ST3189 Coursework codes

200664383

2023-04-02

https://www.openml.org/data/download/1586225/php0iVrYT

https://www.openml.org/data/download/21377442/file16a868cf35f5.arff

#Regression————————————————————————————–

#CART on runtime using predictor variables

#install and load ‘rpart’ package which provides the functions for building decision trees (for CART) install.packages(“rpart”) install.packages(“lattice”) install.packages(“caret”) library(ggplot2) library(rpart) library(caret) library(readr) library(lattice)

#set working directory to import dataset setwd(“C:/Users/User/Documents/ST3189 Machine Learning/ST3189 coursework”)

#load dataset CPMP <- read.csv(“CPMP.csv”)

#remove duplicates CPMP_nd <- unique(CPMP)

Set random seed to make it reproducible

set.seed(123)

Split data into train set (70%) and test set (30%)

train_indices <- sample(nrow(CPMP_nd), round(0.7 * nrow(CPMP_nd))) CPMP_train_data <- CPMP_nd[train_indices,] CPMP_test_data <- CPMP_nd[-train_indices,]

#check class of train set class(CPMP_train_data$runtime)

#model the training set (CART) model_CPMP <- rpart(runtime ~ stacks + tiers + overstowing.stack.pct + empty.stack.pct + left.density , data = CPMP_train_data, method = “anova”)

#print the summary of the model summary(model_CPMP)

#plot the tree par(mar = c(5, 5, 5, 5)) #set size of plot area windows(width=10, height=12) plot(model_CPMP) text(model_CPMP, use.n=TRUE, all=TRUE, cex=0.8)

#use test set to make predictions pred_CPMP <- predict(model_CPMP, newdata = CPMP_test_data)

Evaluate model using MSE, R-squared, and MAE

mse <- mean((CPMP_test_data$runtime - pred_CPMP)^2)r_squared <- cor(CPMP_test_data$runtime, pred_CPMP)^2 mae <- mean(abs(CPMP_test_data$runtime - pred_CPMP))

Print evaluation metrics

cat("Mean Squared Error (MSE):", mse, "") cat("R-squared:", r_squared, "") cat("Mean Absolute Error (MAE):", mae, "")

#since the r squared value is near 0 and the mse and mae is very high , we will run PCA to reduce the complexity of the dataset, before modelling it again with CART

#PCA———————————————————————————————— #load required libraries library(caret) library(randomForest) library(dplyr) library(tidyr)

Select the predictor variables

predictors <- c("stacks", "tiers", "overstowing.stack.pct", "empty.stack.pct", "left.density")

Scale and perform PCA on the training data

CPMP_scaled_train <- scale(CPMP_train_data[, predictors])

pca <- prcomp(CPMP_scaled_train[, predictors]) cpmp_train_pca <- predict(pca, CPMP_scaled_train)

plot(pca)

#choose principal components to train model cpmp_train_model <- data.frame(PC1 = cpmp_train_pca[,1], PC2 = cpmp_train_pca[,2], PC3 = cpmp_train_pca[,3], runtime = CPMP_train_data$runtime)

Remove all rows with missing values from dataset

cpmp_train_model <- na.omit(cpmp_train_model)

Train a CART model on the PCA-transformed data

cpmp_cart_model <- train(runtime ~ ., data = cpmp_train_model, method = “rpart”, trControl = trainControl(method = “cv”, number = 10))

Print a summary of model

print(cpmp_cart_model)

#predict runtime with train set cpmp_train_predicted <- predict(cpmp_cart_model, newdata = cpmp_train_model)

Scale testing data using the same standard deviation and as the train data

CPMP_scaled_test <- scale(CPMP_test_data[, predictors], center = attr(CPMP_scaled_train, “scaled:center”), scale = attr(CPMP_scaled_train, “scaled:scale”))

Project test data onto the principal components obtained from training data

cpmp_test_pca <- predict(pca, CPMP_scaled_test)

View the first few rows of the test data after PCA

head(cpmp_test_pca)

cpmp_test_model <- data.frame(PC1 = cpmp_test_pca[,1], PC2 = cpmp_test_pca[,2], PC3 = cpmp_test_pca[,3], runtime = CPMP_test_data$runtime)

