Online News Popularity Analysis - Machine Learning Assignment

Fahad

2023-11-01

```
##Referneces and Websites for The codes below. Most codes have been taken collectively from different s

#https://rpubs.com/namitakadam28/335507
#PG Bank Case
#https://rpubs.com/bipinkg/666804
#https://jtr13.github.io/cc21fall2/feature-selection-in-r.html
#https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/
#ChatGPT
#Stackoverflow
#Github
#https://www.kaggle.com/code/thehapyone/exploratory-analysis-for-online-news-popularity
```

```
#Load the data
dta <- read.csv("OnlineNewsPopularity.csv")
```

```
## Exploratory Data analysis and Data cleaning

#Check for structure and missing values.We can confirm that there are no missing values.
sum(is.na(dta))
```

```
## [1] 0
```

```
# We will remove URL and timedelta because they are non-predictive variables and we will remove is.week

dta1 <- subset( dta, select = -c(url, timedelta, is_weekend, n_unique_tokens, n_non_stop_words, n_non_s

#So we have few factor columns when checked from the structure of data so we need to convert them to nu
dta1$data_channel_is_lifestyle<- as.numeric(dta1$data_channel_is_lifestyle)
dta1$data_channel_is_entertainment<- as.numeric(dta1$data_channel_is_entertainment)
dta1$data_channel_is_bus<- as.numeric(dta1$data_channel_is_bus)
dta1$data_channel_is_socmed<- as.numeric(dta1$data_channel_is_socmed)
dta1$data_channel_is_tech<- as.numeric(dta1$data_channel_is_tech)
dta1$data_channel_is_world<- as.numeric(dta1$data_channel_is_world)
dta1$weekday_is_monday<- as.numeric(dta1$weekday_is_monday)
dta1$weekday_is_tuesday<- as.numeric(dta1$weekday_is_tuesday)
dta1$weekday_is_wednesday<- as.numeric(dta1$weekday_is_wednesday)
dta1$weekday_is_thursday<- as.numeric(dta1$weekday_is_thursday)
dta1$weekday_is_friday<- as.numeric(dta1$weekday_is_friday)
dta1$weekday_is_saturday<- as.numeric(dta1$weekday_is_saturday)
dta1$weekday_is_sunday<- as.numeric(dta1$weekday_is_sunday)
```

```r
##Correlation Matrix for Feature Selection


#Now we will run correlation and see for all the x values that are most correlated among each other and

#Multicollinearity: High correlation between features can lead to multicollinearity in linear models li

#Overfitting: Including highly correlated features can lead to overfitting, where the model fits the tr

set.seed(7)
# load the library
library(mlbench)
library(caret)
```

```
## Loading required package: ggplot2
```

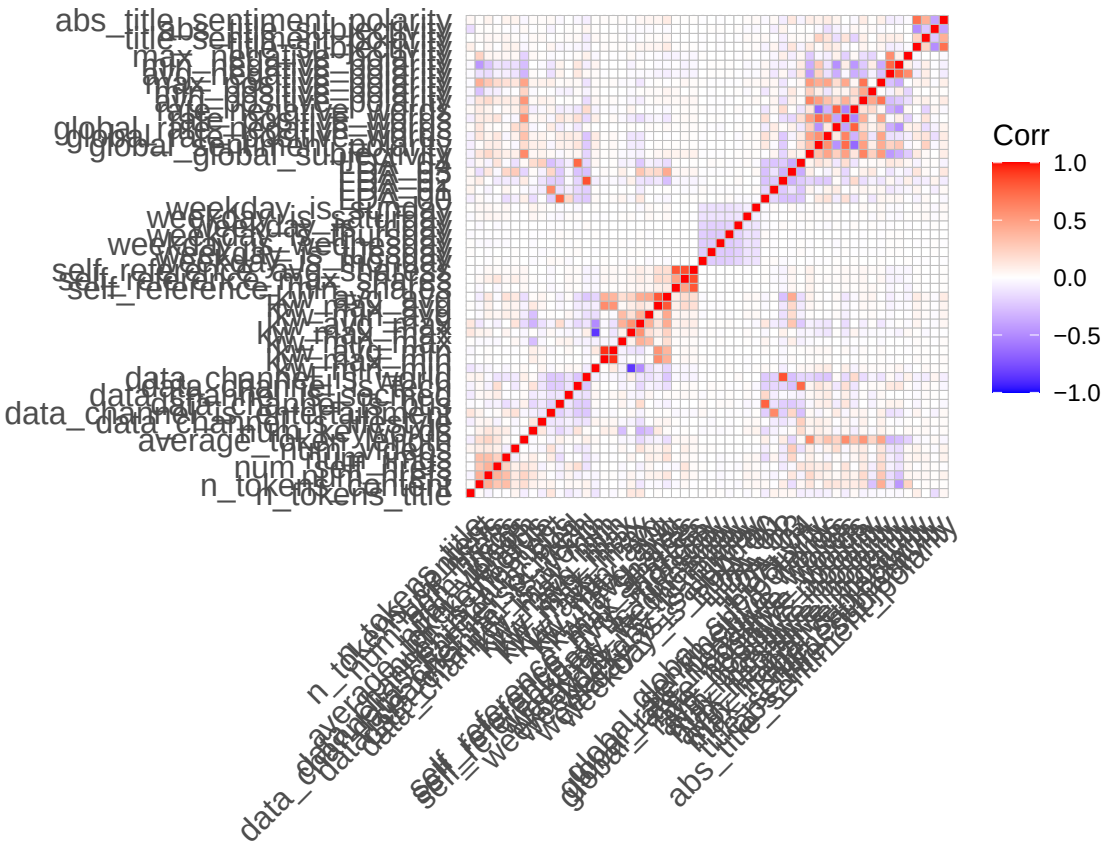
```
## Loading required package: lattice
```

```r
library(ggplot2)
library(lattice)

# calculate correlation matrix
correlationMatrix <- cor(dta1[,1:54])
# find attributes that are highly corrected (ideally >0.75)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.75)
# print indexes of highly correlated attributes
print(highlyCorrelated)
```

```
## [1] 23 14 44 34 15 25 16 24
```

```r
library(ggcorrplot)
