```
function [trainedClassifier,↙
validationAccuracy] =↙
trainClassifier(trainingData)
% [trainedClassifier,↙
validationAccuracy] =↙
trainClassifier(trainingData)
% Returns a trained classifier and↙
its accuracy. This code recreates↙
the
% classification model trained in↙
Classification Learner app. Use the
% generated code to automate↙
training the same model with new↙
data, or to
% learn how to programmatically↙
train models.
%
%  Input:
%      trainingData: A table↙
```

```
containing the same predictor and↙
response
%       columns as those imported↙
into the app.
%
%
%  Output:
%       trainedClassifier: A struct↙
containing the trained classifier.↙
The
%       struct contains various↙
fields with information about the↙
trained
%       classifier.
%
%       trainedClassifier.predictFcn:↙
A function to make predictions on↙
new
%       data.
```

```
%
%       validationAccuracy: A double↙
representing the validation accuracy↙
as
%       a percentage. In the app,↙
the Models pane displays the↙
validation
%       accuracy for each model.
%
% Use the code to train the model↙
with new data. To retrain your
% classifier, call the function from↙
the command line with your original
% data or new data as the input↙
argument trainingData.
%
% For example, to retrain a↙
classifier trained with the original↙
data set
```

```
% T, enter:
%   [trainedClassifier,↙
validationAccuracy] =↙
trainClassifier(T)
%
% To make predictions with the↙
returned 'trainedClassifier' on new↙
data T2,
% use
%   [yfit,scores] =↙
trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at↙
least the same predictor columns as↙
used
% during training. For details,↙
enter:
%   trainedClassifier.HowToPredict
```

```
% Auto-generated by MATLAB on 21-↙
Apr-2024 11:11:40


% Extract predictors and response
% This code processes the data into↙
the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'REF', 'ALT',↙
'CLNREVSTAT', 'CLNSIG',↙
'CLNSIGCONF', 'Transcript',↙
'Consequence', 'pLI', 'GERP',↙
'Eigen', 'FATHMM', 'M_CAP',↙
'MetaRNN', 'MetaSVM', 'MPC',↙
'MutationAssessor',↙
'MutationTaster', 'MVP',↙
'Polyphen2_HVAR', 'PrimateAI',↙
'PROVEAN', 'SIFT4G', 'gMVP',↙
```

```
'VEST4', 'REVEL', 'CADD', 'EVE',↙
'gnomAD_AF_popmax', 'gnomAD3_AF',↙
'UKBB_AF'};
predictors = inputTable(:,↙
predictorNames);
response = inputTable.Pathogenicity;
isCategoricalPredictor = [true,↙
true, true, true, true, true, true,↙
false, false, false, false, false,↙
false, false, false, false, false,↙
false, false, false, false, false,↙
false, false, false, false, true,↙
false, false, false];
classNames = {'Benign';↙
'Benign_VUS'; 'Likely_Benign';↙
'Likely_Pathogenic'; 'Pathogenic';↙
'Pathogenic_VUS'; 'VUS_Benign';↙
'VUS_Pathogenic'};
```

```
% Train a classifier
% This code specifies all the↙
classifier options and trains the↙
classifier.
