Natural Language Processing (NLP)

Sasha Ajay Malkani

2025-06-28

## Importing the dataset

dataset\_original = read.delim("Restaurant\_Reviews.tsv", quote = "", stringsAsFactors = FALSE)  
head(dataset\_original)

## Review  
## 1 Wow... Loved this place.  
## 2 Crust is not good.  
## 3 Not tasty and the texture was just nasty.  
## 4 Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.  
## 5 The selection on the menu was great and so were the prices.  
## 6 Now I am getting angry and I want my damn pho.  
## Liked  
## 1 1  
## 2 0  
## 3 0  
## 4 1  
## 5 1  
## 6 0

## Cleaning the texts

# install.packages('tm')  
# install.packages('SnowballC')  
library(tm)

## Loading required package: NLP

library(SnowballC)  
corpus = VCorpus(VectorSource(dataset\_original$Review))  
corpus = tm\_map(corpus, content\_transformer(tolower))  
corpus = tm\_map(corpus, removeNumbers)  
corpus = tm\_map(corpus, removePunctuation)  
corpus = tm\_map(corpus, removeWords, stopwords())  
corpus = tm\_map(corpus, stemDocument)  
corpus = tm\_map(corpus, stripWhitespace)

## Creating the Bag of Words model

dtm = DocumentTermMatrix(corpus)  
dtm = removeSparseTerms(dtm, 0.999)  
dataset = as.data.frame(as.matrix(dtm))  
dataset$Liked = dataset\_original$Liked

## Encoding the target feature as factor

dataset$Liked = factor(dataset$Liked, levels = c(0, 1))

## Splitting the dataset into the Training set and Test set

# install.packages('caTools')  
library(caTools)  
set.seed(123)  
split = sample.split(dataset$Liked, SplitRatio = 0.80)  
training\_set = subset(dataset, split == TRUE)  
test\_set = subset(dataset, split == FALSE)

## Fitting Random Forest classification to the Training set

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

## randomForest 4.7-1.2

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

classifier = randomForest(x = training\_set[-692],   
 y = training\_set$Liked,   
 ntree = 10)

## Predicting the Test set results

y\_pred = predict(classifier, newdata = test\_set[-692])  
y\_pred

## 4 9 10 16 17 21 24 33 39 40 41 48 56 58 59 61 63 73 76 82   
## 1 1 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0   
## 92 93 98 99 105 112 113 115 116 122 123 142 150 152 154 157 158 159 161 169   
## 1 0 0 1 0 0 1 1 0 0 1 1 0 1 0 1 1 1 0 0   
## 182 183 184 188 190 191 193 199 202 203 210 211 217 222 228 239 240 250 251 255   
## 0 0 0 0 1 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1   
## 258 262 264 270 272 276 287 292 303 306 314 318 326 328 337 344 345 346 349 351   
## 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0   
## 353 361 363 364 370 375 395 396 397 399 412 413 415 416 430 433 445 446 453 456   
## 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 1 1 1 0   
## 466 469 470 473 486 495 496 509 519 521 525 528 531 535 539 545 548 555 560 563   
## 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1   
## 568 570 574 583 586 591 598 606 613 614 618 625 628 633 634 639 641 647 648 653   
## 1 0 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1 0 1 1   
## 658 668 674 679 688 694 698 712 715 716 719 730 739 743 752 759 761 768 780 789   
## 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1   
## 795 807 809 811 817 818 821 844 848 849 853 855 863 868 874 882 890 891 892 894   
## 0 0 1 1 0 1 0 0 1 0 1 1 0 0 1 0 1 1 1 0   
## 900 905 906 912 915 920 924 931 935 938 939 941 953 956 965 973 977 983 985 996   
## 1 0 1 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0   
## Levels: 0 1

## Making the Confusion Matrix

cm = table(test\_set[, 692], y\_pred)  
cm

## y\_pred  
## 0 1  
## 0 82 18  
## 1 23 77

## Performance Metrics

accuracy <- sum(diag(cm)) / sum(cm)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.795"

precision <- cm[2, 2] / sum(cm[2, ])  
print(paste("Precision:", precision))

## [1] "Precision: 0.77"

recall <- cm[2, 2] / sum(cm[, 2])  
print(paste("Recall:", recall))

## [1] "Recall: 0.810526315789474"

f1\_score <- 2 \* (precision \* recall) / (precision + recall)  
print(paste("F1 Score:", f1\_score))

## [1] "F1 Score: 0.78974358974359"

## Fitting Logistic Regression to the Training set

classifier = glm(formula = Liked ~ .,  
 family = binomial,  
 data = training\_set)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Predicting the Test set results

prob\_pred = predict(classifier, type = 'response', newdata = test\_set[-692])