ggcorrplot(correlationMatrix)
```




```
# From correlation matrix we got 10 columns that we need to remove and we will create now dta2 with omi

columns_to_remove <- c(23, 14, 44, 34, 15, 25, 16, 24)
dta2 <- dta1[, -columns_to_remove]
```

```
## Model 1: Linear Regression Model

# Load the required libraries
library(caret)

set.seed(123)

# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta2$shares, p = 0.7, list = FALSE)
training_data <- dta2[index, ]
testing_data <- dta2[-index, ]

# Train the linear regression model
linear_model <- lm(shares ~ ., data = training_data)

# Make predictions on the testing data
predictions <- predict(linear_model, newdata = testing_data)

# Calculate Mean Absolute Error (MAE)
mae <- mean(abs(predictions - testing_data$shares))
cat("Mean Absolute Error (MAE):", mae, "\n")
```

```
## Mean Absolute Error (MAE): 3085.7
```

```
# Calculate Mean Squared Error (MSE)
mse <- mean((predictions - testing_data$shares)^2)
cat("Mean Squared Error (MSE):", mse, "\n")
```

```
## Mean Squared Error (MSE): 108797503
```

```
# Calculate R-squared (R2)
ssr <- sum((predictions - testing_data$shares)^2)
sst <- sum((testing_data$shares - mean(testing_data$shares))^2)
r_squared <- 1 - (ssr / sst)
cat("R-squared (R2):", r_squared, "\n")
```

```
## R-squared (R2): 0.01561256
```

```
#This model is not right for this specific case and we cant use it because the value for R square is ve
```

```
##Model 2: Logistic Regression

# Load the required libraries
library(caret)

set.seed(123)

# Define a threshold for popularity (e.g., more than 1400 shares)
threshold <- 1400

# Create a binary outcome variable 'popular' based on the threshold
dta2$popular <- ifelse(dta2$shares > threshold, 1, 0)
dta3 <- data <- subset(dta2, select = -shares)

# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta3$popular, p = 0.7, list = FALSE)
training_data <- dta3[index, ]
testing_data <- dta3[-index, ]

# Train the logistic regression model
logistic_model <- glm(popular ~ ., data = training_data, family = binomial)

# Make predictions on the testing data
predictions <- predict(logistic_model, newdata = testing_data, type = "response")

# Convert predicted probabilities to binary outcomes (e.g., popular or not popular)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Calculate accuracy
accuracy <- mean(predicted_classes == testing_data$popular)
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.6318843
```

```r
#Logistic regression correctly predicted that out of all the samples in the test dataset, the model mad
```

```r
## Model 3: Decision Tree
# Load the required libraries

library(rpart)
set.seed(123)

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

index <- createDataPartition(dta2$shares, p = 0.7, list = FALSE)
training_data <- dta2[index, ]
testing_data <- dta2[-index, ]

# Define the formula for the decision tree model
formula <- shares ~ .

