**TW-1**

# Load necessary libraries

install.packages(c("mlbench", "dplyr", "ggplot2", "reshape2", "caret"))

library(mlbench)

library(dplyr)

library(ggplot2)

library(reshape2)

library(caret)

# Load and inspect the data

data("BostonHousing")

housing <- BostonHousing

str(housing)

# Density Plot of Median Value House Price

ggplot(housing, aes(x = medv)) +

  stat\_density() +

  labs(x = "Median Value ($1000s)", y = "Density", title = "Density Plot of Median Value House Price in Boston") +

  theme\_minimal()

# Predicted vs. Original Values Visualization

housing %>%

  select(crim, rm, age, rad, tax, lstat, medv) %>%

  melt(id.vars = "medv") %>%

  ggplot(aes(x = value, y = medv, colour = variable)) +

  geom\_point(alpha = 0.7) +

  stat\_smooth(colour = "black") +

  facet\_wrap(~variable, scales = "free", ncol = 2) +

  labs(x = "Variable Value", y = "Median House Price ($1000s)") +

  theme\_minimal()

# Train-Test Split

set.seed(123)

trainIndex <- createDataPartition(housing$medv, p = 0.75, list = FALSE)

train <- housing[trainIndex, ]

test <- housing[-trainIndex, ]

# Fit a Linear Model

first\_lm <- lm(medv ~ crim + rm + tax + lstat, data = train)

cat("First linear model has an r-squared value of", round(summary(first\_lm)$r.squared, 3), "\n")

# Improve the Model with Log Transformation

second\_lm <- lm(log(medv) ~ crim + rm + tax + lstat, data = train)

cat("Second linear model has an r-squared value of", round(summary(second\_lm)$r.squared, 3), "\n")

# Check the mean of residuals

cat("Mean of residuals:", abs(mean(second\_lm$residuals)), "\n")

# Predicted vs. Original Values Plot

predicted <- predict(second\_lm, newdata = test)

results <- data.frame(predicted = exp(predicted), original = test$medv)

ggplot(results, aes(x = predicted, y = original)) +

  geom\_point() +

  stat\_smooth() +

  labs(x = "Predicted Values", y = "Original Values", title = "Predicted vs. Original Values") +

  theme\_minimal()

**TW-2**

# Install and load necessary packages

install.packages("KernelKnn")

library(KernelKnn)

# Load data and preprocess

data(ionosphere, package = 'KernelKnn')

ionosphere = ionosphere[, -2]

X = scale(ionosphere[, -ncol(ionosphere)])

y = as.numeric(factor(ionosphere[, ncol(ionosphere)]))

# Split the data into training and testing sets

set.seed(123)

spl\_train = sample(seq\_along(y), round(length(y) \* 0.75))

spl\_test = setdiff(seq\_along(y), spl\_train)

# Evaluation metric

acc = function(y\_true, preds) {

  out = table(y\_true, max.col(preds, ties.method = "random"))

  sum(diag(out)) / sum(out)

}

# KNN with Euclidean distance

preds\_TEST = KernelKnn(X[spl\_train, ], TEST\_data = X[spl\_test, ], y[spl\_train], k = 5,

                       method = 'euclidean', regression = FALSE, Levels = unique(y))

head(preds\_TEST)

# KNN with Canberra distance and Tricube kernel

preds\_TEST\_tric = KernelKnn(X[spl\_train, ], TEST\_data = X[spl\_test, ], y[spl\_train], k = 10,

                            method = 'canberra', weights\_function = 'tricube', regression = FALSE,

                            Levels = unique(y))

head(preds\_TEST\_tric)

# Custom normal distribution kernel function

norm\_kernel = function(W) {

  W = dnorm(W, mean = 0, sd = 1.0)

  W / rowSums(W)

}

# KNN with Canberra distance and custom kernel

preds\_TEST\_norm = KernelKnn(X[spl\_train, ], TEST\_data = X[spl\_test, ], y[spl\_train], k = 10,

                            method = 'canberra', weights\_function = norm\_kernel, regression = FALSE,

                            Levels = unique(y))

head(preds\_TEST\_norm)

# Cross-validation with Tricube kernel

fit\_cv\_pair1 = KernelKnnCV(X, y, k = 10, folds = 5, method = 'canberra',

                           weights\_function = 'tricube', regression = FALSE,

                           Levels = unique(y), threads = 5)

str(fit\_cv\_pair1)

# Cross-validation with Epanechnikov kernel

fit\_cv\_pair2 = KernelKnnCV(X, y, k = 9, folds = 5, method = 'canberra',

                           weights\_function = 'epanechnikov', regression = FALSE,

                           Levels = unique(y), threads = 5)

str(fit\_cv\_pair2)

# Calculate and print accuracy for both parameter sets

calc\_acc = function(fit\_cv) {

  unlist(lapply(1:length(fit\_cv$preds), function(x) acc(y[fit\_cv$folds[[x]]], fit\_cv$preds[[x]])))

}

acc\_pair1 = calc\_acc(fit\_cv\_pair1)

cat('Accuracy for params\_pair1 is:', mean(acc\_pair1), '\n')

acc\_pair2 = calc\_acc(fit\_cv\_pair2)

cat('Accuracy for params\_pair2 is:', mean(acc\_pair2), '\n')

**TW-3**

# Load necessary libraries

install.packages("tm")

install.packages("wordcloud")

install.packages("e1071")

library(tm)

library(wordcloud)

library(e1071)

