machine_learning

John J Padamadan

2024-05-16

```r
data <- read.csv('dataset/Social_Network_Ads.csv')

head(data)
```

```
##    User.ID Gender Age EstimatedSalary Purchased
## 1 15624510   Male  19           19000         0
## 2 15810944   Male  35           20000         0
## 3 15668575 Female  26           43000         0
## 4 15603246 Female  27           57000         0
## 5 15804002   Male  19           76000         0
## 6 15728773   Male  27           58000         0
```

```r
data <- data[,-1]

head(data)
```

```
##   Gender Age EstimatedSalary Purchased
## 1   Male  19           19000         0
## 2   Male  35           20000         0
## 3 Female  26           43000         0
## 4 Female  27           57000         0
## 5   Male  19           76000         0
## 6   Male  27           58000         0
```

```r
# Count the occurrences of each unique value in the 'Purchased' column
table(data$Purchased)
```

```
##
##   0   1
## 257 143
```

```r
# Calculate the proportion of each unique value in the 'Purchased' column
prop.table(table(data$Purchased))
```

```
##
##      0      1
## 0.6425 0.3575
```

```r
dim(data)
```

```
## [1] 400   4
```

cehcking missing values

```r
# Count the number of missing values in each column of the dataframe
na_count <- colSums(is.na(data))
```

```
# Print the result
print(na_count)
```

```
##          Gender             Age EstimatedSalary       Purchased
##               0               0               0               0
```

one hot encoding

```
# Perform one-hot encoding on the "Gender" column
data_one_hot <- cbind(data, model.matrix(~ Gender + 0, data = data))
# Remove the original "Gender" column
data_one_hot <- data_one_hot[, !(names(data_one_hot) %in% c("Gender"))]
# Display the first few rows of the dataset after encoding
head(data_one_hot)
```

```
##   Age EstimatedSalary Purchased GenderFemale GenderMale
## 1  19           19000         0            0          1
## 2  35           20000         0            0          1
## 3  26           43000         0            1          0
## 4  27           57000         0            1          0
## 5  19           76000         0            0          1
## 6  27           58000         0            0          1
```

```
# Encoding the target feature as factor
data_one_hot$Purchased = factor(data_one_hot$Purchased, levels = c(0, 1))
```

Splitting the dataset into the Training set and Test set

```
library(caTools)
# The caTools package in R provides various utility functions for data splitting, sampling, and manipul
```

```
set.seed(123)
split = sample.split(data_one_hot$Purchased, SplitRatio = 0.75)
training_set = subset(data_one_hot, split == TRUE)
test_set = subset(data_one_hot, split == FALSE)
```

Feature Scaling

```
# Standard scaling
training_set[, c(1,2)] = scale(training_set[, c(1,2)]) # Exclude 3rd column from scaling
test_set[, c(1,2)] = scale(test_set[, c(1,2)])
```

Logistic regression model

```
# Fitting Logistic Regression to the Training set
classifier = glm(formula = Purchased ~ ., # GLM - Generalized Linear Models
                 family = binomial, # binomial indicates that logistic regression will be used
                 data = training_set)
```

Doing prediction

```r
prob_pred = predict(classifier, type = 'response', newdata = test_set[-3])
y_pred = ifelse(prob_pred > 0.5, 1, 0)
```

Model evaluation

```r
# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred>0.5)

print(c)
```

```
## function (...)  .Primitive("c")
```

```r
# Calculate accuracy from confusion matrix
accuracy <- sum(diag(cm)) / sum(cm)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.82"
```

kNN model

```r
# To find the optimum number of k value

library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
# Define training control
ctrl <- trainControl(method = "cv",   # Use cross-validation
                     number = 10)     # Number of folds for cross-validation

# Define range of k values to evaluate
k_values <- c(1, 3, 5, 7, 9, 11, 13, 15)  # Example range of k values

# Train KNN model using cross-validation
knn_model <- train(Purchased ~ .,               # Formula for the model
                   data = training_set,         # Training data
                   method = "knn",              # KNN algorithm
                   trControl = ctrl,            # Training control with cross-validation
                   tuneGrid = data.frame(k = k_values))  # Grid of k values to evaluate

# Get optimal k value
optimal_k <- knn_model$bestTune$k
print(paste("Optimal k value:", optimal_k))
```

```
## [1] "Optimal k value: 7"
```

```r
# Fitting K-NN to the Training set and Predicting the Test set results
library(class)

y_pred = knn(train = training_set[, -3],
             test = test_set[, -3],
             cl = training_set[, 3],
             k = 7,
```

```r
             prob = TRUE)
```

Model evaluation

```r
# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)

print(cm)
```

```
##    y_pred
##      0  1
##   0 58  6
##   1  6 30
```

```r
# Calculate accuracy from confusion matrix
accuracy <- sum(diag(cm)) / sum(cm)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.88"
```

```r
library(e1071)
```

```r
classifier = svm(formula = Purchased ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')
```

Doing prediction

```r
# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])
```

Model evaluation

```r
# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)

print(cm)
```

```
##    y_pred
##      0  1
##   0 58  6
##   1 14 22
```

```r
# Calculate accuracy from confusion matrix
accuracy <- sum(diag(cm)) / sum(cm)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.8"
```