Math 156 - Fnial

Wentao Deng, Yifan Jiang, Qize Zhang

Spring 2024

```
library(reticulate)
library(tm)
```

```
##      NLP
```

```
library(text2vec)
library(caret)
```

```
##      ggplot2
```

```
##
##     'ggplot2'
```

```
## The following object is masked from 'package:NLP':
##
##     annotate
```

```
##      lattice
```

```
library(e1071)
library(keras)
```

```
##
##     'keras'
```

```
## The following objects are masked from 'package:text2vec':
##
##     fit, normalize
```

```
library(naivebayes)
```

```
## naivebayes 1.0.0 loaded
```

```
## For more information please visit:
```

```
## https://majkamichal.github.io/naivebayes/
```

```r
library(stringi)
library(dplyr)
```

```
##
##     'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --

## v tibble  3.2.1     v purrr   1.0.1
## v tidyr   1.2.1     v stringr 1.4.1
## v readr   2.1.3     v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x ggplot2::annotate() masks NLP::annotate()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x purrr::lift()       masks caret::lift()
```

```r
library(SnowballC)
library(nnet)
```

```r
# Define the path to the dataset
path <- "D:/my_document/UCLA/2024-spring/24S-MATH-156-LEC-1/project/bbc"

# Get the list of folders
folders <- list.files(path, full.names = TRUE)

# Read the text files and create a dataframe
data <- do.call(rbind, lapply(folders, function(folder) {
  file_paths <- list.files(folder, full.names = TRUE)
  if (length(file_paths) == 0) {
    return(NULL) # Skip empty folders
  }
  text <- sapply(file_paths, function(file_path) {
    # Read the entire file and collapse it into one single string
    paste(readLines(file_path, warn = FALSE), collapse = " ")
  })
  if (length(text) == 0) {
    return(NULL) # Skip if no text is read
  }
  data.frame(text = text, category = basename(folder), stringsAsFactors = FALSE)
}))
```

```r
# Check if data is created successfully
if (is.null(data) || nrow(data) == 0) {
  stop("No data was read from the files. Please check the file paths and contents.")
} else {
  print("Dataframe created successfully.")
}
```

```
## [1] "Dataframe created successfully."
```

```r
data %>%
  group_by(category) %>%
  summarise(count = n())
```

```
## # A tibble: 5 x 2
##   category      count
##   <chr>         <int>
## 1 business        510
## 2 entertainment   386
## 3 politics        417
## 4 sport           511
## 5 tech            401
```

```r
# Data Preprocessing
# Create a text corpus
corpus <- VCorpus(VectorSource(data$text))
# Preprocess the corpus
corpus <- tm_map(corpus, content_transformer(function(x) iconv(x, from = "UTF-8", to = "ASCII", sub = "
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, removeWords, stopwords("en"))
corpus <- tm_map(corpus, stripWhitespace)
# Feature Extraction: Create a document-term matrix using TF-IDF
dtm <- DocumentTermMatrix(corpus)
dtm2 <- removeSparseTerms(dtm, 0.99)
tfidf <- weightTfIdf(dtm2)
```

```r
set.seed(123)
# Prepare training and test datasets
full_data_matrix <- as.matrix(tfidf)
set <- createDataPartition(data$category, p = 0.8, list = FALSE)
train_data <- full_data_matrix[set, ]
test_data <- full_data_matrix[-set, ]
train_labels <- as.factor(data$category[set])
test_labels <- as.factor(data$category[-set])
```

```r
# Naive Bayes Model
model_nb <- naive_bayes(x = as.data.frame(train_data), y = train_labels)
# Predict using the trained model
predictions_nb <- predict(model_nb, as.data.frame(test_data))
# Compute the confusion matrix
```

```r
confMat_nb <- confusionMatrix(predictions_nb, test_labels)
print(confMat_nb)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      business entertainment politics sport tech
##   business            94             4        2     1    2
##   entertainment        1            62        0     0    2
##   politics             4             2       78     0    2
##   sport                0             6        2   100    0
##   tech                 3             3        1     1   74
##
## Overall Statistics
##
##                Accuracy : 0.9189
##                  95% CI : (0.8895, 0.9426)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8981
##
##  Mcnemar's Test P-Value : 0.1249
##
## Statistics by Class:
##
##                      Class: business Class: entertainment Class: politics
## Sensitivity                   0.9216               0.8052          0.9398
## Specificity                   0.9737               0.9918          0.9778
## Pos Pred Value                0.9126               0.9538          0.9070
## Neg Pred Value                0.9765               0.9604          0.9860
## Prevalence                    0.2297               0.1734          0.1869
## Detection Rate                0.2117               0.1396          0.1757
## Detection Prevalence          0.2320               0.1464          0.1937
## Balanced Accuracy             0.9476               0.8985          0.9588
##                      Class: sport Class: tech
## Sensitivity                0.9804      0.9250
## Specificity                0.9766      0.9780
## Pos Pred Value             0.9259      0.9024
## Neg Pred Value             0.9940      0.9834
## Prevalence                 0.2297      0.1802
## Detection Rate             0.2252      0.1667
## Detection Prevalence       0.2432      0.1847
## Balanced Accuracy          0.9785      0.9515
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