# Load data

sms\_spam\_df <- read.csv("H:/6th/DSA/LAB/sms\_spam.csv.csv", stringsAsFactors = FALSE)

str(sms\_spam\_df)

# Create and clean corpus

sms\_corpus <- VCorpus(VectorSource(sms\_spam\_df$text))

clean\_corpus <- tm\_map(sms\_corpus, content\_transformer(tolower))

clean\_corpus <- tm\_map(clean\_corpus, removeNumbers)

clean\_corpus <- tm\_map(clean\_corpus, removePunctuation)

clean\_corpus <- tm\_map(clean\_corpus, removeWords, stopwords())

clean\_corpus <- tm\_map(clean\_corpus, stripWhitespace)

# Inspect cleaned corpus

inspect(clean\_corpus[1:3])

# Create document-term matrix

sms\_dtm <- DocumentTermMatrix(clean\_corpus)

str(sms\_dtm)

# Create wordclouds for spam and ham messages

wordcloud(clean\_corpus[which(sms\_spam\_df$category == "ham")], min.freq = 40)

wordcloud(clean\_corpus[which(sms\_spam\_df$category == "spam")], min.freq = 40)

# Split data into training and test sets

sms\_raw\_train <- sms\_spam\_df[1:4169,]

sms\_raw\_test <- sms\_spam\_df[4170:5559,]

sms\_dtm\_train <- sms\_dtm[1:4169,]

sms\_dtm\_test <- sms\_dtm[4170:5559,]

sms\_corpus\_train <- clean\_corpus[1:4169]

sms\_corpus\_test <- clean\_corpus[4170:5559]

# Create reduced DTM

five\_times\_words <- findFreqTerms(sms\_dtm\_train, 5)

sms\_train <- DocumentTermMatrix(sms\_corpus\_train, control = list(dictionary = five\_times\_words))

sms\_test <- DocumentTermMatrix(sms\_corpus\_test, control = list(dictionary = five\_times\_words))

# Convert counts to factors

convert\_count <- function(x) {

  factor(ifelse(x > 0, "Yes", "No"), levels = c("No", "Yes"))

}

sms\_train <- apply(sms\_train, 2, convert\_count)

sms\_test <- apply(sms\_test, 2, convert\_count)

# Train Naive Bayes classifier and make predictions

sms\_classifier <- naiveBayes(sms\_train, factor(sms\_raw\_train$category))

sms\_test\_pred <- predict(sms\_classifier, newdata = sms\_test)

# Evaluate accuracy

conf\_matrix <- table(sms\_test\_pred, sms\_raw\_test$category)

accuracy <- sum(diag(conf\_matrix)) / sum(conf\_matrix) \* 100

accuracy

**TW-4**

#required packages

#install.packages("caret")

#install.packages("tidyr")

# Attach the dataset to the environment

data(iris)

# Get help on the data

help(iris)

# Rename the data

iris\_dataset<-iris

# View the data

View(iris\_dataset)

# View the top few rows of the data in R console

head(iris\_dataset,7)

# Assigning meaningful column names

colnames(iris\_dataset)<-c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width","Species")

head(iris\_dataset,5)

# Load the Caret package which allows us to partition the data

library(caret)

# We use the dataset to create a partition (80% training 20% testing)

index <- createDataPartition(iris\_dataset$Species, p=0.80, list=FALSE)

# select 20% of the data for testing

testset <- iris\_dataset[-index,]

# select 80% of data to train the models

trainset <- iris\_dataset[index,]

# Dimensions of the data

dim(trainset)

# Structure of the data

str(trainset)

# Summary of the data

summary(trainset)

# Levels of the prediction column

levels(trainset$Species)

## Histogram graph

hist(trainset$Sepal.Width)

## Box plot to understand how the distribution varies by class of flower

par(mfrow=c(1,4))

for(i in 1:4) {

  boxplot(trainset[,i], main=names(trainset)[i])

}

#install.packages("ggplot2")

# begin by loading the library

library(ggplot2)

# Scatter plot

g <- ggplot(data=trainset, aes(x = Petal.Length, y = Petal.Width))

print(g)

g <-g +

  geom\_point(aes(color=Species, shape=Species)) +

  xlab("Petal Length") +

  ylab("Petal Width") +

  ggtitle("Petal Length-Width")+

  geom\_smooth(method="lm")

print(g)

## Box Plot

box <- ggplot(data=trainset, aes(x=Species, y=Sepal.Length)) +

  geom\_boxplot(aes(fill=Species)) +

  ylab("Sepal Length") +

  ggtitle("Iris Boxplot") +

  stat\_summary(fun.y=mean, geom="point", shape=5, size=4)

print(box)

library(ggthemes)

## Histogram

histogram <- ggplot(data=iris, aes(x=Sepal.Width)) +

  geom\_histogram(binwidth=0.2, color="black", aes(fill=Species)) +

  xlab("Sepal Width") +

  ylab("Frequency") +

  ggtitle("Histogram of Sepal Width")+

  theme\_economist()

print(histogram)

## Faceting: Producing multiple charts in one plot

library(ggthemes)

facet <- ggplot(data=trainset, aes(Sepal.Length, y=Sepal.Width, color=Species))+

  geom\_point(aes(shape=Species), size=1.5) +

  geom\_smooth(method="lm") +

  xlab("Sepal Length") +

  ylab("Sepal Width") +

  ggtitle("Faceting") +

  theme\_fivethirtyeight() +

  facet\_grid(. ~ Species) # Along rows

print(facet)