration limit. `glmfit` now tries to detect this perfect separation and display a diagnostic message.

Sign Convention in mdscale

`mdscale` now enforces that, in each column of the output `Y`, the value with the largest magnitude has a positive sign. This change makes results consistent across releases and platforms—small changes used to lead to sign reversals.

Demo of Credit Rating Classification Via Bagged Decision Trees

The credit rating demo that used to be exclusively in Financial Toolbox™ is now available in Statistics Toolbox. The demo uses bagged decision trees for classifying creditworthiness.

To view the demo at the MATLAB command line, enter:

```
showdemo creditratingdemo
```

R2010b

Parallel Computing Support for More Functions

Statistics Toolbox now supports parallel execution for the following functions:

- `candexch`
- `cordexch`
- `daugment`
- `dcovary`
- `nnmf`
- `plsregress`
- `rowexch`
- `sequentialfs`

For more information, see the Parallel Statistics chapter in the User's Guide.

Algorithm to Rank Features in Classification and Regression

New filter algorithm, `relieff`, is based on nearest neighbors. The ReliefF algorithm accounts for correlations among predictors by computing the effect of every predictor on the class label (or true response for regression) locally and then integrates these local estimates over the entire predictor space.

nlmefit Support for Error Models, and nlmefitsa changes

`nlmefit` now supports the following error models:

- `combined`
- `constant`
- `exponential`
- `proportional`

You can specify an error model with both `nlmefitsa` and `nlmefit`.

The `nlmefit` `bic` calculation has changed. Now the degrees of freedom value is based on the number of groups rather than the number of observations. This conforms with the `bic` definition used by the `nlmefitsa` function.

Both `nlmefit` and `nlmefitsa` now store the estimated error parameters in the `errorparm` field of the output `stats` structure. The `rmse` field of the structure now contains the root mean squared residual for all error models; this value is computed on the log scale for the `exponential` model.

Compatibility Considerations

In the previous release, the `rmse` field was used by `nlmefitsa` for both mean squared residual and the estimated error parameter. Change your code, if necessary, to address the appropriate field in the `stats` structure.

As described in “nlmefit Support for Error Models, and nlmefitsa changes” on page 26-2, `nlmefit` now calculates different `bic` values than in previous releases.

Surrogate Splits for Decision Trees

The new surrogate splits feature in `classregtree` allows for better handling of missing values, more accurate estimation of variable importance, and calculation of the predictive measure of association between variables.

New Bagged Decision Tree Properties

`TreeBagger` and `CompactTreeBagger` classes have two new properties:

- `NVarSplit` provides the number of decision splits for each predictor variable.
- `VarAssoc` provides a measure of association between pairs of predictor variables.

Enhanced Cluster Analysis Performance

The `linkage` function has improved performance for the `centroid`, `median`, and `single` linkage methods.

The `linkage` and `pdist` hierarchical cluster analysis functions support larger array dimensions with 64-bit platforms, so can handle larger problems.

Export Probability Objects with dfittool

The distribution fitting GUI (`dfittool`) now allows you to export fits to the MATLAB workspace as probability distribution fit objects. For more information, see Modeling Data Using the Distribution Fitting Tool.

Compatibility Considerations

If you load a distribution fitting session that was created with previous versions of Statistics Toolbox, you cannot save an existing fit. Fit the distribution again to enable saving.

Compute Partial Correlation of Two Variables Correcting for All Other Variables

`partialcorr` now accepts a new syntax, `RHO = partialcorr(X)`, which returns the sample linear partial correlation coefficients between pairs of variables in `X`, controlling for the remaining variables in `X`. For more information, see the function reference page.

Specify Number of Evenly Spaced Quantiles

`quantile` now accepts a new syntax, `Y = quantile(X,N,...)`, which returns quantiles at the cumulative probabilities (1:`N`)/(`N`+1) where `N` is a scalar positive integer value.