# Train the decision tree model
tree_model <- rpart(formula, data = training_data, method = "anova")

# Make predictions on the testing data
predictions <- predict(tree_model, newdata = testing_data)

# Calculate Mean Absolute Error (MAE) for decision tree
tree_mae <- mean(abs(predictions - testing_data$shares))
cat("Decision Tree Mean Absolute Error (MAE):", tree_mae, "\n")
```

```
## Decision Tree Mean Absolute Error (MAE): 2574.197
```

```r
# Calculate Mean Squared Error (MSE) for decision tree
tree_mse <- mean((predictions - testing_data$shares)^2)
cat("Decision Tree Mean Squared Error (MSE):", tree_mse, "\n")
```

```
## Decision Tree Mean Squared Error (MSE): 104507545
```

```r
# Calculate R-squared (R2) for decision tree
tree_r_squared <- 1 - (sum((predictions - testing_data$shares)^2) / sum((testing_data$shares - mean(tes
cat("Decision Tree R-squared (R2):", tree_r_squared, "\n")
```

```
## Decision Tree R-squared (R2): 0.05442761
```

```r
# Create a confusion matrix
confusion_matrix <- table(Actual = testing_data$shares, Predicted = predictions)


# Calculate sensitivity and specificity
TP <- confusion_matrix[2, 2]  # True Positives
TN <- confusion_matrix[1, 1]  # True Negatives
FP <- confusion_matrix[1, 2]  # False Positives
FN <- confusion_matrix[2, 1]  # False Negatives
```

```r
# Sensitivity (True Positive Rate)
sensitivity <- TP / (TP + FN)

# Specificity (True Negative Rate)
specificity <- TN / (TN + FP)

# Accuracy
accuracy <- (TP + TN) / (TP + TN + FP + FN)

cat("Sensitivity (True Positive Rate):", sensitivity, "\n")
```

```
## Sensitivity (True Positive Rate): 0
```

```r
cat("Specificity (True Negative Rate):", specificity, "\n")
```

```
## Specificity (True Negative Rate): 1
```

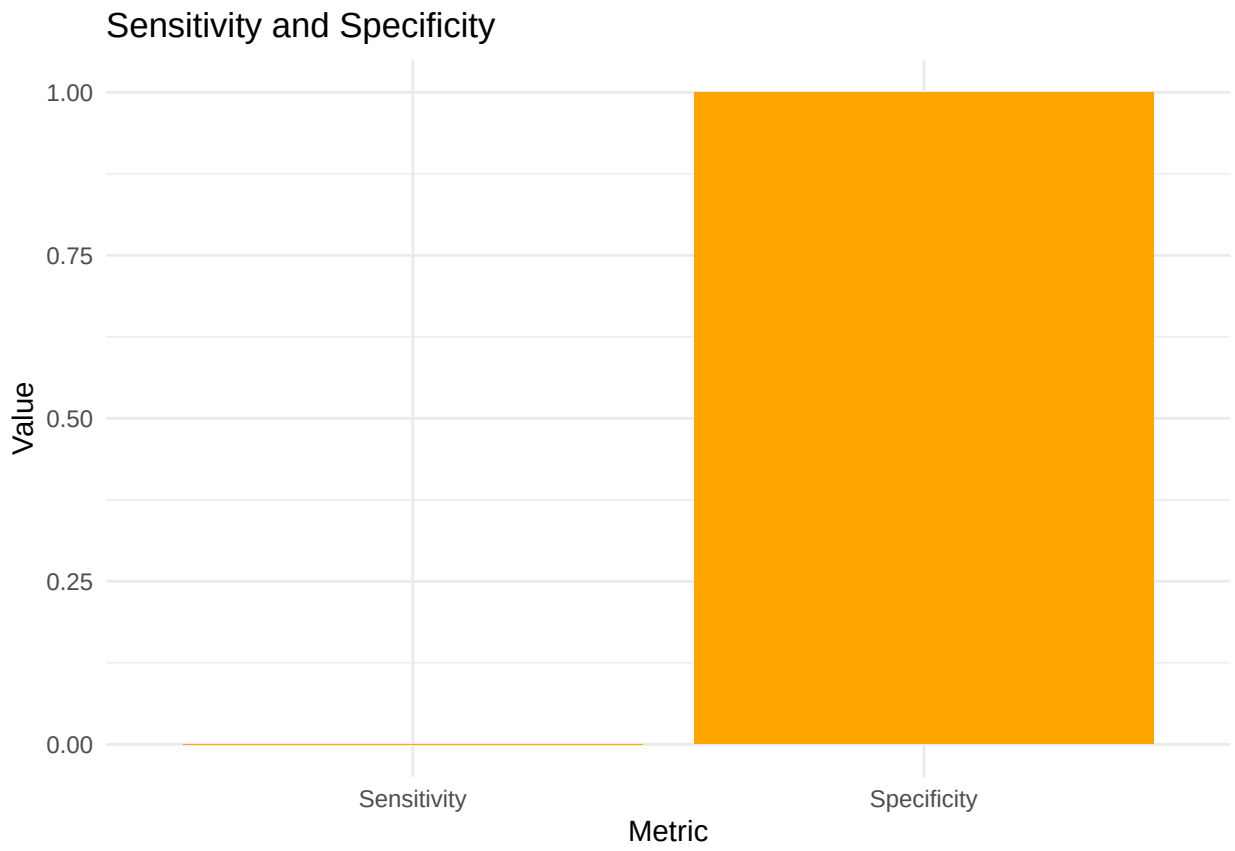
```r
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.5
```

```r
#Visual for specificity and sensitivity.
# Create a data frame with sensitivity and specificity
results <- data.frame(Metric = c("Sensitivity", "Specificity"),
                      Value = c(sensitivity, specificity))

# Load the ggplot2 library
library(ggplot2)

# Create a bar chart
ggplot(results, aes(x = Metric, y = Value)) +
  geom_bar(stat = "identity", fill = "orange") +
  ylim(0, 1) +  # Set the y-axis limits from 0 to 1
  labs(y = "Value") +
  ggtitle("Sensitivity and Specificity") +
  theme_minimal()
```




```
# This model is also not right for this case because it gave an accuracy of 0.5 which means that the mo
```

```
## Model 4: Random Forest

# Load the required library
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta2$shares, p = 0.7, list = FALSE)
training_data <- dta2[index, ]
testing_data <- dta2[-index, ]

# Train the Random Forest model for binary classification
```

```r
rf_model <- randomForest(shares ~ ., data = training_data, ntree = 100)

# Make predictions on the testing data
predictions <- predict(rf_model, newdata = testing_data, type = "response")

# Convert predicted probabilities to binary outcomes (e.g., popular or not popular)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Calculate accuracy for binary classification
accuracy <- mean(predicted_classes == testing_data$popular)
cat("Random Forest Accuracy:", accuracy, "\n")
```

```
## Random Forest Accuracy: 0.493441
```

```r
### Now we will do all models with new set of features using XGboost feature selection method.
```

```r