```
##
##     'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine

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

```
rf_model <- randomForest(x = train_data, y = train_labels, ntree = 100)
```

```
# Evaluate the model
predictions <- predict(rf_model, newdata = test_data)
confusionMatrix(predictions, test_labels)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      business entertainment politics sport tech
##   business            98             4        2     0    1
##   entertainment        0            69        0     0    1
##   politics             2             1       77     0    1
##   sport                2             1        2   102    2
##   tech                 0             2        2     0   75
##
## Overall Statistics
##
##                Accuracy : 0.9482
##                  95% CI : (0.9233, 0.9669)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9349
##
##  Mcnemar's Test P-Value : 0.1887
##
## Statistics by Class:
##
##                      Class: business Class: entertainment Class: politics
## Sensitivity                   0.9608               0.8961          0.9277
## Specificity                   0.9795               0.9973          0.9889
## Pos Pred Value                0.9333               0.9857          0.9506
## Neg Pred Value                0.9882               0.9786          0.9835
## Prevalence                    0.2297               0.1734          0.1869
## Detection Rate                0.2207               0.1554          0.1734
## Detection Prevalence          0.2365               0.1577          0.1824
## Balanced Accuracy             0.9702               0.9467          0.9583
##                      Class: sport Class: tech
## Sensitivity                1.0000      0.9375
## Specificity                0.9795      0.9890
## Pos Pred Value             0.9358      0.9494
## Neg Pred Value             1.0000      0.9863
## Prevalence                 0.2297      0.1802
## Detection Rate             0.2297      0.1689
```

```
## Detection Prevalence       0.2455      0.1779
## Balanced Accuracy          0.9898      0.9633
```

```r
library(class)
# Train KNN model
knn_model <- knn(train = train_data, test = test_data, cl = train_labels, k = 5)

# Evaluate the model
conf_matrix <- confusionMatrix(knn_model, test_labels)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      business entertainment politics sport tech
##   business            66             0        0     0    0
##   entertainment        1            36        0     0    0
##   politics            26            29       79    21    4
##   sport                2             0        1    71    0
##   tech                 7            12        3    10   76
##
## Overall Statistics
##
##                Accuracy : 0.7387
##                  95% CI : (0.6952, 0.779)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6744
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: business Class: entertainment Class: politics
## Sensitivity                   0.6471              0.46753          0.9518
## Specificity                   1.0000              0.99728          0.7784
## Pos Pred Value                1.0000              0.97297          0.4969
## Neg Pred Value                0.9048              0.89926          0.9860
## Prevalence                    0.2297              0.17342          0.1869
## Detection Rate                0.1486              0.08108          0.1779
## Detection Prevalence          0.1486              0.08333          0.3581
## Balanced Accuracy             0.8235              0.73240          0.8651
##                      Class: sport Class: tech
## Sensitivity                0.6961      0.9500
## Specificity                0.9912      0.9121
## Pos Pred Value             0.9595      0.7037
## Neg Pred Value             0.9162      0.9881
## Prevalence                 0.2297      0.1802
## Detection Rate             0.1599      0.1712
## Detection Prevalence       0.1667      0.2432
## Balanced Accuracy          0.8437      0.9310
```

```r
library(rpart)
# Train Decision Tree model
tree_model <- rpart(train_labels ~ ., data = as.data.frame(train_data), method = "class")

# Predict using the Decision Tree model
tree_predictions <- predict(tree_model, newdata = as.data.frame(test_data), type = "class")

# Evaluate the Decision Tree model
tree_conf_matrix <- confusionMatrix(tree_predictions, test_labels)
print("Decision Tree Confusion Matrix:")
```

```
## [1] "Decision Tree Confusion Matrix:"
```

```r
print(tree_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      business entertainment politics sport tech
##   business            80             8        6     2   10
##   entertainment        2            54        3     2   11
##   politics             4             0       57     0    0
##   sport               11            15       14    96    9
##   tech                 5             0        3     2   50
##
## Overall Statistics
##
##                Accuracy : 0.759
##                  95% CI : (0.7165, 0.7981)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6957
##
##  Mcnemar's Test P-Value : 1.174e-08
##
## Statistics by Class:
##
##                      Class: business Class: entertainment Class: politics
## Sensitivity                   0.7843               0.7013          0.6867
## Specificity                   0.9240               0.9510          0.9889
## Pos Pred Value                0.7547               0.7500          0.9344
## Neg Pred Value                0.9349               0.9382          0.9321
## Prevalence                    0.2297               0.1734          0.1869
## Detection Rate                0.1802               0.1216          0.1284
## Detection Prevalence          0.2387               0.1622          0.1374
## Balanced Accuracy             0.8541               0.8261          0.8378
##                      Class: sport Class: tech
## Sensitivity                0.9412      0.6250
## Specificity                0.8567      0.9725
## Pos Pred Value             0.6621      0.8333
## Neg Pred Value             0.9799      0.9219
```

```
## Prevalence                0.2297      0.1802
## Detection Rate            0.2162      0.1126
## Detection Prevalence      0.3266      0.1351
## Balanced Accuracy         0.8990      0.7988
```

```r
# Load necessary libraries for Logistic Regression
library(glmnet)
```

```
##      Matrix
```

```
##
##     'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