Control Location and Orientation of Marginal Histograms with scatterhist

`scatterhist` now accepts three parameter name/value pairs that control where and how the histogram plots appear. The new parameter names are `NBins`, `Location`, and `Direction`. For more information, see the function reference page.

Return Bootstrapped Statistics with bootci

`bootci` has a new output option which returns the bootstrapped statistic computed for each of the `NBoot` bootstrap replicate samples. For more information, see the function reference page.

R2010a

Version: 7.3

New Features

Bug Fixes

Stochastic Algorithm Functionality in NLME Models

New stochastic algorithm for fitting NLME models is more robust with respect to starting values, enables parameter transformations, and relaxes assumption of constant error variance. See `nlmefitsa`.

k-Nearest Neighbor Searching

New functions for *k*-Nearest Neighbor (*k*NN) search efficiently to find the closest points to any query point. For information, see k-Nearest Neighbor Search and Radius Search.

Confidence Intervals Option in perfcurve

A new option in the `perfcurve` function computes confidence intervals for classifier performance curves.

Observation Weights Options in Resampling Functions

New options to weight resampling probabilities broaden the range of models supported by `bootstrp`, `bootci`, and `perfcurve` functions.

R2009b

New Parallel Computing Support for Certain Functions

Statistics Toolbox now supports parallel execution for the following functions:

- `bootci`
- `bootstrp`
- `crossval`
- `jackknife`
- `TreeBagger`

For more information on parallel computing in the Statistics Toolbox, see Parallel Computing Support for Resampling Methods.

New Stack and Unstack Methods for Dataset Arrays

`dataset.unstack` converts a “tall” dataset array to an equivalent dataset array that is in "wide format", by "unstacking" a single variable in the tall dataset array into multiple variables in wide. `dataset.stack` reverses this manipulation by converting a “wide” dataset array to an equivalent dataset array that is in "tall format", by "stacking up" multiple variables in the wide dataset array into a single variable in tall.

New Support for SAS Transport (.xpt) Files

Statistics Toolbox now supports importing and exporting files in SAS Transport (.xpt) format. For more information, see the `xptread` and `dataset.export` reference pages.

New Output Function in nlmefit for Monitoring or Canceling Calculations

The `nlmefit` function now supports using an output function to monitor or cancel calculations. For more information, see the `nlmefit` reference page.

R2009a

Enhanced Dataset Functionality

- An enhanced `dataset.join` method provides additional types of join operations:
  - `join` can now perform more complicated inner and outer join operations that allow a many-to-many correspondence between dataset arrays `A` and `B`, and allow unmatched observations in either `A` or `B`.
  - `join` can be of `Type` `'inner'`, `'leftouter'`, `'rightouter'`, `'fullouter'`, or `'outer'` (which is a synonym for `'fullouter'`). For an inner join, the dataset array, `C`, only contains observations corresponding to a combination of key values that occurred in both `A` and `B`. For a left (or right) outer join, `C` also contains observations corresponding to keys in `A` (or `B`) that did not match any in `B` (or `A`).
  - `join` can now return index vectors indicating the correspondence between observations in `C` and those in `A` and `B`.
  - `join` now supports using multiple keys.
  - `join` now supports an optional parameter for specifying missing key behavior rather than raising an error.
- An enhanced `dataset.export` method now supports exporting directly to Microsoft® Excel® files.

New Naïve Bayes Classification

- The `NaiveBayes` classification object is suitable for data sets that contain many predictors or features.
- It supports normal, kernel, multinomial, and multivariate multinomial distributions.

New Ensemble Methods for Classification and Regression Trees

- New classification objects, `TreeBagger` and `CompactTreeBagger`, provide improved performance through bootstrap aggregation (bagging).
- Includes Breiman's "random forest" method.
- Enhanced `classregtree` has more options for growing and pruning trees.

New Performance Curve Function

- New `perfcurve` function provides graphical method to evaluate classification results.
- Includes ROC (receiver operating characteristic) and other curves.