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :  
## prediction from rank-deficient fit; attr(\*, "non-estim") has doubtful cases

print(prob\_pred)

## 4 9 10 16 17 21   
## 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 2.220446e-16 3.333333e-01   
## 24 33 39 40 41 48   
## 2.220446e-16 2.220446e-16 1.066921e-07 5.100875e-11 2.220446e-16 1.000000e+00   
## 56 58 59 61 63 73   
## 2.220446e-16 1.000000e+00 1.000000e+00 2.220446e-16 2.220446e-16 1.000000e+00   
## 76 82 92 93 98 99   
## 2.220446e-16 1.000000e+00 2.220446e-16 9.999809e-01 2.220446e-16 1.000000e+00   
## 105 112 113 115 116 122   
## 2.220446e-16 2.220446e-16 1.000000e+00 1.000000e+00 2.220446e-16 2.220446e-16   
## 123 142 150 152 154 157   
## 2.220446e-16 1.000000e+00 2.220446e-16 1.999616e-10 2.220446e-16 1.000000e+00   
## 158 159 161 169 182 183   
## 1.396504e-01 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 1.000000e+00   
## 184 188 190 191 193 199   
## 2.220446e-16 1.000000e+00 2.220446e-16 1.000000e+00 2.220446e-16 1.000000e+00   
## 202 203 210 211 217 222   
## 1.000000e+00 2.220446e-16 2.220446e-16 2.220446e-16 1.000000e+00 2.220446e-16   
## 228 239 240 250 251 255   
## 1.000000e+00 1.000000e+00 2.220446e-16 2.220446e-16 1.000000e+00 1.000000e+00   
## 258 262 264 270 272 276   
## 1.803923e-01 9.845554e-01 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00   
## 287 292 303 306 314 318   
## 1.000000e+00 2.220446e-16 2.220446e-16 1.000000e+00 2.220446e-16 1.000000e+00   
## 326 328 337 344 345 346   
## 2.220446e-16 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 1.000000e+00   
## 349 351 353 361 363 364   
## 1.000000e+00 2.220446e-16 9.998531e-01 2.220446e-16 7.472076e-01 1.000000e+00   
## 370 375 395 396 397 399   
## 2.220446e-16 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16 1.000000e+00   
## 412 413 415 416 430 433   
## 1.000000e+00 1.000000e+00 2.220446e-16 2.220446e-16 9.992794e-01 1.000000e+00   
## 445 446 453 456 466 469   
## 1.000000e+00 2.220446e-16 1.000000e+00 2.220446e-16 1.000000e+00 1.000000e+00   
## 470 473 486 495 496 509   
## 2.220446e-16 2.220446e-16 2.220446e-16 1.000000e+00 3.333333e-01 2.220446e-16   
## 519 521 525 528 531 535   
## 1.000000e+00 2.220446e-16 6.674391e-08 2.220446e-16 1.000000e+00 2.220446e-16   
## 539 545 548 555 560 563   
## 8.781656e-12 1.000000e+00 1.000000e+00 1.000000e+00 9.999940e-01 6.384419e-01   
## 568 570 574 583 586 591   
## 1.000000e+00 1.000000e+00 1.828384e-01 1.000000e+00 1.000000e+00 2.220446e-16   
## 598 606 613 614 618 625   
## 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00   
## 628 633 634 639 641 647   
## 1.000000e+00 2.220446e-16 1.000000e+00 9.837730e-01 1.000000e+00 7.884924e-12   
## 648 653 658 668 674 679   
## 1.000000e+00 2.220446e-16 1.000000e+00 2.220446e-16 2.220446e-16 9.999999e-01   
## 688 694 698 712 715 716   
## 1.000000e+00 1.000000e+00 3.333333e-01 1.000000e+00 1.000000e+00 2.220446e-16   
## 719 730 739 743 752 759   
## 1.000000e+00 2.220446e-16 2.220446e-16 1.000000e+00 1.000000e+00 1.000000e+00   
## 761 768 780 789 795 807   
## 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 2.220446e-16   
## 809 811 817 818 821 844   
## 1.000000e+00 1.000000e+00 5.671977e-11 2.220446e-16 1.000000e+00 3.469673e-10   
## 848 849 853 855 863 868   
## 1.000000e+00 1.000000e+00 2.220446e-16 1.000000e+00 1.000000e+00 1.000000e+00   
## 874 882 890 891 892 894   
## 2.220446e-16 2.220446e-16 2.220446e-16 2.220446e-16 1.000000e+00 1.000000e+00   
## 900 905 906 912 915 920   
## 1.000000e+00 2.220446e-16 1.000000e+00 2.220446e-16 2.220446e-16 1.000000e+00   
## 924 931 935 938 939 941   
## 1.000000e+00 2.220446e-16 1.000000e+00 2.220446e-16 2.220446e-16 2.220446e-16   
## 953 956 965 973 977 983   
## 1.000000e+00 2.220446e-16 2.220446e-16 1.000000e+00 2.220446e-16 1.000000e+00   
## 985 996   
## 2.220446e-16 1.000000e+00   
## attr(,"non-estim")  
## 59 61 73 76 82 98 112 122 157 182 193 210 211 239 240 250 255 258 262 264   
## 15 16 18 19 20 23 26 30 36 41 47 51 52 56 57 58 60 61 62 63   
## 270 303 318 326 337 345 349 351 353 395 416 430 445 446 453 456 469 555 574 598   
## 64 69 72 73 75 77 79 80 81 87 94 95 97 98 99 100 102 118 123 127   
## 658 668 674 715 716 739 743 752 789 795 818 853 855 890 892 924 941   
## 141 142 143 149 150 153 154 155 160 161 166 171 172 177 179 187 192