```r
# Train Logistic Regression model using glmnet
logistic_model <- cv.glmnet(train_data, train_labels, family = "multinomial", type.multinomial = "groupe

# Predict using the Logistic Regression model
logistic_predictions <- predict(logistic_model, newx = test_data, s = "lambda.min", type = "class")

# Evaluate the Logistic Regression model
logistic_conf_matrix <- confusionMatrix(as.factor(logistic_predictions), test_labels)
print("Logistic Regression Confusion Matrix:")
```

```
## [1] "Logistic Regression Confusion Matrix:"
```

```r
print(logistic_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      business entertainment politics sport tech
##   business            99             1        1     0    2
##   entertainment        0            75        0     0    0
##   politics             2             1       81     0    2
##   sport                0             0        1   102    0
##   tech                 1             0        0     0   76
##
## Overall Statistics
##
##                Accuracy : 0.9752
##                  95% CI : (0.9561, 0.9876)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9689
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: business Class: entertainment Class: politics
## Sensitivity                   0.9706               0.9740          0.9759
## Specificity                   0.9883               1.0000          0.9861
## Pos Pred Value                0.9612               1.0000          0.9419
## Neg Pred Value                0.9912               0.9946          0.9944
## Prevalence                    0.2297               0.1734          0.1869
## Detection Rate                0.2230               0.1689          0.1824
## Detection Prevalence          0.2320               0.1689          0.1937
## Balanced Accuracy             0.9794               0.9870          0.9810
##                      Class: sport Class: tech
## Sensitivity                1.0000      0.9500
## Specificity                0.9971      0.9973
## Pos Pred Value             0.9903      0.9870
## Neg Pred Value             1.0000      0.9891
## Prevalence                 0.2297      0.1802
## Detection Rate             0.2297      0.1712
## Detection Prevalence       0.2320      0.1734
## Balanced Accuracy          0.9985      0.9736
```

