**#Decision Tree**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Returns a trained regression model and its RMSE. This code recreates the

% model trained in Regression 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:

% trainedModel: A struct containing the trained regression model. The

% struct contains various fields with information about the trained

% model.

% trainedModel.predictFcn: A function to make predictions on new data.

% validationRMSE: A double representing the validation RMSE. In the

% app, the Models pane displays the validation RMSE for each model

% This code processes the data into the right shape for training the

% model.

inputTable = trainingData;

predictorNames = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Train a regression model

% This code specifies all the model options and trains the model.

regressionTree = fitrtree(...

predictors, ...

response, ...

'MinLeafSize', 4, ...

'MaximalDepth', 7,...

'Surrogate', 'off');

% Create the result struct with predict function

predictorExtractionFcn = @(t) t(:, predictorNames);

treePredictFcn = @(x) predict(regressionTree, x);

trainedModel.predictFcn = @(x) treePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct

trainedModel.RequiredVariables = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6','Var7', 'Var8'};

trainedModel.RegressionTree = regressionTree;

trainedModel.HowToPredict = sprintf('To make predictions on a new table, T, use: \n yfit = c.predictFcn(T) \nreplace ''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''), ''appregression\_exportmodeltoworkspace'')">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 = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Perform cross-validation

partitionedModel = crossval(trainedModel.RegressionTree, 'KFold', 5);

% Compute validation predictions

validationPredictions = kfoldPredict(partitionedModel);

% Compute validation RMSE

validationRMSE = sqrt(kfoldLoss(partitionedModel, 'LossFun', 'mse'));

**#Bagged Trees**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Returns a trained regression model and its RMSE. This code recreates the

% model trained in Regression 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:

% trainedModel: A struct containing the trained regression model. The

% struct contains various fields with information about the trained

% model.

% trainedModel.predictFcn: A function to make predictions on new data.

% validationRMSE: A double representing the validation RMSE. In the

% app, the Models pane displays the validation RMSE for each model.

% This code processes the data into the right shape for training the

% model.

inputTable = trainingData;

predictorNames = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Train a regression model

% This code specifies all the model options and trains the model.

template = templateTree(...

'MinLeafSize', 8, ...

'NumVariablesToSample', 'all');

regressionEnsemble = fitrensemble(...

predictors, ...

response, ...

'Method', 'Bag', ...

'NumLearningCycles', 30, ...

'Learners', template);

% Create the result struct with predict function

predictorExtractionFcn = @(t) t(:, predictorNames);

ensemblePredictFcn = @(x) predict(regressionEnsemble, x);

trainedModel.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct

trainedModel.RequiredVariables = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7', 'Var8'};

trainedModel.RegressionEnsemble = regressionEnsemble;

trainedModel.HowToPredict = sprintf('To make predictions on a new table, T, use: \n yfit = c.predictFcn(T) \nreplace ''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''), ''appregression\_exportmodeltoworkspace'')">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 = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Perform cross-validation

partitionedModel = crossval(trainedModel.RegressionEnsemble, 'KFold', 5);

% Compute validation predictions

validationPredictions = kfoldPredict(partitionedModel);

% Compute validation RMSE

validationRMSE = sqrt(kfoldLoss(partitionedModel, 'LossFun', 'mse'));

**#Boosted Trees**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Returns a trained regression model and its RMSE. This code recreates the

% model trained in Regression 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:

% trainedModel: A struct containing the trained regression model. The

% struct contains various fields with information about the trained

% model.

% trainedModel.predictFcn: A function to make predictions on new data.

% validationRMSE: A double representing the validation RMSE. In the

% app, the Models pane displays the validation RMSE for each model.

%

% This code processes the data into the right shape for training the

% model.

inputTable = trainingData;

predictorNames = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Train a regression model

% This code specifies all the model options and trains the model.

template = templateTree(...

'MinLeafSize', 8, ...

'NumVariablesToSample', 'all');

regressionEnsemble = fitrensemble(...

predictors, ...

response, ...

'Method', 'LSBoost', ...

'NumLearningCycles', 30, ...

'Learners', template, ...