New Probability Distribution Objects

- Provides a consistent interface for working with probability distributions.
- Can be created directly using the `ProbDistUnivParam` constructor, or fit to data using the `fitdist` function.
- Option to fit distributions by group.
- Includes kernel object methods and parametric object methods that you can use to analyze the distribution represented by the object.

- Includes kernel object properties and parametric object properties that you can access to determine the fit results and evaluate their accuracy.
- Related enhancements in the `chi2gof`, `histfit`, `kstest`, `probplot`, and `qqplot` functions.

R2008b

Version: 7.0

New Features

Compatibility Considerations

Classification

The new `confusionmat` function tabulates misclassifications by comparing known and predicted classes of observations.

Data Organization

Dataset arrays constructed by the `dataset` function can now be written to an external text file using the new `export` function.

When reading external text files into a dataset array, `dataset` has a new `'TreatAsEmpty'` parameter for specifying strings to be treated as empty.

Compatibility Considerations

In previous versions, `dataset` used `eval` to evaluate strings in external text files before writing them into a dataset array. As a result, strings such as `'1/1/2008'` were treated as numerical expressions with two divides. Now, `dataset` treats such expressions as strings, and writes a string variable into the dataset array whenever a column in the external file contains a string that does not represent a valid scalar value.

Model Assessment

The cross-validation function, `crossval`, has new options for directly specifying loss functions for mean-squared error or misclassification rate, without having to provide a separate function M-file.

Multivariate Methods

The `procrustes` function has new options for computing linear transformations without scale or reflection components.

Probability Distributions

The multivariate normal functions `mvnpdf`, `mvncdf`, and `mvnrnd` now accept vector specification of diagonal covariance matrices, with corresponding gains in computational efficiency.

The hypergeometric distribution has been added to both the `disttool` and `randtool` graphical user interfaces.

Compatibility Considerations

The `ksdensity` function may give different answers for the case where there are censoring times beyond the last observed value. In this case, `ksdensity` tries to reduce the bias in its density estimate by folding kernel functions across a folding point so that they do not extend into the area that is completely censored. Two things have changed for this release:

1. In previous releases the folding point was the last observed value. In this release it is the first censoring time after the last observed value.

2 The folding procedure is applied not just when the `'function'` parameter is `'pdf'`, but for all `'function'` values.

Regression Analysis

The new `nlmefit` function fits nonlinear mixed-effects models to data with both fixed and random sources of variation. Mixed-effects models are commonly used with data over multiple groups, where measurements are correlated within groups but independent between groups.

Statistical Visualization

The `boxplot` function has new options for handling multiple grouping variables and extreme outliers.

The `lsline`, `gline`, `refline`, and `refcurve` functions now work with scatter plots produced by the `scatter` function. In previous versions, these functions worked only with scatter plots produced by the `plot` function.

The following visualization functions now have custom data cursors, displaying information such as observation numbers, group numbers, and the values of related variables:

- `andrewsplot`
- `biplot`
- `ecdf`
- `glyphplot`
- `gplotmatrix`
- `gscatter`
- `normplot`
- `parallelcoords`
- `probplot`
- `qqplot`
- `scatterhist`
- `wblplot`

Compatibility Considerations

Changes to `boxplot` have altered a number of default behaviors:

- Box labels are now drawn as text objects rather than tick labels. Any code that customizes the box labels by changing tick marks should now set the tick locations as well as the tick labels.
- The function no longer returns a handles array with a fixed number handles, and the order and meaning of the handles now depends on which options are selected. To locate a handle of interest, search for its `'Tag'` property using `findobj`. `'Tag'` values for box plot components are listed on the `boxplot` reference page.
- There are now valid handles for outliers, even when boxes have no outliers. In previous releases, the handles array returned by the function had `NaN` values in place of handles when boxes had no outliers. Now the `'xdata'` and `'ydata'` for outliers are `NaN` when there are no outliers.

- For small groups, the `'notch'` parameter sometimes produces notches that extend outside of the box. In previous releases, the notch was truncated to the extent of the box, which could produce a misleading display. A new value of `'markers'` for this parameter avoids the display issue.