## To assess how different models respond to a new set of variables, we will conduct a fresh round of f

## XGBoost Feature Selection method

# Install and load the required library

library(xgboost)
library(caret)

# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta2$shares, p = 0.7, list = FALSE)
training_data <- dta2[index, ]
testing_data <- dta2[-index, ]

# Define the formula for XGBoost and create DMatrix
X_train <- xgb.DMatrix(data = as.matrix(training_data[, -1]), label = training_data$shares)

# Specify XGBoost parameters
xgb_params <- list(
  objective = "reg:squarederror",
  booster = "gbtree",
  eval_metric = "mae"
)

# Train the XGBoost model for feature selection
xgb_model <- xgboost(data = X_train, params = xgb_params, nrounds = 100)
```

```
## [1]  train-mae:2418.283242
## [2]  train-mae:1706.172786
## [3]  train-mae:1207.569941
## [4]  train-mae:854.937306
## [5]  train-mae:604.740230
## [6]  train-mae:427.916206
## [7]  train-mae:304.664406
## [8]  train-mae:217.915481
## [9]  train-mae:155.819216
```

```
## [10] train-mae:113.268634
## [11] train-mae:85.090591
## [12] train-mae:67.216082
## [13] train-mae:56.021626
## [14] train-mae:48.158106
## [15] train-mae:43.391990
## [16] train-mae:40.513799
## [17] train-mae:38.819040
## [18] train-mae:37.587155
## [19] train-mae:36.563499
## [20] train-mae:35.063857
## [21] train-mae:33.987595
## [22] train-mae:33.440751
## [23] train-mae:33.131374
## [24] train-mae:32.597296
## [25] train-mae:32.261709
## [26] train-mae:32.045027
## [27] train-mae:31.768764
## [28] train-mae:30.471539
## [29] train-mae:29.527639
## [30] train-mae:29.421594
## [31] train-mae:29.183613
## [32] train-mae:29.033821
## [33] train-mae:28.794946
## [34] train-mae:28.246881
## [35] train-mae:27.491124
## [36] train-mae:26.771384
## [37] train-mae:26.492230
## [38] train-mae:26.267407
## [39] train-mae:26.074472
## [40] train-mae:25.933519
## [41] train-mae:25.173936
## [42] train-mae:24.662992
## [43] train-mae:23.650544
## [44] train-mae:21.909046
## [45] train-mae:21.111091
## [46] train-mae:20.979781
## [47] train-mae:20.041469
## [48] train-mae:19.866140
## [49] train-mae:19.786565
## [50] train-mae:19.575660
## [51] train-mae:19.497380
## [52] train-mae:18.845500
## [53] train-mae:18.780044
## [54] train-mae:18.154025
## [55] train-mae:17.861935
## [56] train-mae:17.711548
## [57] train-mae:17.585962
## [58] train-mae:17.461174
## [59] train-mae:17.166629
## [60] train-mae:17.017738
## [61] train-mae:16.166473
## [62] train-mae:15.954423
## [63] train-mae:14.850050
```

```
## [64] train-mae:14.368950
## [65] train-mae:13.911735
## [66] train-mae:13.477128
## [67] train-mae:13.318702
## [68] train-mae:13.014793
## [69] train-mae:12.682477
## [70] train-mae:12.592811
## [71] train-mae:12.422017
## [72] train-mae:12.386416
## [73] train-mae:12.246620
## [74] train-mae:11.938405
## [75] train-mae:11.819948
## [76] train-mae:11.723133
## [77] train-mae:10.967065
## [78] train-mae:10.595320
## [79] train-mae:10.501997
## [80] train-mae:10.313747
## [81] train-mae:10.083968
## [82] train-mae:9.821363
## [83] train-mae:9.614438
## [84] train-mae:9.532096
## [85] train-mae:9.204823
## [86] train-mae:8.566332
## [87] train-mae:8.437901
## [88] train-mae:8.339309
## [89] train-mae:8.288652
## [90] train-mae:7.889229
## [91] train-mae:7.801529
## [92] train-mae:7.530420
## [93] train-mae:7.363115
## [94] train-mae:7.239551
## [95] train-mae:7.107815
## [96] train-mae:7.047728
## [97] train-mae:6.962417
## [98] train-mae:6.845480
## [99] train-mae:6.811204
## [100]    train-mae:6.691242
```

