**Price Prediction Models in MATLAB**

**#Decision Tree**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Use the code to train the model with new data. To retrain your model,

% call the function from the command line with your original data or new

% data as the input argument trainingData.

% 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];

% Train a regression 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'};

trainedModel.RegressionTree = regressionTree;

trainedModel.About = 'This struct is a trained model exported from Regression Learner R2024b.';

trainedModel.HowToPredict = sprintf();

% 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'));

**#Random Forest**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Use the code to train the model with new data. To retrain your model,

% call the function from the command line with your original data or new

% data as the input argument trainingData.

% 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];

% Train a regression 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'};

trainedModel.RegressionEnsemble = regressionEnsemble;

trainedModel.About = 'This struct is a trained model exported from Regression Learner R2024b.';

trainedModel.HowToPredict = sprintf();

% 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'));

**#GBT**

function [trainedModel, validationRMSE] = trainRegressionModel(trainingData)

% Use the code to train the model with new data. To retrain your model,

% call the function from the command line with your original data or new

% data as the input argument trainingData.

% 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];

% 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', 0.7);

% 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'};

trainedModel.RegressionEnsemble = regressionEnsemble;

trainedModel.About = 'This struct is a trained model exported from Regression Learner R2024b.';

trainedModel.HowToPredict = sprintf();

% 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'));

**#ANN**

% This script assumes these variables are defined:

%

% Egt\_inp - input data.

% Egt\_output - target data.

x = Egt\_inp';

t = Egt\_output';

d = Test\_input';

o = Test\_output';

% Choose a Training Function

% 'trainlm'

% 'trainbr'

% 'trainscg'

trainFcn = 'trainscg';

% Create a Fitting Network

hiddenLayerSize = [6 4];

net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training, Validation

net.divideParam.trainRatio = 80/100;

net.divideParam.valRatio = 20/100;

% Train the Network

[net,tr] = train(net,x,t);

% Test the Network

y = net(x);

e = gsubtract(t,y);

performance = perform(net,t,y)

performance2=mae(e)

a = net(d);

h = gsubtract(o,a);

performance3 = perform(net,0,a)

performance2=mae(h)

% View the Network

view(net)

% Plots

figure, plotperform(tr)

figure, plottrainstate(tr)

figure, ploterrhist(e)

figure, plotregression(t,y)

figure, plotfit(net,x,t)

**Clustering in R Studio**

#selected libraries

library(dplyr)

library(tidyverse)

library(ggplot2)

library(factoextra)

library(class)

library(cluster)

library(clValid)

# Price consumption graph

ggplot(data=dataset,aes(x=Fiyat,y=Tuketim)) + geom\_point()

#Turnover and Consumption Graph

ggplot(data=dataset,aes(x=Ciro/1000,y=Tuketim)) + geom\_point()

#Noisy data cleaning (Negative ve -500 €/Ton prices cleaning).

new\_dataset <- filter(dataset, Fiyat > 500)

#only serial references preferred

new\_dataset2 <- filter(new\_dataset, Durum == "EVET")

#unknown projects has cleared

new\_dataset3 <- na.omit(new\_dataset2)

#Turnover Analysis

ciroanaliz <- select(new\_dataset3, Projeno, Ciro)

ciroanaliz1 <- group\_by(ciroanaliz, Projeno)

ciroanaliz2 <- summarise(ciroanaliz1, sum(Ciro))

arrange(ciroanaliz2, desc(`sum(Ciro)`))

boxplot(ciroanaliz2$Projeno)

boxplot(ciroanaliz2$`sum(Ciro)`)

hist(ciroanaliz2$`sum(Ciro)`/1000)

hist(dataset$Fiyat,xlab = "Fiyat",ylab = "Sıklık")

hist(new\_dataset3$Fiyat,xlab = "Fiyat",ylab = "Sıklık")

#Anderson Darling Application for the distribution

library(nortest)

ad.test(new\_dataset3$Fiyat)

ad.test(dataset$Fiyat)

ggplot(data=new\_dataset3,aes(x=Fiyat,y=Tüketim)) + geom\_point()

ggplot(data=new\_dataset3,aes(x=Ciro/1000,y=Tüketim)) + geom\_point()

#library(openxlsx)

#write.xlsx(new\_dataset3, file="newdataset3.xlsx")

#write.xlsx(scaledata, file="scaledata.xlsx")

#Clustering steps

library(factoextra)

mydata.labes <- new\_dataset3$Projeno

clustering1=select(new\_dataset3,c(3,9,10,11))

clustering2=select(clustering1,c(1,4))

scaledata3 <- scale(clustering2)

#Distance Calculations

clustering2<- dist(scaledata3,method = "euclidian")

#Ideal Cluster Numbers with K-Means

fviz\_nbclust(scaledata3, kmeans, method = "wss") + labs(subtitle="Elbow Method")

#fviz\_nbclust(scaledata, kmeans, method = "silhouette")+labs(subtitle="Silhouette

Method")

fviz\_nbclust(scaledata3, kmeans, method = "silhouette")+labs(subtitle="Silhouette

Method")

fviz\_nbclust(scaledata3, kmeans, method = "gap\_stat")+ labs(subtitle="Gap Statistics

Method")

km.out <- kmeans(scaledata3, centers = 4, nstart=300)

print(km.out)

#Cluster visualization

km.clusters <-km.out$cluster

rownames(scaledata3) <- paste(new\_dataset3$Projeno,1:dim(scaledata3)[1],sep = "\_")

fviz\_cluster(list(data=scaledata3, cluster=km.clusters))

table(km.clusters,new\_dataset3$Projeno)

table(km.clusters,new\_dataset3$Durum)

clmethods <- c("hierarchical","kmeans","pam")

intern <- clValid(scaledata3, nClust = 3:7,clMethods = clmethods,

validation = "internal")

summary(intern)