As a consequence, the `anova1` function, which displays notched box plots for grouped data, may show notches that extend outside the boxes.

Utility Functions

The statistics options structure created by `statset` now includes a `Jacobian` field to specify whether or not an objective function can return the Jacobian as a second output.

R2008a

Descriptive Statistics

Bootstrap confidence intervals computed by `bootci` are now more accurate for lumpy data.

Compatibility Considerations

The formula for `bootci` confidence intervals of type `'bca'` or `'cper'` involves the proportion of bootstrap statistics less than the observed statistic. The formula now takes into account cases where there are many bootstrap statistics exactly equal to the observed statistic.

Model Assessment

Two new cross-validation functions, `cvpartition` and `crossval`, partition data and assess models in regression, classification, and clustering applications.

Multivariate Methods

A new sequential feature selection function, `sequentialfs`, selects predictor subsets that optimize user-defined prediction criteria.

The new `nnmf` function performs nonnegative matrix factorization (NMF) for dimension reduction.

Probability Distributions

The new `sobolset` and `haltonset` functions produce quasi-random point sets for applications in Monte Carlo integration, space-filling experimental designs, and global optimization. Options allow you to skip, leap over, and scramble the points. The `qrandstream` function provides corresponding quasi-random number streams for intermittent sampling.

Regression Analysis

The new `plsregress` function performs partial least-squares regression for data with correlated predictors.

Statistical Visualization

The `normspec` function now shades regions of a normal density curve that are either inside or outside specification limits.

Utility Functions

The statistics options structure created by `statset` now includes fields for `TolTypeFun` and `TolTypeX`, to specify tolerances on objective functions and parameter values, respectively.

R2007b

Cluster Analysis

The new `gmdistribution` class represents Gaussian mixture distributions, where random points come from different multivariate normal distributions with certain probabilities. The `gmdistribution` constructor creates mixture models with specified means, covariances, and mixture proportions, or by fitting a mixture model with a specified number of components to data. Methods for the class include:

- `fit` — Distribution fitting function
- `pdf` — Probability density function
- `cdf` — Cumulative distribution function
- `random` — Random number generator
- `cluster` — Data clustering
- `posterior` — Cluster posterior probabilities
- `mahal` — Mahalanobis distance

The `cluster` function for hierarchical clustering now accepts a vector of cutoff values, and returns a matrix of cluster assignments, with one column per cutoff value.

Compatibility Considerations

The `kmeans` function now returns a vector of cluster indices of length *n*, where *n* is the number of rows in the input data matrix `X`, even when `X` contains `NaN` values. In the past, rows of `X` with `NaN` values were ignored, and the vector of cluster indices was correspondingly reduced in size. Now the vector of cluster indices contains `NaN` values where rows have been ignored, consistent with other toolbox functions.

Design of Experiments

A new option in the *D*-optimal design function `candexch` specifies fixed design points in the row-exchange algorithm. A similar feature is already available for the `daugment` function, which uses the coordinate-exchange algorithm.

Hypothesis Tests

The `kstest` function now uses a more accurate method to calculate the *p*-value for a single-sample Kolmogorov-Smirnov test.

Compatibility Considerations

`kstest` now compares the computed *p*-value to the desired cutoff, rather than comparing the test statistic to a table of values. Results may differ from those in previous releases, especially for small samples in two-sided tests where an asymptotic formula was used in the past.

Probability Distributions

A new fitting function, `copulafit`, has been added to the family of functions that describe dependencies among variables using copulas. The function fits parametric copulas to data, providing a link between models of marginal distributions and models of data correlations.