```
# Get feature importance scores
importance_scores <- xgb.importance(model = xgb_model)

# Print the feature importance scores
print(importance_scores)
```

```
##                          Feature         Gain        Cover    Frequency
##  1:                       shares 9.981886e-01 9.744605e-01 0.6384083045
##  2:                    num_hrefs 1.608407e-03 1.318167e-04 0.0255190311
##  3:            n_tokens_content 1.290030e-04 5.081556e-03 0.0532006920
##  4:          global_subjectivity 2.264727e-05 1.196797e-04 0.0125432526
##  5: abs_title_sentiment_polarity 7.228667e-06 1.913546e-04 0.0051903114
##  6:   self_reference_avg_sharess 4.347689e-06 1.385193e-04 0.0116782007
##  7:                       LDA_02 3.556228e-06 6.015383e-04 0.0129757785
##  8:                  kw_avg_min 2.939979e-06 5.713467e-04 0.0142733564
##  9:                  kw_min_max 2.932538e-06 1.769230e-05 0.0047577855
```

```
## 10:                   kw_min_avg 2.872393e-06 1.315510e-03 0.0056228374
## 11:    global_sentiment_polarity 2.793402e-06 1.538325e-03 0.0134083045
## 12:                       LDA_01 2.635373e-06 1.904006e-03 0.0147058824
## 13:        min_positive_polarity 2.614589e-06 3.240167e-04 0.0064878893
## 14:                       LDA_04 2.460281e-06 2.175610e-04 0.0099480969
## 15:                   kw_max_avg 2.089647e-06 7.012311e-04 0.0190311419
## 16:   global_rate_positive_words 2.040829e-06 1.700997e-04 0.0077854671
## 17:          rate_positive_words 1.801766e-06 1.547624e-04 0.0077854671
## 18:                       LDA_03 1.715815e-06 5.200390e-03 0.0108131488
## 19:                 num_keywords 1.620872e-06 6.026252e-05 0.0069204152
## 20:                   num_videos 1.315646e-06 5.138980e-03 0.0090830450
## 21:                     num_imgs 1.143558e-06 1.669598e-04 0.0155709343
## 22:     title_sentiment_polarity 8.151528e-07 1.442557e-04 0.0069204152
## 23:        max_negative_polarity 7.620549e-07 4.263060e-05 0.0038927336
## 24:                   kw_avg_max 6.926910e-07 5.941112e-04 0.0121107266
## 25:                   kw_max_max 6.122973e-07 1.400892e-05 0.0025951557
## 26:         average_token_length 4.440051e-07 2.225728e-04 0.0134083045
## 27:                num_self_hrefs 3.984780e-07 8.960880e-05 0.0125432526
## 28:        avg_positive_polarity 3.722926e-07 1.229404e-04 0.0069204152
## 29:           title_subjectivity 2.543631e-07 3.321081e-05 0.0038927336
## 30:        avg_negative_polarity 2.100199e-07 1.562116e-04 0.0095155709
## 31:          data_channel_is_bus 2.028871e-07 3.321081e-06 0.0017301038
## 32:   global_rate_negative_words 1.720058e-07 1.774665e-04 0.0077854671
## 33:       abs_title_subjectivity 8.543854e-08 1.576004e-05 0.0030276817
## 34:    data_channel_is_lifestyle 5.132093e-08 2.862168e-05 0.0008650519
## 35:        min_negative_polarity 4.204236e-08 1.425046e-05 0.0021626298
## 36:            weekday_is_friday 3.746183e-08 2.143607e-05 0.0025951557
## 37:        max_positive_polarity 2.506086e-08 8.695194e-06 0.0012975779
## 38:          weekday_is_thursday 2.115930e-08 1.268049e-05 0.0012975779
## 39:          weekday_is_saturday 1.286512e-08 5.434496e-07 0.0004325260
## 40:            weekday_is_monday 9.511719e-09 8.997110e-05 0.0004325260
## 41:            weekday_is_sunday 1.127542e-09 4.830663e-07 0.0004325260
## 42:         weekday_is_wednesday 9.550266e-10 1.147283e-06 0.0004325260
##                         Feature         Gain        Cover    Frequency
```