y\_pred = ifelse(prob\_pred > 0.5, 1, 0)  
print(y\_pred)

## 4 9 10 16 17 21 24 33 39 40 41 48 56 58 59 61 63 73 76 82   
## 1 1 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1 0 1   
## 92 93 98 99 105 112 113 115 116 122 123 142 150 152 154 157 158 159 161 169   
## 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 1   
## 182 183 184 188 190 191 193 199 202 203 210 211 217 222 228 239 240 250 251 255   
## 0 1 0 1 0 1 0 1 1 0 0 0 1 0 1 1 0 0 1 1   
## 258 262 264 270 272 276 287 292 303 306 314 318 326 328 337 344 345 346 349 351   
## 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0   
## 353 361 363 364 370 375 395 396 397 399 412 413 415 416 430 433 445 446 453 456   
## 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0 1 0   
## 466 469 470 473 486 495 496 509 519 521 525 528 531 535 539 545 548 555 560 563   
## 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 1 1 1   
## 568 570 574 583 586 591 598 606 613 614 618 625 628 633 634 639 641 647 648 653   
## 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0   
## 658 668 674 679 688 694 698 712 715 716 719 730 739 743 752 759 761 768 780 789   
## 1 0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1   
## 795 807 809 811 817 818 821 844 848 849 853 855 863 868 874 882 890 891 892 894   
## 1 0 1 1 0 0 1 0 1 1 0 1 1 1 0 0 0 0 1 1   
## 900 905 906 912 915 920 924 931 935 938 939 941 953 956 965 973 977 983 985 996   
## 1 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 0 1 0 1

## Making the Confusion Matrix

cm = table(test\_set[, 692], y\_pred)  
cm

## y\_pred  
## 0 1  
## 0 49 51  
## 1 39 61

## Performance Metrics

accuracy <- sum(diag(cm)) / sum(cm)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.55"

precision <- cm[2, 2] / sum(cm[2, ])  
print(paste("Precision:", precision))

## [1] "Precision: 0.61"

recall <- cm[2, 2] / sum(cm[, 2])  
print(paste("Recall:", recall))

## [1] "Recall: 0.544642857142857"

f1\_score <- 2 \* (precision \* recall) / (precision + recall)  
print(paste("F1 Score:", f1\_score))

## [1] "F1 Score: 0.575471698113208"

## Fitting K - Nearest Neighbours (KNN) to the Training set

library(class)  
y\_pred = knn(train = training\_set[, -692],  
 test = test\_set[, -692],  
 cl = training\_set[, 692],  
 k = 5,  
 prob = TRUE)

## Making the Confusion Matrix

cm = table(test\_set[, 692], y\_pred)  
cm

## y\_pred  
## 0 1  
## 0 75 25  
## 1 46 54

## Performance Metrics

accuracy <- sum(diag(cm)) / sum(cm)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.645"

precision <- cm[2, 2] / sum(cm[2, ])  
print(paste("Precision:", precision))

## [1] "Precision: 0.54"

recall <- cm[2, 2] / sum(cm[, 2])  
print(paste("Recall:", recall))

## [1] "Recall: 0.683544303797468"

f1\_score <- 2 \* (precision \* recall) / (precision + recall)  
print(paste("F1 Score:", f1\_score))

## [1] "F1 Score: 0.603351955307262"

## Fitting Linear Support Vector Machines (SVM) to the Training set

# install.packages('e1071')  
library(e1071)  
  
classifier = svm(formula = Liked ~ .,  
 data = training\_set,  
 type = 'C-classification',  
 kernel = 'linear')

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):  
## Variable(s) 'boot' and 'brick' and 'eye' and 'given' and 'legit' and 'mall' and  
## 'oven' and 'peanut' and 'pure' and 'scallop' and 'show' and 'tap' constant.  
## Cannot scale data.

