```
function [trainedClassifier, validationAccuracy] =
 kNearNeighbors(trainingData)
% trainClassifier(trainingData)
%  returns a trained classifier and its validation accuracy.
%  This code recreates the classification model trained in
%  Classification Learner app.
%
%   Input:
%       trainingData: the training data of same data type as imported
%        in the app (table or matrix).
%
%   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. It takes an input of the same form as this
 training
%        code (table or matrix) and returns predictions for the
 response.
%        If you supply a matrix, include only the predictors columns
 (or
%        rows).
%
%       validationAccuracy: a double containing the validation
 accuracy
%        score in percent. In the app, the History list displays this
%        overall accuracy score 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 T,
%  use
%    yfit = trainedClassifier.predictFcn(T)
%
%  To automate training the same classifier with new data, or to learn
 how
%  to programmatically train classifiers, examine the generated code.

% Auto-generated by MATLAB on 14-Feb-2016 20:38:18
```

```
% Convert input to table
inputTable = table(trainingData);
inputTable.Properties.VariableNames = {'column'};

% Split matrices in the input table into vectors
inputTable =
 [inputTable(:,setdiff(inputTable.Properties.VariableNames,
 {'column'})), array2table(table2array(inputTable(:,
{'column'})), 'VariableNames',
 {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_

% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
predictorNames =
 {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_
predictors = inputTable(:, predictorNames);
response = inputTable.column_37;

% Train a classifier
% This code specifies all the classifier options and trains the
 classifier.
classificationKNN = fitcknn(...
    predictors, ...
    response, ...
    'Distance', 'Cityblock', ...
    'Exponent', [], ...
    'NumNeighbors', 4, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', [1; 2; 3; 4; 5; 6]);

trainedClassifier.ClassificationKNN = classificationKNN;
convertMatrixToTableFcn = @(x) table(x, 'VariableNames', {'column'});
splitMatricesInTableFcn = @(t)
 [t(:,setdiff(t.Properties.VariableNames, {'column'})),
 array2table(table2array(t(:,{'column'})), 'VariableNames',
 {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_
extractPredictorsFromTableFcn = @(t) t(:, predictorNames);
predictorExtractionFcn = @(x)
 extractPredictorsFromTableFcn(splitMatricesInTableFcn(convertMatrixToTableFcn(x))
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x)
 knnPredictFcn(predictorExtractionFcn(x));
% Convert input to table
inputTable = table(trainingData);
inputTable.Properties.VariableNames = {'column'};

% Split matrices in the input table into vectors
inputTable =
 [inputTable(:,setdiff(inputTable.Properties.VariableNames,
 {'column'})), array2table(table2array(inputTable(:,
{'column'})), 'VariableNames',
 {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_
```

```
% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
predictorNames =
 {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_
predictors = inputTable(:, predictorNames);
response = inputTable.column_37;


% Set up holdout validation
cvp = cvpartition(response, 'Holdout', 0.31);
trainingPredictors = predictors(cvp.training,:);
trainingResponse = response(cvp.training,:);

% Train a classifier
% This code specifies all the classifier options and trains the
 classifier.
classificationKNN = fitcknn(...
    trainingPredictors, ...
    trainingResponse, ...
    'Distance', 'Cityblock', ...
    'Exponent', [], ...
    'NumNeighbors', 4, ...
    'DistanceWeight', 'Equal', ...
    'Standardize', true, ...
    'ClassNames', [1; 2; 3; 4; 5; 6]);

knnPredictFcn = @(x) predict(classificationKNN, x);
validationPredictFcn = @(x) knnPredictFcn(x);

% Compute validation accuracy
validationPredictors = predictors(cvp.test,:);
validationResponse = response(cvp.test,:);

[validationPredictions, validationScores] =
 validationPredictFcn(validationPredictors);
correctPredictions = (validationPredictions == validationResponse);
validationAccuracy = sum(correctPredictions)/
length(correctPredictions);
```

```
Not enough input arguments.

Error in kNearNeighbors (line 45)
inputTable = table(trainingData);
```

Published with MATLAB® R2015b