```r
# Sort the importance scores in descending order
sorted_scores <- importance_scores[order(-importance_scores[, "Gain"]), ]

# Select the top N features with the highest gain (e.g., top 10 features)
top_n_features <- head(sorted_scores, n = 20 )

# Print the top features
print(top_n_features)
```

```
##                         Feature         Gain        Cover   Frequency
##  1:                      shares 9.981886e-01 9.744605e-01 0.638408304
##  2:                   num_hrefs 1.608407e-03 1.318167e-04 0.025519031
##  3:              n_tokens_content 1.290030e-04 5.081556e-03 0.053200692
##  4:           global_subjectivity 2.264727e-05 1.196797e-04 0.012543253
##  5: abs_title_sentiment_polarity 7.228667e-06 1.913546e-04 0.005190311
##  6:   self_reference_avg_sharess 4.347689e-06 1.385193e-04 0.011678201
##  7:                       LDA_02 3.556228e-06 6.015383e-04 0.012975779
##  8:                   kw_avg_min 2.939979e-06 5.713467e-04 0.014273356
##  9:                   kw_min_max 2.932538e-06 1.769230e-05 0.004757785
```

```
## 10:                   kw_min_avg 2.872393e-06 1.315510e-03 0.005622837
## 11:    global_sentiment_polarity 2.793402e-06 1.538325e-03 0.013408304
## 12:                       LDA_01 2.635373e-06 1.904006e-03 0.014705882
## 13:        min_positive_polarity 2.614589e-06 3.240167e-04 0.006487889
## 14:                       LDA_04 2.460281e-06 2.175610e-04 0.009948097
## 15:                   kw_max_avg 2.089647e-06 7.012311e-04 0.019031142
## 16:   global_rate_positive_words 2.040829e-06 1.700997e-04 0.007785467
## 17:          rate_positive_words 1.801766e-06 1.547624e-04 0.007785467
## 18:                       LDA_03 1.715815e-06 5.200390e-03 0.010813149
## 19:                 num_keywords 1.620872e-06 6.026252e-05 0.006920415
## 20:                   num_videos 1.315646e-06 5.138980e-03 0.009083045
```

```r
# Select the top N features from your dataset (replace N with the number you want)
selected_features <- dta2[, c(top_n_features$Feature, "shares")]

# Update the original dataset with the selected features
dta_xgb <- selected_features
```

```r
##Model 1: Liner Regression

# Select the top N features from your dataset (replace N with the number you want)
selected_features <- training_data[, top_n_features$Feature]

# Add the target variable to the selected features
selected_features$shares <- training_data$shares

# Train the linear regression model using the selected features
linear_model <- lm(shares ~ ., data = selected_features)

# Make predictions on the testing data
testing_data_subset <- testing_data[, top_n_features$Feature]
predictions <- predict(linear_model, newdata = testing_data_subset)

# Calculate Mean Absolute Error (MAE)
mae <- mean(abs(predictions - testing_data$shares))
cat("Linear Regression Mean Absolute Error (MAE):", mae, "\n")
```

```
## Linear Regression Mean Absolute Error (MAE): 3061.291
```

```r
# Calculate Mean Squared Error (MSE)
mse <- mean((predictions - testing_data$shares)^2)
cat("Linear Regression Mean Squared Error (MSE):", mse, "\n")
```

```
## Linear Regression Mean Squared Error (MSE): 101219715
```

```r
# Calculate R-squared (R2)
ssr <- sum((predictions - testing_data$shares)^2)
sst <- sum((testing_data$shares - mean(testing_data$shares))^2)
r_squared <- 1 - (ssr / sst)
cat("Linear Regression R-squared (R2):", r_squared, "\n")
```

```