A number of probability functions now have improved accuracy, especially for extreme parameter values. The functions are:

- `betainv` — More accurate for probabilities in `P` near 1.
- `binocdf` — More efficient and less likely to run out of memory for large values in `X`.
- `binopdf` — More accurate when the probabilities in `P` are on the order of `eps`.
- `fcdf` — More accurate when the parameter ratios `V2./V1` are much less than the values in `X`.
- `ncx2cdf` — More accurate in some extreme cases that previously returned `0`.
- `poisscdf` — More efficient and less likely to run out of memory for large values in `X`.
- `tcdf` — More accurate when the squares of the values in `X` are much less than the parameters in `V`.
- `tinv` — More accurate when the probabilities in `P` are very close to 0.5 and the outputs are very small in magnitude.

Function-style syntax for `paretotails` objects has been removed.

Compatibility Considerations

The changes to the probability functions listed above may lead to different, but more accurate, outputs than in previous releases.

In previous releases, syntax of the form `obj(x)` for a `paretotails` objects `obj` invoked the `cdf` method. This syntax now produces a warning. To evaluate the cumulative distribution function, use the syntax `cdf(obj,x)`.

Regression Analysis

The new `corrcov` function converts a covariance matrix to the corresponding correlation matrix.

The `mvregress` function now supports an option to force the estimated covariance matrix to be diagonal.

Compatibility Considerations

In previous releases the `mvregress` function, when using the `'cwls'` algorithm, estimated the covariance of coefficients `COVB` using the estimated, rather than the initial, covariance of the responses `SIGMA`. The initial `SIGMA` is now used, and `COVB` differs to a degree dependent on the difference between the initial and final estimates of `SIGMA`.

Statistical Visualization

The `boxplot` function has a new `'compact'` plot style suitable for displaying large numbers of groups.

R2007a

Data Organization

New categorical and dataset arrays are available for organizing and processing statistical data.

- Categorical arrays facilitate the use of nominal and ordinal categorical data.
- Dataset arrays provide a natural way to encapsulate heterogeneous statistical data and metadata, so that it can be accessed and manipulated using familiar methods analogous to those for numerical matrices.
- Categorical and dataset arrays are supported by a variety of new functions for manipulating the encapsulated data.
- Categorical arrays are now accepted as input arguments in all Statistics Toolbox functions that make use of grouping variables.

Hypothesis Testing

Expanded options are available for linear hypothesis testing.

- The new `linhyptest` function performs linear hypothesis tests on parameters such as regression coefficients. These tests have the form `H*b = c` for specified values of `H` and `c`, where `b` is a vector of unknown parameters.
- The `covb` output from `regstats` and the `SIGMA` output from `nlinfit` are suitable for use as the covariance matrix input argument required by `linhyptest`. The following functions have been modified to return a `covb` output for use with `linhyptest`: `coxphfit`, `glmfit`, `mnrfit`, `robustfit`.

Multivariate Statistics

The new `cholcov` function computes a Cholesky-like decomposition of a covariance matrix, even if the matrix is not positive definite. Factors are useful in many of the same ways as Cholesky factors, such as imposing correlation on random number generators.

The `classify` function for discriminant analysis has been improved.

- The function now computes the coefficients of the discriminant functions that define boundaries between classification regions.
- The output of the function is now of the same type as the input grouping variable `group`.

Compatibility Considerations

The `classify` function now returns outputs of different type than it did in the past. If the input argument `group` is a logical vector, output is now converted to a logical vector. In the past, output was returned as a cell array of `0`s and `1`s. If `group` is numeric, the output is now converted to the same type. For example, if `group` is of type `uint8`, the output will be of type `uint8`.

Probability Distributions

New `paretotails` objects are available for modeling distributions with an empirical cdf or similar distribution in the center and generalized Pareto distributions in the tails.

- The `paretotails` function converts a data sample to a `paretotails` object. The objects are useful for generating random samples from a distribution similar to the data, but with tail behavior that is less discrete than the empirical distribution.
- Objects from the `paretotails` class are supported by a variety of new methods for working with the piecewise distribution.
- The `paretotails` class provides function-like behavior, so that `p(x)` evaluates the cdf of `p` at values `x`.

Regression Analysis

The new `mvregresslike` function is a utility related to the `mvregress` function for fitting regression models to multivariate data with missing values. The new function computes the objective (log likelihood) function, and can also compute the estimated covariance matrix for the parameter estimates.