```r
# Load necessary libraries for Gradient Boosting with xgboost
library(xgboost)
```

```
##
##     'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
# Prepare data for xgboost
train_data_matrix <- xgb.DMatrix(data = train_data, label = as.numeric(train_labels) - 1)
test_data_matrix <- xgb.DMatrix(data = test_data, label = as.numeric(test_labels) - 1)

# Set parameters for xgboost
params <- list(
  booster = "gbtree",
  objective = "multi:softprob",
  num_class = length(unique(train_labels)),
  eval_metric = "mlogloss"
)

# Train the xgboost model
set.seed(123)
xgb_model <- xgb.train(
  params = params,
  data = train_data_matrix,
  nrounds = 100,
  watchlist = list(train = train_data_matrix, test = test_data_matrix),
```

```
  verbose = 1
)
```

```
## [1]  train-mlogloss:1.104094 test-mlogloss:1.159983
## [2]  train-mlogloss:0.823748 test-mlogloss:0.909442
## [3]  train-mlogloss:0.642182 test-mlogloss:0.752117
## [4]  train-mlogloss:0.507891 test-mlogloss:0.640970
## [5]  train-mlogloss:0.411420 test-mlogloss:0.555289
## [6]  train-mlogloss:0.336692 test-mlogloss:0.492087
## [7]  train-mlogloss:0.279658 test-mlogloss:0.444738
## [8]  train-mlogloss:0.233386 test-mlogloss:0.408442
## [9]  train-mlogloss:0.199233 test-mlogloss:0.376671
## [10] train-mlogloss:0.169523 test-mlogloss:0.354788
## [11] train-mlogloss:0.145254 test-mlogloss:0.330177
## [12] train-mlogloss:0.125711 test-mlogloss:0.310444
## [13] train-mlogloss:0.108867 test-mlogloss:0.294368
## [14] train-mlogloss:0.094432 test-mlogloss:0.279986
## [15] train-mlogloss:0.083372 test-mlogloss:0.268060
## [16] train-mlogloss:0.073728 test-mlogloss:0.259297
## [17] train-mlogloss:0.065250 test-mlogloss:0.248887
## [18] train-mlogloss:0.058432 test-mlogloss:0.240512
## [19] train-mlogloss:0.052057 test-mlogloss:0.234867
## [20] train-mlogloss:0.047070 test-mlogloss:0.227579
## [21] train-mlogloss:0.042462 test-mlogloss:0.222581
## [22] train-mlogloss:0.038146 test-mlogloss:0.217483
## [23] train-mlogloss:0.034449 test-mlogloss:0.213824
## [24] train-mlogloss:0.031380 test-mlogloss:0.209866
## [25] train-mlogloss:0.028609 test-mlogloss:0.205862
## [26] train-mlogloss:0.026240 test-mlogloss:0.202966
## [27] train-mlogloss:0.024065 test-mlogloss:0.200901
## [28] train-mlogloss:0.022109 test-mlogloss:0.197285
## [29] train-mlogloss:0.020459 test-mlogloss:0.196078
## [30] train-mlogloss:0.018877 test-mlogloss:0.194367
## [31] train-mlogloss:0.017543 test-mlogloss:0.192190
## [32] train-mlogloss:0.016434 test-mlogloss:0.189934
## [33] train-mlogloss:0.015312 test-mlogloss:0.189685
## [34] train-mlogloss:0.014374 test-mlogloss:0.189021
## [35] train-mlogloss:0.013425 test-mlogloss:0.186594
## [36] train-mlogloss:0.012679 test-mlogloss:0.186319
## [37] train-mlogloss:0.011961 test-mlogloss:0.185499
## [38] train-mlogloss:0.011336 test-mlogloss:0.184285
## [39] train-mlogloss:0.010737 test-mlogloss:0.183756
## [40] train-mlogloss:0.010217 test-mlogloss:0.184686
## [41] train-mlogloss:0.009754 test-mlogloss:0.182646
## [42] train-mlogloss:0.009347 test-mlogloss:0.182437
## [43] train-mlogloss:0.008990 test-mlogloss:0.181468
## [44] train-mlogloss:0.008655 test-mlogloss:0.181074
## [45] train-mlogloss:0.008342 test-mlogloss:0.180998
## [46] train-mlogloss:0.008053 test-mlogloss:0.180645
## [47] train-mlogloss:0.007772 test-mlogloss:0.180159
## [48] train-mlogloss:0.007513 test-mlogloss:0.179895
## [49] train-mlogloss:0.007296 test-mlogloss:0.179427
## [50] train-mlogloss:0.007055 test-mlogloss:0.178435
```