## Linear Regression R-squared (R2): 0.01080917
```

```r
#Again the value for R square is very low.
```

```r
##Model 2:Decision Tree
# Load the required libraries

library(rpart)
set.seed(123)

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

index <- createDataPartition(dta_xgb$shares, p = 0.7, list = FALSE)
training_data <- dta_xgb[index, ]
testing_data <- dta_xgb[-index, ]

# Define the formula for the decision tree model
formula <- shares ~ .

# Train the decision tree model
tree_model <- rpart(formula, data = training_data, method = "anova")

# Make predictions on the testing data
predictions <- predict(tree_model, newdata = testing_data)

# Calculate Mean Absolute Error (MAE) for decision tree
tree_mae <- mean(abs(predictions - testing_data$shares))
cat("Decision Tree Mean Absolute Error (MAE):", tree_mae, "\n")
```

```
## Decision Tree Mean Absolute Error (MAE): 882.4869
```

```r
# Calculate Mean Squared Error (MSE) for decision tree
tree_mse <- mean((predictions - testing_data$shares)^2)
cat("Decision Tree Mean Squared Error (MSE):", tree_mse, "\n")
```

```
## Decision Tree Mean Squared Error (MSE): 9787805
```

```r
# Calculate R-squared (R2) for decision tree
tree_r_squared <- 1 - (sum((predictions - testing_data$shares)^2) / sum((testing_data$shares - mean(tes
cat("Decision Tree R-squared (R2):", tree_r_squared, "\n")
```

```
## Decision Tree R-squared (R2): 0.9114411
```

```r
# Create a confusion matrix
confusion_matrix <- table(Actual = testing_data$shares, Predicted = predictions)


# Calculate sensitivity and specificity
TP <- confusion_matrix[2, 2]  # True Positives
TN <- confusion_matrix[1, 1]  # True Negatives
FP <- confusion_matrix[1, 2]  # False Positives
FN <- confusion_matrix[2, 1]  # False Negatives
```

```r
# Sensitivity (True Positive Rate)
sensitivity <- TP / (TP + FN)

# Specificity (True Negative Rate)
specificity <- TN / (TN + FP)

# Accuracy
accuracy <- (TP + TN) / (TP + TN + FP + FN)

cat("Sensitivity (True Positive Rate):", sensitivity, "\n")
```

```
## Sensitivity (True Positive Rate): 0
```

```r
cat("Specificity (True Negative Rate):", specificity, "\n")
```

```
## Specificity (True Negative Rate): 1
```

```r
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.5
```

```r
#It is still giving an accuracy of 0.5 so this model is not accurate at all because it is making random
```

```r
## Model 3: Logistic Regression

# Define a threshold for popularity (e.g., more than 1400 shares)
threshold <- 1400

# Create a binary outcome variable 'popular' based on the threshold
dta_xgb$popular <- ifelse(dta_xgb$shares > threshold, 1, 0)
dta_xgb1 <- data <- subset(dta_xgb, select = -shares)

# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta_xgb1$popular, p = 0.7, list = FALSE)
training_data <- dta_xgb1[index, ]
testing_data <- dta_xgb1[-index, ]

# Train the logistic regression model with the selected features
logistic_model <- glm(popular ~ ., data = training_data, family = gaussian)

# Make predictions on the testing data
predictions <- predict(logistic_model, newdata = testing_data, type = "response")

# Convert predicted probabilities to binary outcomes (e.g., popular or not popular)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Calculate accuracy
accuracy <- mean(predicted_classes == testing_data$popular)
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.6401244
```

```r
#The accuracy level has improved a little with with 20 variables and now it has become almost 65%.