New `classregtree` objects are available for creating and analyzing classification and regression trees.

- The `classregtree` function fits a classification or regression tree to training data. The objects are useful for predicting response values from new predictors.
- Objects from the `classregtree` class are supported by a variety of new methods for accessing information about the tree.
- The `classregtree` class provides function-like behavior, so that `t(X)` evaluates the tree `t` at predictor values in `X`.
- The following functions now create or operate on objects from the new `classregtree` class: `treefit`, `treedisp`, `treeval`, `treefit`, `treeprune`, `treetest`.

Compatibility Considerations

Objects from the `classregtree` class are intended to be compatible with the structure arrays that were produced in previous versions by the classification and regression tree functions listed above. In particular, `classregtree` supports dot indexing of the form `t.property` to obtain properties of the object `t`. The class also provides function-like behavior through parenthesis indexing, so that `t(x)` uses the tree `t` to classify or compute fitted values for predictors `x`, rather than index into `t` as a structure array as it did in the past. As a result, cell arrays should now be used to aggregate `classregtree` objects.

Statistical Visualization

The new `scatterhist` function produces a scatterplot of 2D data and illustrates the marginal distributions of the variables by drawing histograms along the two axes. The function is also useful for viewing properties of random samples produced by functions such as `copularnd`, `mvnrnd`, and `lhsdesign`.

Other Improvements

- The `mvtrnd` function now produces a single random sample from the multivariate *t* distribution if the `cases` input argument is absent.

- The `zscore` function, which centers and scales input data by mean and standard deviation, now returns the means and standard deviations as additional outputs.

R2006b

Demos

The following demo has been updated:

- Selecting a Sample Size — Modified to highlight the new `sampsizepwr` function

Design of Experiments

The following visualization functions, commonly used in the design of experiments, have been added:

- `interactionplot` — Two-factor interaction plot for the mean
- `maineffectsplot` — Main effects plot for the mean
- `multivarichart` — Multivari chart for the mean

Hypothesis Tests

The following functions for hypothesis testing have been added or improved:

- `jbtest` — Replaces the chi-square approximation of the test statistic, which is asymptotic, with a more accurate algorithm that interpolates *p*-values from a table of quantiles. A new option allows you to run Monte Carlo simulations to compute *p*-values outside of the table.
- `lillietest` — Uses an improved version of Lilliefors' table of quantiles, covering a wider range of sample sizes and significance levels, with more accurate values. New options allow you to test for exponential and extreme value distributions, as well as normal distributions, and to run Monte Carlo simulations to compute *p*-values outside of the tables.
- `runstest` — Adds a test for runs up and down to the existing test for runs above or below a specified value.
- `sampsizepwr` — New function to compute the sample size necessary for a test to have a specified power. Options are available for choosing a variety of test types.

Compatibility Considerations

If the significance level for a test lies outside the range of tabulated values, [0.001, 0.5], then both `jbtest` and `lillietest` now return an error. In previous versions, `jbtest` returned an approximate *p*-value and `lillietest` returned an error outside a smaller range, [0.01, 0.2]. Error messages suggest using the new Monte Carlo option for computing values outside the range of tabulated values.

If the data sample for a test leads to a *p*-value outside the range of tabulated values, then both `jbtest` and `lillietest` now return, with a warning, either the smallest or largest tabulated value. In previous versions, `jbtest` returned an approximate *p*-value and `lillietest` returned `NaN`.

Multinomial Distribution

The multinomial distribution has been added to the list of almost 50 probability distributions supported by the toolbox.

- `mnpdf` — Multinomial probability density function
- `mnrnd` — Multinomial random number generator

Regression Analysis

Multinomial Regression

Support has been added for multinomial regression modeling of discrete multi-category response data, including multinomial logistic regression. The following new functions supplement the regression models in `glmfit` and `glmval` by providing for a wider range of response values:

- `mnrfit` — Fits a multinomial regression model to data
- `mnrval` — Computes predicted probabilities for the multinomial regression model

Multivariate Regression

The new `mvregress` function carries out multivariate regression on data with missing response values. An option allows you to specify how missing data is handled.

