Online Cryptography Course

Patrick Atak

18/07/2020

Part I

Research Question

A Kenyan entrepreneur has created an online cryptography course and would want to advertise it on her blog. She currently targets audiences originating from various countries. In the past, she ran ads to advertise a related course on the same blog and collected data in the process. She would now like to employ your services as a Data Science Consultant to help her identify which individuals are most likely to click on her ads.

Metric of Success

In order to work on the above problem, I need to do the following:

1.Find and deal with outliers, anomalies, and missing data within the dataset.

2.Perform univariate and bivariate analysis and supervised learning while recording your observations.

3.From your insights provide a conclusion and recommendation.

Reading in the CSV dataset

```
ads <- read.csv('advertising.csv')
head(ads)
```

==========

Checking for Outliers, Anomalies and Missing data within the dataset.

```
# Identifying missing data
is.na(ads)

# Finding the total missing values
colSums(is.na(ads))

# Dealing with the missing
```

```r
na.omit(ads)

#There seems to be no missing values in this dataset
```

==========

Handling of Outliers

```r
# Using a boxplot to visualise any existing outlier.
# Boxplot about the daily internet usage.
boxplot(ads$Daily.Internet.Usage)

# Function boxplot.stats which lists the outliers in the vectors.
boxplot.stats(ads$Daily.Internet.Usage)$out

# From the plot and the stats function, there seems to be no outlier present.
```

==========

Handling the duplicated data

```r
# Using duplicated() function to check for duplicates across rows.
dupl_ads_rows = ads[duplicated(ads),]

# Using the unique() function to remove the duplicated rows.
unique_ads_rows = unique(ads)
```

Getting the statistical summary of the data

```r
# Statistics of the data
summary(ads)

# Structure of the data
str(ads)
```

==========

Performing Univariate EDA & Graphical EDA

```r
# Performing an analysis of a single variable. (Area Income).
# calculating the mean.
x <- unique_ads_rows$Area.Income
avg <- mean(x)
```

```r
# calculating the median.
mid <- median(x)

# calculating the mode.
getmode <- function(v){
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v,uniqv)))]
}
most <- getmode(x)


# calculating the maximum.
high <- max(x)

# calculating the minimum.
low <- min(x)

# calculating the range.
rng <- range(x)

# calculating the quantile.
qtile <- quantile(x)

# calculating the variance.
vari <- var(x)

# calculating the standard deviation.
stdd <- sd(x)
```

```r
# Performing the Univariate Graphical Plots
# Box Plot
boxplot(x)

# getting the frequency table
area_income <- unique_ads_rows$Area.Income
income_frequecy <- table(area_income)

# barplot of the area income
barplot(income_frequency)

# histogram of the area income
hist(income_frequency)
```

==========

Performing Bivariate EDA & Graphical EDA

```r
# using two variables. Area Income & Daily Internet Usage
daily_internet_use <- unique_ads_rows$Daily.Internet.Usage
#
# performing a covariance
```

```r
cov(area_income, daily_internet_use)

# correlation coefficient
cor(area_income, daily_internet_use)

# creating a scatter plot
plot(area_income, daily_internet_use, xlab="Area Income", ylab="Daily Internet Use")
```

==========

Including Plots

```r
# creating a scatter plot of Area Income vs Daily Internet Usage
plot(area_income, daily_internet_use, xlab="Area Income", ylab="Daily Internet Use")

# creating a scatter plot of Area Income vs Daily Time Spent on The Site
plot(ads$Area.Income, ads$Daily.Time.Spent.on.Site, xlab="Area Income", ylab="Daily Time Spent on Site")

# install the GGally package
library(GGally)
library(ggplot2)
# visualise the correlation matrix
ggcorr(ads, method = c("everything", "pearson"))
```

==========

Performing Supervised Learning

Simple Linear Regression

```r
# Simple Linear Regression

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)

set.seed(123)

split = sample.split(ads$Clicked.on.Ad, SplitRatio = 0.8)
training_set = subset(ads, split == TRUE)
test_set = subset(ads, split == FALSE)

# Feature Scaling
# training_set = scale(training_set)
# test_set = scale(test_set)

# Fitting Simple Linear Regression to the Training set
regressor = lm(formula = Clicked.on.Ad ~ Daily.Time.Spent.on.Site,
               data = training_set)
```

```r
# Predicting the Test set results
y_pred = predict(regressor, newdata = test_set)

# Summary
summary(regressor)
```

==========

Multiple Linear Regression

```r
# Multiple Linear Regression

# Selecting the multiple columns
input <- ads[,c('Daily.Time.Spent.on.Site', 'Daily.Internet.Usage','Age')]

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(ads$Clicked.on.Ad, SplitRatio = 0.8)
multi_training_set = subset(ads, split == TRUE)
multi_test_set = subset(ads, split == FALSE)

# Feature Scaling
# multi_training_set = scale(training_set)
# multi_test_set = scale(test_set)

# Fitting Multiple Linear Regression to the Training set
multi_regressor = lm(formula = Clicked.on.Ad ~ Daily.Time.Spent.on.Site + Daily.Internet.Usage + Age, da

# Predicting the Test set results
y_multi_pred = predict(multi_regressor, newdata = multi_test_set)
```