```
## [51] train-mlogloss:0.006864 test-mlogloss:0.177796
## [52] train-mlogloss:0.006683 test-mlogloss:0.177828
## [53] train-mlogloss:0.006510 test-mlogloss:0.177290
## [54] train-mlogloss:0.006344 test-mlogloss:0.177232
## [55] train-mlogloss:0.006181 test-mlogloss:0.176894
## [56] train-mlogloss:0.006054 test-mlogloss:0.176621
## [57] train-mlogloss:0.005920 test-mlogloss:0.177928
## [58] train-mlogloss:0.005796 test-mlogloss:0.177696
## [59] train-mlogloss:0.005689 test-mlogloss:0.178185
## [60] train-mlogloss:0.005581 test-mlogloss:0.178573
## [61] train-mlogloss:0.005485 test-mlogloss:0.178097
## [62] train-mlogloss:0.005396 test-mlogloss:0.177902
## [63] train-mlogloss:0.005304 test-mlogloss:0.177817
## [64] train-mlogloss:0.005229 test-mlogloss:0.178031
## [65] train-mlogloss:0.005145 test-mlogloss:0.177510
## [66] train-mlogloss:0.005074 test-mlogloss:0.177747
## [67] train-mlogloss:0.005011 test-mlogloss:0.177321
## [68] train-mlogloss:0.004941 test-mlogloss:0.177541
## [69] train-mlogloss:0.004874 test-mlogloss:0.178404
## [70] train-mlogloss:0.004818 test-mlogloss:0.178486
## [71] train-mlogloss:0.004757 test-mlogloss:0.178018
## [72] train-mlogloss:0.004703 test-mlogloss:0.177940
## [73] train-mlogloss:0.004650 test-mlogloss:0.178391
## [74] train-mlogloss:0.004605 test-mlogloss:0.178365
## [75] train-mlogloss:0.004559 test-mlogloss:0.178462
## [76] train-mlogloss:0.004519 test-mlogloss:0.178601
## [77] train-mlogloss:0.004479 test-mlogloss:0.178465
## [78] train-mlogloss:0.004440 test-mlogloss:0.178855
## [79] train-mlogloss:0.004405 test-mlogloss:0.178752
## [80] train-mlogloss:0.004365 test-mlogloss:0.179174
## [81] train-mlogloss:0.004327 test-mlogloss:0.179674
## [82] train-mlogloss:0.004292 test-mlogloss:0.179670
## [83] train-mlogloss:0.004252 test-mlogloss:0.179446
## [84] train-mlogloss:0.004228 test-mlogloss:0.179374
## [85] train-mlogloss:0.004203 test-mlogloss:0.179745
## [86] train-mlogloss:0.004176 test-mlogloss:0.179976
## [87] train-mlogloss:0.004145 test-mlogloss:0.179944
## [88] train-mlogloss:0.004117 test-mlogloss:0.180194
## [89] train-mlogloss:0.004090 test-mlogloss:0.180045
## [90] train-mlogloss:0.004065 test-mlogloss:0.180234
## [91] train-mlogloss:0.004044 test-mlogloss:0.180313
## [92] train-mlogloss:0.004016 test-mlogloss:0.180272
## [93] train-mlogloss:0.003990 test-mlogloss:0.180198
## [94] train-mlogloss:0.003968 test-mlogloss:0.180280
## [95] train-mlogloss:0.003949 test-mlogloss:0.180265
## [96] train-mlogloss:0.003927 test-mlogloss:0.180319
## [97] train-mlogloss:0.003907 test-mlogloss:0.180208
## [98] train-mlogloss:0.003888 test-mlogloss:0.180216
## [99] train-mlogloss:0.003870 test-mlogloss:0.180129
## [100]    train-mlogloss:0.003848 test-mlogloss:0.180251
```