#Predict runtime with scaled test set cpmp_test_predicted <- predict(cpmp_cart_model, newdata = cpmp_test_model)

Print the summary of model

print(cpmp_test_predicted)

Calculate RMSE for train and test predictions

train_rmse <- RMSE(cpmp_train_predicted, CPMP_train_data$runtime)test_r mse < -RMSE(cpmp_t est_p redicted, CPMP_t est_d ata$runtime)

Print RMSE values

cat(paste("Train RMSE:", train_rmse, "")) cat(paste("Test RMSE:", test_rmse, ""))

install.packages("rpart.plot") library(rpart.plot)

Plot actual vs predicted runtime for train data

plot(CPMP_train_data$runtime, cpmp_train_predicted, main = "Actual vs Predicted Runtime (Train Data)", xlab = "Actual Runtime", ylab = "Predicted Runtime")

Plot actual vs predicted runtime for test data

plot(CPMP_test_data$runtime, cpmp_test_predicted, main = "Actual vs Predicted Runtime (Test Data)", xlab = "Actual Runtime", ylab = "Predicted Runtime")

#Evaluate the performance of the CART model using mean squared error (MSE) and R-squared mse <- mean((cpmp_test_predicted - CPMP_test_data$runtime)^2)rsq < -cor(cpmp_test_predicted, CPMP_test_data$runtime)^2

#pruning the decision tree

#showing which variable has the strongest relevance to determining runtime summary(pca) print(pca$rotation[,1])

#Since CART modelling still produced a low R-squared value and high MSE value, we will try randomForest to model the dataset

RandomForest——————————————————————————

#Fit a random forest model using randomForest() function rf_model <- randomForest(runtime ~ ., data = CPMP_train_data, ntree = 500, importance = TRUE)

#Use the predict() function to make predictions on the test set using fitted model prediction_CPMP <- predict(rf_model, newdata = CPMP_test_data)

#Evaluate performance of the model using mean squared error (MSE) and R-squared

mse2 <- mean((predictions_CPMP - CPMP_test_data$runtime)^2)

$$rsq2 < -cor(\text{predictions\_CPMP}, \text{CPMP\_test\_data}runtime)^2$$

#Plotting the out-of-bag error rate versus the number of trees in forest using the plot() function plot(rf_model)

#Visualize the predicted versus observed values of the target variable using a scatter plot prediction_CPMP <- predict(rf_model, CPMP_test_data) windows(width=10, height=8)

adjusting width and height of plot (not sure if want to keep this) plot(predictions_CPMP, CPMP_test_data$runtime, main = “Predicted vs Observed Values”) abline(0, 1, col = “red”)

#randomForets on overstowage.pct

rf_model2 <- randomForest(overstowage.pct ~ ., data = CPMP_train_data, ntree = 500, importance = TRUE)

#Use the predict() function to make predictions on the test set using the fitted model

prediction_CPMP2 <- predict(rf_model2, newdata = CPMP_test_data)

#Evaluate the performance of the model using mean squared error (MSE) and R-squared mse2 <- mean((prediction_CPMP2 - CPMP_test_data$overstowage.pct)^2$)

$$rsq2 < -cor(\text{predictions\_CPMP2 , CPMP\_test\_data}overstowage.pct)^2$$

#Plot the out-of-bag error rate versus the number of trees in the forest using the plot() function plot(rf_model2) ————————————————————————————————————

#linear regression on overstowage

Define the predictor variables

predictors <- c(“stacks”, “tiers”, “left.density”)

Fit a linear regression model to predict overstowing.stack.pct

model <- lm(overstowage.pct ~ stacks + tiers + left.density, data = CPMP_train_data)

#summarize the model summary(model)

Predict overstowage.pct values using the model and predictor variables

prediction <- predict(model, newdata = CPMP_test_data[, predictors])

#visualise results library(ggplot2) library(dplyr) ggplot(CPMP_test_data, aes(x = prediction, y = overstowage.pct)) + geom_point() + geom_smooth(method = “lm”, se = FALSE) + labs(title = “Linear Regression Prediction vs Actual Overstowage Percentage”, x = “Predicted Overstowage Percentage”, y = “Actual Overstowage Percentage”)