Survival Analysis

`coxphfit` — A new option allows you to specify the values at which the baseline hazard is computed.

Statistical Process Control

The following new functions consolidate and expand upon existing functions for statistical process control:

- `capability` — Computes a wider range of probabilities and capability indices than the `capable` function found in previous releases
- `controlchart` — Displays a wider range of control charts than the `ewmaplot`, `schart`, and `xbarplot` functions found in previous releases
- `controlrules` — Supplements the new `controlchart` function by providing for a wider range of control rules (Western Electric and Nelson)
- `gagerr` — Performs a gage repeatability and reproducibility study on measurements grouped by operator and part

Compatibility Considerations

The `capability` function subsumes the `capable` function that appeared in previous versions of Statistics Toolbox software, and the `controlchart` function subsumes the functions `ewmaplot`, `schart`, and `xbarplot`. The older functions remain in the toolbox for backwards compatibility, but they are no longer documented or supported.

R2006a

Version: 5.2

New Features

Bug Fixes

Analysis of Variance

Support for nested and continuous factors has been added to the `anovan` function for *N*-way analysis of variance.

Bootstrapping

The following functions have been added to supplement the existing `bootstrp` function for bootstrap estimation:

- `bootci` — Computes confidence intervals of a bootstrapped statistic. An option allows you to choose the type of the bootstrap confidence interval.
- `jackknife` — Draws jackknife samples from a data set and computes statistics on each sample

Demos

The following demos have been added to the toolbox:

- Bayesian Analysis for a Logistic Regression Model
- Time Series Regression of Airline Passenger Data

The following demo has been updated to demonstrate new features:

- Random Number Generation

Design of Experiments

The new `fracfactgen` function finds a set of fractional factorial design generators suitable for fitting a specified model.

The following functions for *D*-optimal designs have been enhanced:

- `cordexch`, `daugment`, `dcovary`, `rowexch` — New options specify the range of values and the number of levels for each factor, exclude factor combinations, treat factors as categorical rather than continuous, control the number of iterations, and repeat the design generation process from random starting points
- `candexch` — New options control the number of iterations and repeat the design generation process from random starting points
- `candgen` — New options specify the range of values and the number of levels for each factor, and treat factors as categorical rather than continuous
- `x2fx` — New option treats factors as categorical rather than continuous

Hypothesis Tests

The new `dwtest` function performs a Durbin-Watson test for autocorrelation in linear regression.

Multivariate Distributions

Two new functions have been added to compute multivariate cdfs. These supplement existing functions for pdfs and random number generators for the same distributions.

- `mvncdf` — Cumulative distribution function for the multivariate normal distribution
- `mvtcdf` — Cumulative distribution function for the multivariate *t* distribution

Random Number Generation

Copulas

New functions have been added to the toolbox that allow you to use copulas to model correlated multivariate data and generate random numbers from multivariate distributions.

- `copulacdf` — Cumulative distribution function for a copula
- `copulaparam` — Copula parameters as a function of rank correlation
- `copulapdf` — Probability density function for a copula
- `copularnd` — Random numbers from a copula
- `copulastat` — Rank correlation for a copula

Markov Chain Monte Carlo Methods

The following functions generate random numbers from nonstandard distributions using Markov Chain Monte Carlo methods:

- `mhsample` — Generate random numbers using the Metropolis-Hasting algorithm
- `slicesample` — Generate random numbers using a slice sampling algorithm

Pearson and Johnson Systems of Distributions

Support has been added for random number generation from Pearson and Johnson systems of distributions.