==========

K-Nearest Neighbours

```r
set.seed(1234)

# Randomizing the rows, creates a uniform distribution of 150
random <- runif(150)
ads_random <- ads[order(random),]

# Selecting the first 6 rows from iris_random
head(ads_random)
```

```r
# Normalizing the numerical variables of the data set. Normalizing the numerical values is really effec
# as it provides a measure from 0 to 1 which corresponds to min value to the max value of the data colu
```

```r
# We define a normal function which will normalize the set of values according to its minimum value and
normal <- function(x) (
  return( ((x - min(x)) /(max(x)-min(x))) )
)
normal(1:5)
ads_new <- as.data.frame(lapply(ads_random[,1:4], normal))
summary(ads_new)
```

```r
# Lets now create test and train data sets

train_knn <- ads_new[1:800,]
test_knn <- ads_new[801:1000,]
train_sp_knn <- ads_new[1:800,10]
test_sp_knn <- ads_new[801:1000,10]
```

```r
# Now we can use the K-NN algorithm. Lets call the "class" package which contains the K-NN algorithm.
# We then have to provide 'k' value which is no of nearest neighbours(NN) to look for
# in order to classify the test data point.
# Lets build a model on it; cl is the class of the training data set and k is the no of neighbours to l
# in order to classify it accordingly.

library(class)
require(class)
model <- knn(train= train_knn,test=test_knn, ,cl= train_sp_knn,k=13)
table(factor(model))
table(test_sp_knn,model)
```

==========

Decision Tree Classifiers

```r
# Decision Tree Classification

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')

library(caTools)
set.seed(123)


# Fitting Decision Tree Classification to the Training set
# install.packages('rpart')

library(rpart)

classifier = rpart(Clicked.on.Ad ~ Daily.Time.Spent.on.Site + Age + Area.Income + Daily.Internet.Usage,

# rpart plot
#rpart.plot(classifier)
```

```r
# Plotting the tree
plot(classifier)
text(classifier)
```

==========

Support Vector Machines

```r
# We first install the caret package.
# This package will be very helfpul in providing us with
# direct access to various functions for training our model
# with various machine learning algorithms such
# as KNN, SVM, Decision Tree, Linear Regression etc.
# ---
#
install.packages('caret')
library(caret)
```

```r
# Next we split the data into training set and testing set.
#
# ---
# - The "y" parameter takes the value of variable according to which data needs to be partitioned.
# In our case, target variable is at Clicked.on.Ad, so we are passing ads$Clicked.on.Ad
# - The "p" parameter holds a decimal value in the range of 0-1. It's to show the percentage of the spl
# We are using p=0.7. It means that data split should be done in 70:30 ratio.
# So, 70% of the data is used for training and the remaining 30% is for testing the model.
# - The "list" parameter is for whether to return a list or matrix.
# We are passing FALSE for not returning a list
# ---
#
intrain <- createDataPartition(y = ads$Clicked.on.Ad, p= 0.7, list = FALSE)
train_svm <- ads[intrain,]
test_svm <- ads[-intrain,]
```

```r
# We check the dimensions of out training dataframe and testing dataframe
# ---
#
dim(train_svm);
dim(test_svm);
```

```r
# We then clean the data using the anyNA() method that checks for any null values.
# ---
#
anyNA(ads)
```

```r
# Then check the summary of our data by using the summary() function
# ---
#
summary(ads)
```

```r
# Before we train our model we will need to control all the computational overheads.
# We will implement this through the trainControl() method.
# This will allow us to use the train() function provided by the caret package.
# ---
# The trainControl method will take three parameters:
# a) The "method" parameter defines the resampling method,
# in this demo we'll be using the repeatedcv or the repeated cross-validation method.
# b) The next parameter is the "number", this basically holds the number of resampling iterations.
# c) The "repeats " parameter contains the sets to compute for our repeated cross-validation.
# We are using setting number =10 and repeats =3
# ---
#
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

svm_Linear <- train(Clicked.on.Ad ~ Daily.Time.Spent.on.Site + Age + Area.Income + Daily.Internet.Usage
trControl=trctrl,
preProcess = c("center", "scale"),
tuneLength = 10)
```

```r
# We can then check the reult of our train() model as shown below
# ---
#
svm_Linear
```

```r
# We can use the predict() method for predicting results as shown below.
# We pass 2 arguements, our trained model and our testing data frame.
# ---
#
test_pred <- predict(svm_Linear, newdata = test_knn)
test_pred
```

```r
# Now checking for our accuracy of our model by using a confusion matrix
# ---
#
confusionMatrix(table(test_pred, test_knn$Clicked.on.Ad))
```

==========

Naive Bayes Classifier

```r
# We will now install and load the required packages
# ---
#
install.packages('tidyverse')
library(tidyverse)

install.packages('ggplot2')
library(ggplot2)

install.packages('caret')
```

```
library(caret)

install.packages('caretEnsemble')
library(caretEnsemble)

install.packages('psych')
library(psych)

install.packages('Amelia')
library(Amelia)

install.packages('mice')
library(mice)

install.packages('GGally')
library(GGally)

install.packages('rpart')
library(rpart)

install.packages('randomForest')
library(randomForest)
```

```
# Splitting data into training and test data sets
# ---
#
indxTrain <- createDataPartition(y = ads$Clicked.on.Ad,p = 0.7,list = FALSE)
train_naive <- data[indxTrain,]
test_naive <- data[-indxTrain,]
```

```
# Comparing the outcome of the training and testing phase
# ---
# Creating objects x which holds the predictor variables and y which holds the response variables
# ---
#
x = training[,-9]
y = training$Clicked.on.Ad
```

```
# Loading our inbuilt e1071 package that holds the Naive Bayes function.
# ---
install.packages('e1071')
library(e1071)
```

```
# Now building our model
# ---
#
model = train(x,y,'nb',trControl=trainControl(method='cv',number=10))
```

```
# Model Evalution
# ---
# Predicting our testing set
```

```r
#
Predict <- predict(model,newdata = testing)

# Getting the confusion matrix to see accuracy value and other parameter values
# ---
#
confusionMatrix(Predict, testing$Clicked.on.Ad)
```