'LearnRate', 0.1);

% Create the result struct with predict function

predictorExtractionFcn = @(t) t(:, predictorNames);

ensemblePredictFcn = @(x) predict(regressionEnsemble, x);

trainedModel.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct

trainedModel.RequiredVariables = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7', 'Var8'};

trainedModel.RegressionEnsemble = regressionEnsemble;

trainedModel.HowToPredict = sprintf('To make predictions on a new table, T, use: \n yfit = c.predictFcn(T) \nreplace ''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''), ''appregression\_exportmodeltoworkspace'')">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 = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Perform cross-validation

partitionedModel = crossval(trainedModel.RegressionEnsemble, 'KFold', 5);

% Compute validation predictions

validationPredictions = kfoldPredict(partitionedModel);

% Compute validation RMSE

validationRMSE = sqrt(kfoldLoss(partitionedModel, 'LossFun', 'mse'));

**#Single-layered ANN**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Returns a trained regression model and its RMSE. This code recreates the

% model trained in Regression 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:

% trainedModel: A struct containing the trained regression model. The

% struct contains various fields with information about the trained

% model.

% trainedModel.predictFcn: A function to make predictions on new data.

% validationRMSE: A double representing the validation RMSE. In the

% app, the Models pane displays the validation RMSE for each model.

% This code processes the data into the right shape for training the

% model.

inputTable = trainingData;

predictorNames = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6','Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Train a regression model

% This code specifies all the model options and trains the model.

regressionNeuralNetwork = fitrnet(...

predictors, ...

response, ...

'LayerSizes', 8, ...

'Activations', 'relu', ...

'Lambda', 0, ...

'IterationLimit', 1000, ...

'Standardize', true);

% Create the result struct with predict function

predictorExtractionFcn = @(t) t(:, predictorNames);

neuralNetworkPredictFcn = @(x) predict(regressionNeuralNetwork, x);

trainedModel.predictFcn = @(x) neuralNetworkPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct

trainedModel.RequiredVariables = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6','Var7', 'Var8'};

trainedModel.RegressionNeuralNetwork = regressionNeuralNetwork;

trainedModel.HowToPredict = sprintf('To make predictions on a new table, T, use: \n yfit = c.predictFcn(T) \nreplace ''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''), ''appregression\_exportmodeltoworkspace'')">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 = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6','Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Perform cross-validation

partitionedModel = crossval(trainedModel.RegressionNeuralNetwork, 'KFold', 5);

% Compute validation predictions

validationPredictions = kfoldPredict(partitionedModel);

% Compute validation RMSE

validationRMSE = sqrt(kfoldLoss(partitionedModel, 'LossFun', 'mse'));

**#Bi-layered ANN**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Returns a trained regression model and its RMSE. This code recreates the

% model trained in Regression 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:

% trainedModel: A struct containing the trained regression model. The

% struct contains various fields with information about the trained

% model.

% trainedModel.predictFcn: A function to make predictions on new data.

% validationRMSE: A double representing the validation RMSE. In the

% app, the Models pane displays the validation RMSE for each model.

% This code processes the data into the right shape for training the

% model.

inputTable = trainingData;

predictorNames = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7' };

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Train a regression model

% This code specifies all the model options and trains the model.

regressionNeuralNetwork = fitrnet(...

predictors, ...

response, ...

'LayerSizes', [8 12], ...

'Activations', 'relu', ...

'Lambda', 0, ...

'IterationLimit', 1000, ...

'Standardize', true);

% Create the result struct with predict function

predictorExtractionFcn = @(t) t(:, predictorNames);

neuralNetworkPredictFcn = @(x) predict(regressionNeuralNetwork, x);

trainedModel.predictFcn = @(x) neuralNetworkPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct

trainedModel.RequiredVariables = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7', 'Var8'};

trainedModel.RegressionNeuralNetwork = regressionNeuralNetwork;

trainedModel.HowToPredict = sprintf('To make predictions on a new table, T, use: \n yfit = c.predictFcn(T) \nreplace ''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''), ''appregression\_exportmodeltoworkspace'')">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 = {'Var1', 'Var2', 'Var3', 'Var4', 'Var5', 'Var6', 'Var7'};

predictors = inputTable(:, predictorNames);

response = inputTable.Var8;

isCategoricalPredictor = [false, false, false, false, false, false, false];

% Perform cross-validation

partitionedModel = crossval(trainedModel.RegressionNeuralNetwork, 'KFold', 5);

% Compute validation predictions

validationPredictions = kfoldPredict(partitionedModel);

% Compute validation RMSE

validationRMSE = sqrt(kfoldLoss(partitionedModel, 'LossFun', 'mse'));