## Predicting the Test set results

y\_pred = predict(classifier, newdata = test\_set[-692])  
y\_pred

## 4 9 10 16 17 21 24 33 39 40 41 48 56 58 59 61 63 73 76 82   
## 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0   
## 92 93 98 99 105 112 113 115 116 122 123 142 150 152 154 157 158 159 161 169   
## 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1 0   
## 182 183 184 188 190 191 193 199 202 203 210 211 217 222 228 239 240 250 251 255   
## 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 1 1   
## 258 262 264 270 272 276 287 292 303 306 314 318 326 328 337 344 345 346 349 351   
## 0 0 1 0 1 0 1 0 0 0 0 0 0 1 1 1 0 1 1 0   
## 353 361 363 364 370 375 395 396 397 399 412 413 415 416 430 433 445 446 453 456   
## 1 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 1   
## 466 469 470 473 486 495 496 509 519 521 525 528 531 535 539 545 548 555 560 563   
## 1 1 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1   
## 568 570 574 583 586 591 598 606 613 614 618 625 628 633 634 639 641 647 648 653   
## 1 0 1 0 1 1 1 1 1 0 0 1 1 0 1 0 1 0 0 1   
## 658 668 674 679 688 694 698 712 715 716 719 730 739 743 752 759 761 768 780 789   
## 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1   
## 795 807 809 811 817 818 821 844 848 849 853 855 863 868 874 882 890 891 892 894   
## 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 0 1 1 1 0   
## 900 905 906 912 915 920 924 931 935 938 939 941 953 956 965 973 977 983 985 996   
## 1 1 1 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0   
## Levels: 0 1

## Making the Confusion Matrix

cm = table(test\_set[, 692], y\_pred)  
cm

## y\_pred  
## 0 1  
## 0 78 22  
## 1 19 81

## Performance Metrics

accuracy <- sum(diag(cm)) / sum(cm)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.795"

precision <- cm[2, 2] / sum(cm[2, ])  
print(paste("Precision:", precision))

## [1] "Precision: 0.81"

recall <- cm[2, 2] / sum(cm[, 2])  
print(paste("Recall:", recall))

## [1] "Recall: 0.786407766990291"

f1\_score <- 2 \* (precision \* recall) / (precision + recall)  
print(paste("F1 Score:", f1\_score))

## [1] "F1 Score: 0.798029556650246"

## Fitting Kernel Support Vector Machines (SVM) to the Training set

# install.packages("e1071")  
library(e1071)  
  
classifier = svm(formula = Liked ~ .,  
 data = training\_set,  
 type = 'C-classification',  
 kernel = 'radial')

## Warning in svm.default(x, y, scale = scale, ..., na.action = na.action):  
## Variable(s) 'boot' and 'brick' and 'eye' and 'given' and 'legit' and 'mall' and  
## 'oven' and 'peanut' and 'pure' and 'scallop' and 'show' and 'tap' constant.  
## Cannot scale data.

## Predicting the Test set results

y\_pred = predict(classifier, newdata = test\_set[-692])  
y\_pred

## 4 9 10 16 17 21 24 33 39 40 41 48 56 58 59 61 63 73 76 82   
## 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0   
## 92 93 98 99 105 112 113 115 116 122 123 142 150 152 154 157 158 159 161 169   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 182 183 184 188 190 191 193 199 202 203 210 211 217 222 228 239 240 250 251 255   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 258 262 264 270 272 276 287 292 303 306 314 318 326 328 337 344 345 346 349 351   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 353 361 363 364 370 375 395 396 397 399 412 413 415 416 430 433 445 446 453 456   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 466 469 470 473 486 495 496 509 519 521 525 528 531 535 539 545 548 555 560 563   
## 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 568 570 574 583 586 591 598 606 613 614 618 625 628 633 634 639 641 647 648 653   
## 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 658 668 674 679 688 694 698 712 715 716 719 730 739 743 752 759 761 768 780 789   
## 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0   
## 795 807 809 811 817 818 821 844 848 849 853 855 863 868 874 882 890 891 892 894   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 900 905 906 912 915 920 924 931 935 938 939 941 953 956 965 973 977 983 985 996   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## Levels: 0 1