#Classification——————————————————————————

#install and load necessary libraries install.packages(“randomForest”) install.packages(“ggplot2”) install.packages(“lattice”) library(rpart) library(randomForest) library(ggplot2) library(caret) library(readr) library(lattice)

#set working directory to import dataset setwd(“C:/Users/User/Documents/ST3189 Machine Learning/ST3189 coursework”)

#load data bd <- read.csv(“blood_donation.csv.csv”)

#remove duplicates bd_nd <- unique(bd)

Set random seed to make it reproducible

set.seed(123)

Split data into train set (70%) and test set (30%)

train_indices <- sample(nrow(bd_nd), round(0.7 * nrow(bd_nd))) bd_train_data <- bd_nd[train_indices,] bd_test_data <- bd_nd[-train_indices,]

#check class of bd train set class(bd_train_data$Class)

#model the training set (CART) model <- rpart(Class ~ recency + frequency, data = bd_train_data)

#print the summary of the model summary(model)

#plot the tree par(mar = c(5, 5, 5, 5)) #set size of plot area windows(width=10, height=8) plot(model) text(model, use.n=TRUE, all=TRUE, cex=0.8)

#use test set to evaluate performance

pred <- predict(model, newdata = bd_test_data)

#compare predicted value to actual value table(pred, bd_test_data$Class)

accuracy <- sum(pred == bd_test_data$Class) / length(bd_test_data$Class)

precision <- sum(pred & bd_test_data$Class) / sum(pred)

recall <- sum(pred & bd_test_data$Class)/sum(bd_test_data$Class)

f1_score <- 2 * precision * recall / (precision + recall)

#Load the necessary libraries

library(dplyr) library(tidyr) library(ggplot2) library(caret)

Check if the target variable is binary

if (length(unique(bd_nd$Class)) > 2) { stop("Target variable is not binary") }

Replace 1 with 0 and 2 with 1 in the Class column

bd$Class < -ifelse(bd$Class == 1, 0, 1) bd$Class < -ifelse(bd$Class == 2, 1, bd$Class)

Save the modified dataset

write.csv(bd, "bd.csv", row.names = FALSE)

Set random seed to make it reproducible

set.seed(123)

Split data into train set (70%) and test set (30%)

train_indices <- sample(nrow(bd), round(0.7 * nrow(bd))) bd_train_data <- bd[train_indices,] bd_test_data <- bd[-train_indices,]

Perform logistic regression

logistic_model <- glm(Class ~ recency + frequency, data = bd_train_data, family = "binomial")

Making predictions on the test data

pred <- predict(logistic_model, newdata = bd_test_data, type = "response") predicted_class <- ifelse(pred >0.5, 1, 0)

predicted_class <- factor(predicted_class) bd_test_data$Class < -factor(bd_t est_d ata$Class)

Check the levels of both variables

levels(predicted_class) levels(bd_test_data$Class)

Re-level the predicted variable to match the reference variable

predicted_class <- factor(predicted_class, levels = levels(bd_test_data$Class))

Check the levels again to make sure they match

levels(predicted_class) levels(bd_test_data$Class)

#Evaluate the model confusion_matrix <- confusionMatrix(data = predicted_class, reference = bd_test_data$Class)

#visualizing the confusion matrix mosaicplot(confusion_matrix$table, main = "Confusion Matrix")

randomForest————————————————————————

library(randomForest)

Set seed to make it reproducible

set.seed(123)

bd <- read.csv("bd.csv") bd_nd <- na.omit(bd_nd)

#Split the data into training set (70%) and testing set (30%)

train_row <- sample(nrow(bd), round(0.7 * nrow(bd)))

bd_train_data <- bd_nd[train_row,]

bd_test_data <- bd_nd[-train_row,]

Train randomForest model

bd_model_rf <- randomForest(Class ~ ., data = bd_train_data, ntree = 80, mtry = sqrt(ncol(bd_train_data)))

Make predictions on test data

pred_bd <- predict(bd_model_rf, bd_test_data)

Evaluate accuracy of predictions

accuracy <- mean(pred_bd == bd_test_data$Class)

Re-level the predicted variable to match the reference variable

pred_bd <- factor(pred_bd, levels = levels(bd_test_data$Class))

response has 5 or fewer unique values, indicating that randomForest is not suitable to model this dataset——