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'ClassNames', classNames);

% Create the result struct with↙
predict function
predictorExtractionFcn = @(t) t(:,↙
predictorNames);
treePredictFcn = @(x) predict↙
(classificationTree, x);
trainedClassifier.predictFcn = @(x)↙
```

```
treePredictFcn↙
(predictorExtractionFcn(x));

% Add additional fields to the↙
result struct
trainedClassifier.RequiredVariables↙
= {'ALT', 'CADD', 'CLNREVSTAT',↙
'CLNSIG', 'CLNSIGCONF',↙
'Consequence', 'EVE', 'Eigen',↙
'FATHMM', 'GERP', 'MPC', 'MVP',↙
'M_CAP', 'MetaRNN', 'MetaSVM',↙
'MutationAssessor',↙
'MutationTaster', 'PROVEAN',↙
'Polyphen2_HVAR', 'PrimateAI',↙
'REF', 'REVEL', 'SIFT4G',↙
'Transcript', 'UKBB_AF', 'VEST4',↙
'gMVP', 'gnomAD3_AF',↙
'gnomAD_AF_popmax', 'pLI'};
trainedClassifier.ClassificationTree↙
```

```
= classificationTree;
trainedClassifier.About = 'This↙
struct is a trained model exported↙
from Classification Learner↙
R2023a.';
trainedClassifier.HowToPredict =↙
sprintf('To make predictions on a↙
new table, T, use: \n  [yfit,scores]↙
= c.predictFcn(T) \nreplacing ''c''↙
with the name of the variable that↙
is this struct, e.g.↙
''trainedModel''. \n \nThe table, T,↙
must contain the variables returned↙
by: \n  c.RequiredVariables↙
\nVariable formats (e.g.↙
matrix/vector, datatype) must match↙
the original training data.↙
\nAdditional variables are ignored.↙
\n \nFor more information, see <a↙
```

```
href="matlab:helpview(fullfile↙
(docroot, ''stats'', ''stats.map''),↙
''appclassification_exportmodeltowor↙
kspace'')">How to predict using an↙
exported model</a>.');

% Extract predictors and response
% This code processes the data into↙
the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'REF', 'ALT',↙
'CLNREVSTAT', 'CLNSIG',↙
'CLNSIGCONF', 'Transcript',↙
'Consequence', 'pLI', 'GERP',↙
'Eigen', 'FATHMM', 'M_CAP',↙
'MetaRNN', 'MetaSVM', 'MPC',↙
'MutationAssessor',↙
'MutationTaster', 'MVP',↙
```

```
'Polyphen2_HVAR', 'PrimateAI',↙
'PROVEAN', 'SIFT4G', 'gMVP',↙
'VEST4', 'REVEL', 'CADD', 'EVE',↙
'gnomAD_AF_popmax', 'gnomAD3_AF',↙
'UKBB_AF'};
predictors = inputTable(:,↙
predictorNames);
response = inputTable.Pathogenicity;
isCategoricalPredictor = [true,↙
true, true, true, true, true, true,↙
false, false, false, false, false,↙
false, false, false, false, false,↙
false, false, false, false, false,↙
false, false, false, false, true,↙
false, false, false];
classNames = {'Benign';↙
'Benign_VUS'; 'Likely_Benign';↙
'Likely_Pathogenic'; 'Pathogenic';↙
'Pathogenic_VUS'; 'VUS_Benign';↙
```

```
'VUS_Pathogenic'};

% Set up holdout validation
cvp = cvpartition(response,↙
'Holdout', 0.17);
trainingPredictors = predictors(cvp.↙
training, :);
trainingResponse = response(cvp.↙
training, :);
trainingIsCategoricalPredictor =↙
isCategoricalPredictor;

% Train a classifier
% This code specifies all the↙
classifier options and trains the↙
classifier.
classificationTree = fitctree(...
    trainingPredictors, ...
    trainingResponse, ...
```

```
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 100, ...
    'Surrogate', 'off', ...
    'ClassNames', classNames);

% Create the result struct with↙
predict function
treePredictFcn = @(x) predict↙
(classificationTree, x);
validationPredictFcn = @(x)↙
treePredictFcn(x);

% Add additional fields to the↙
result struct


% Compute validation predictions
validationPredictors = predictors↙
(cvp.test, :);
```

```
validationResponse = response(cvp.↙
test, :);
[validationPredictions,↙
validationScores] =↙
validationPredictFcn↙
(validationPredictors);

% Compute validation accuracy
correctPredictions = strcmp( strtrim↙
(validationPredictions), strtrim↙
(validationResponse));
isMissing = cellfun(@(x) all(isspace↙
(x)), validationResponse,↙
'UniformOutput', true);
correctPredictions =↙
correctPredictions(~isMissing);
validationAccuracy = sum↙
(correctPredictions)/length↙
(correctPredictions);
```