## Making the Confusion Matrix

cm = table(test\_set[, 692], y\_pred)  
cm

## y\_pred  
## 0 1  
## 0 100 0  
## 1 94 6

## Performance Metrics

accuracy <- sum(diag(cm)) / sum(cm)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.53"

precision <- cm[2, 2] / sum(cm[2, ])  
print(paste("Precision:", precision))

## [1] "Precision: 0.06"

recall <- cm[2, 2] / sum(cm[, 2])  
print(paste("Recall:", recall))

## [1] "Recall: 1"

f1\_score <- 2 \* (precision \* recall) / (precision + recall)  
print(paste("F1 Score:", f1\_score))

## [1] "F1 Score: 0.113207547169811"

## Fitting Naive Bayes Model to the Training set

# install.packages("e1071")  
library(e1071)  
  
classifier = naiveBayes(x = training\_set[,-692],  
 y = training\_set$Liked)

## Predicting the Test set results

y\_pred = predict(classifier, newdata = test\_set[-692])  
y\_pred

## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1  
## [38] 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1  
## [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1  
## [149] 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1  
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## Levels: 0 1

## Making the Confusion Matrix

cm = table(test\_set[, 692], y\_pred)  
cm

## y\_pred  
## 0 1  
## 0 5 95  
## 1 4 96

## Performance Metrics

accuracy <- sum(diag(cm)) / sum(cm)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.505"

precision <- cm[2, 2] / sum(cm[2, ])  
print(paste("Precision:", precision))

## [1] "Precision: 0.96"

recall <- cm[2, 2] / sum(cm[, 2])  
print(paste("Recall:", recall))

## [1] "Recall: 0.50261780104712"

f1\_score <- 2 \* (precision \* recall) / (precision + recall)  
print(paste("F1 Score:", f1\_score))

## [1] "F1 Score: 0.65979381443299"

## Fitting Decision Tree Model to the Training set

# install.packages('rpart')  
library(rpart)  
classifier = rpart(formula = Liked ~ .,  
 data = training\_set)

## Predicting the Test set results

y\_pred = predict(classifier, newdata = test\_set[-692], type = 'class')  
y\_pred

## 4 9 10 16 17 21 24 33 39 40 41 48 56 58 59 61 63 73 76 82   
## 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0   
## 92 93 98 99 105 112 113 115 116 122 123 142 150 152 154 157 158 159 161 169   
## 0 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 1 0 0   
## 182 183 184 188 190 191 193 199 202 203 210 211 217 222 228 239 240 250 251 255   
## 0 0 0 0 1 1 0 1 1 0 0 0 1 0 1 0 0 0 1 1   
## 258 262 264 270 272 276 287 292 303 306 314 318 326 328 337 344 345 346 349 351   
## 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0   
## 353 361 363 364 370 375 395 396 397 399 412 413 415 416 430 433 445 446 453 456   
## 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0   
## 466 469 470 473 486 495 496 509 519 521 525 528 531 535 539 545 548 555 560 563   
## 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1   
## 568 570 574 583 586 591 598 606 613 614 618 625 628 633 634 639 641 647 648 653   
## 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 1 0 1 1   
## 658 668 674 679 688 694 698 712 715 716 719 730 739 743 752 759 761 768 780 789   
## 1 0 1 1 1 0 0 0 1 1 0 1 1 1 0 1 1 1 1 1   
## 795 807 809 811 817 818 821 844 848 849 853 855 863 868 874 882 890 891 892 894   
## 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0   
## 900 905 906 912 915 920 924 931 935 938 939 941 953 956 965 973 977 983 985 996   
## 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0   
## Levels: 0 1

## Making the Confusion Matrix

cm = table(test\_set[, 692], y\_pred)  
cm

## y\_pred  
## 0 1  
## 0 85 15  
## 1 43 57

## Performance Metrics

accuracy <- sum(diag(cm)) / sum(cm)  
print(paste("Accuracy:", accuracy))

## [1] "Accuracy: 0.71"

precision <- cm[2, 2] / sum(cm[2, ])  
print(paste("Precision:", precision))

## [1] "Precision: 0.57"

recall <- cm[2, 2] / sum(cm[, 2])  
print(paste("Recall:", recall))

## [1] "Recall: 0.791666666666667"

f1\_score <- 2 \* (precision \* recall) / (precision + recall)  
print(paste("F1 Score:", f1\_score))

## [1] "F1 Score: 0.662790697674419"

Note that the echo = FALSE parameter was added to the code chunk to prevent printing of the R code that generated the plot.