- `pearsrnd` — Random numbers from a distribution in the Pearson system
- `johnsrnd` — Random numbers from a distribution in the Johnson system

Robust Regression

To supplement the `robustfit` function, the following functions now have options for robust fitting:

- `nlinfit` — Nonlinear least-squares regression
- `nlparci` — Confidence intervals for parameters in nonlinear regression
- `nlpredci` — Confidence intervals for predictions in nonlinear regression

Statistical Process Control

The following control chart functions now support time-series objects:

- `xbarplot` — Xbar plot
- `schart` — Standard deviation chart
- `ewmaplot` — Exponentially weighted moving average plot

R14SP3

Version: 5.1

New Features

Demos

The following demos have been added to the toolbox:

- Curve Fitting and Distribution Fitting
- Fitting a Univariate Distribution Using Cumulative Probabilities
- Fitting an Orthogonal Regression Using Principal Components Analysis
- Modelling Tail Data with the Generalized Pareto Distribution
- Pitfalls in Fitting Nonlinear Models by Transforming to Linearity
- Weighted Nonlinear Regression

The following demo has been updated:

- Modelling Data with the Generalized Extreme Value Distribution

Descriptive Statistics

The new `partialcorr` function computes the correlation of one set of variables while controlling for a second set of variables.

The `grpstats` function now computes a wider variety of descriptive statistics for grouped data. Choices include the mean, standard error of the mean, number of elements, group name, standard deviation, variance, confidence interval for the mean, and confidence interval for new observations. The function also supports the computation of user-defined statistics.

Hypothesis Tests

Chi-Square Goodness-of-Fit Test

The new `chi2gof` function tests if a sample comes from a specified distribution, against the alternative that it does not come from that distribution, using a chi-square test statistic.

Variance Tests

Three functions have been added to test sample variances:

- `vartest` — One-sample chi-square variance test. Tests if a sample comes from a normal distribution with specified variance, against the alternative that it comes from a normal distribution with a different variance.
- `vartest2` — Two-sample *F*-test for equal variances. Tests if two independent samples come from normal distributions with the same variance, against the alternative that they come from normal distributions with different variances.
- `vartestn` — Bartlett multiple-sample test for equal variances. Tests if multiple samples come from normal distributions with the same variance, against the alternative that they come from normal distributions with different variances.

Ansari-Bradley Test

The new `ansaribradley` function tests if two independent samples come from the same distribution, against the alternative that they come from distributions that have the same median and shape but different variances.

Tests of Randomness

The new `runstest` function tests if a sequence of values comes in random order, against the alternative that the ordering is not random.

Probability Distributions

Support has been added for two new distributions:

- “Generalized Extreme Value Distribution” on page 36-3
- “Generalized Pareto Distribution” on page 36-3

Generalized Extreme Value Distribution

The Generalized Extreme Value distribution combines the Gumbel, Frechet, and Weibull distributions into a single distribution. It is used to model extreme values in data.

The following distribution functions have been added:

- `gevcdf` — Cumulative distribution function
- `gevfit` — Parameter estimation function
- `gevinv` — Inverse cumulative distribution function
- `gevlike` — Negative log-likelihood function
- `gevpdf` — Probability density function
- `gevrnd` — Random number generator
- `gevstat` — Distribution statistics

Generalized Pareto Distribution

The Generalized Pareto distribution is used to model the tails of a data distribution.

The following distribution functions have been added:

- `gpcdf` — Cumulative distribution function
- `gpfit` — Parameter estimation function
- `gpinv` — Inverse cumulative distribution function
- `gplike` — Negative log-likelihood function
- `gppdf` — Probability density function
- `gprnd` — Random number generator
- `gpstat` — Distribution statistics

Regression Analysis

- The new `coxphfit` function fits Cox's proportional hazards regression model to data.
- The new `invpred` function estimates the inverse prediction intervals for simple linear regression.
- The `polyconf` function has new options to let you specify the confidence interval computed.

Statistical Visualization

Both the `ecdf` and `ksdensity` functions now produce plots when no output arguments are specified.

R14SP2

Version: 5.0.2

New Features

Bug Fixes

Multivariate Statistics

The `cophenet` function now returns cophenetic distances as well as the cophenetic correlation coefficient.