```

```r
## Model 4: Random Forest with XGBoost Dataset for Binary Classification

# Load the required library
library(randomForest)

# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta_xgb$shares, p = 0.7, list = FALSE)
training_data <- dta_xgb[index, ]
testing_data <- dta_xgb[-index, ]

# Train the Random Forest model for binary classification
rf_model <- randomForest(shares ~ ., data = training_data, ntree = 100)

# Make predictions on the testing data
predictions <- predict(rf_model, newdata = testing_data, type = "response")

# Convert predicted probabilities to binary outcomes (e.g., popular or not popular)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Calculate accuracy for binary classification
accuracy <- mean(predicted_classes == testing_data$popular)
cat("Random Forest Accuracy:", accuracy, "\n")
```

```
## Random Forest Accuracy: 0.493441
```

```r
# We got an accuracy level of 49% which means this model is not workable.
```

```r
## Model 5: Naive Bayes for Benchmarking

#Naive Bayes accuracy with 54 variables.

# Load the required library
library(e1071)

# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta_xgb$shares, p = 0.7, list = FALSE)
training_data <- dta_xgb[index, ]
testing_data <- dta_xgb[-index, ]

# Train the Naive Bayes model
nb_model <- naiveBayes(popular ~ ., data = training_data)

# Make predictions on the testing data
predictions <- predict(nb_model, newdata = testing_data, type = "class")

# Calculate accuracy for binary classification
accuracy <- mean(predictions == testing_data$popular)
cat("Naive Bayes Accuracy:", accuracy, "\n")
```

```
## Naive Bayes Accuracy: 0.860999
```

```r
## Model 5: Naive Bayes for Benchmarking

#Naive Bayes accuracy with 20 variables.

# Load the required library
library(e1071)

# Split the data into training (70%) and testing (30%) sets
index <- createDataPartition(dta_xgb$shares, p = 0.7, list = FALSE)
training_data <- dta2[index, ]
testing_data <- dta2[-index, ]

# Train the Naive Bayes model
nb_model <- naiveBayes(popular ~ ., data = training_data)

# Make predictions on the testing data
predictions <- predict(nb_model, newdata = testing_data, type = "class")

# Calculate accuracy for binary classification
accuracy <- mean(predictions == testing_data$popular)
cat("Naive Bayes Accuracy:", accuracy, "\n")
```

```
## Naive Bayes Accuracy: 0.8307265
```

```r
model_comparison_1 <- data.frame(
  Model = c("Naive Bayes", "Logistic Reg", "Random Forest", "Decision Tree"),
  Accuracy = c(0.8595, 0.6318, 0.4934, 0.5)
)

model_comparison_1
```

```
##           Model Accuracy
## 1   Naive Bayes   0.8595
## 2  Logistic Reg   0.6318
## 3 Random Forest   0.4934
## 4 Decision Tree   0.5000
```

```r
model_comparison_2 <- data.frame(
  Model = c("Naive Bayes", "Logistic Reg", "Random Forest", "Decision Tree"),
  Accuracy = c(0.8324, 0.6497, 0.4934, 0.5)
)

model_comparison_2
```

```
##           Model Accuracy
## 1   Naive Bayes   0.8324
## 2  Logistic Reg   0.6497
## 3 Random Forest   0.4934
## 4 Decision Tree   0.5000
```