```
# Predict using the xgboost model
xgb_predictions <- predict(xgb_model, newdata = test_data_matrix)
xgb_predictions_class <- max.col(matrix(xgb_predictions, ncol = length(unique(train_labels)), byrow = TR
```

```r
# Evaluate the xgboost model
xgb_conf_matrix <- confusionMatrix(as.factor(xgb_predictions_class), as.factor(as.numeric(test_labels)) +
print("Gradient Boosting (xgboost) Confusion Matrix:")
```

```
## [1] "Gradient Boosting (xgboost) Confusion Matrix:"
```

```r
print(xgb_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4
##          0  94   3   2   0   3
##          1   0  71   1   0   1
##          2   5   2  75   0   0
##          3   1   0   3 102   2
##          4   2   1   2   0  74
##
## Overall Statistics
##
##                Accuracy : 0.9369
##                  95% CI : (0.9101, 0.9577)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9208
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity            0.9216   0.9221   0.9036   1.0000   0.9250
## Specificity            0.9766   0.9946   0.9806   0.9825   0.9863
## Pos Pred Value         0.9216   0.9726   0.9146   0.9444   0.9367
## Neg Pred Value         0.9766   0.9838   0.9779   1.0000   0.9836
## Prevalence             0.2297   0.1734   0.1869   0.2297   0.1802
## Detection Rate         0.2117   0.1599   0.1689   0.2297   0.1667
## Detection Prevalence   0.2297   0.1644   0.1847   0.2432   0.1779
## Balanced Accuracy      0.9491   0.9583   0.9421   0.9912   0.9556
```

```r
# Load necessary libraries for LightGBM
library(lightgbm)
```

```
##
##     'lightgbm'
```

```
## The following object is masked from 'package:xgboost':
##
##     slice
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
# Prepare data for LightGBM
train_data_lgb <- lgb.Dataset(data = train_data, label = as.numeric(train_labels) - 1)
test_data_lgb <- lgb.Dataset(data = test_data, label = as.numeric(test_labels) - 1, free_raw_data = FALS

# Set parameters for LightGBM
params <- list(
  objective = "multiclass",
  num_class = length(unique(train_labels)),
  metric = "multi_logloss"
)

# Train the LightGBM model
set.seed(123)
lgb_model <- lgb.train(
  params = params,
  data = train_data_lgb,
  nrounds = 100,
  valids = list(test = test_data_lgb),
  verbose = 1
)
```

```
## [LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.150576 secon
## You can set `force_row_wise=true` to remove the overhead.
## And if memory is not enough, you can set `force_col_wise=true`.
## [LightGBM] [Info] Total Bins 61996
## [LightGBM] [Info] Number of data points in the train set: 1781, number of used features: 2679
## [LightGBM] [Info] Start training from score -1.473663
## [LightGBM] [Info] Start training from score -1.751589
## [LightGBM] [Info] Start training from score -1.673789
## [LightGBM] [Info] Start training from score -1.471215
## [LightGBM] [Info] Start training from score -1.713489
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [1]:  test's multi_logloss:1.33829
## [2]:  test's multi_logloss:1.15935
## [3]:  test's multi_logloss:1.01979
## [4]:  test's multi_logloss:0.907906
## [5]:  test's multi_logloss:0.821587
## [6]:  test's multi_logloss:0.744973
## [7]:  test's multi_logloss:0.68048
## [8]:  test's multi_logloss:0.625217
## [9]:  test's multi_logloss:0.578939
## [10]:  test's multi_logloss:0.535728
## [11]:  test's multi_logloss:0.500517
## [12]:  test's multi_logloss:0.46639
## [13]:  test's multi_logloss:0.437285
## [14]:  test's multi_logloss:0.410828
## [15]:  test's multi_logloss:0.387274
```

```
## [16]:  test's multi_logloss:0.365913
## [17]:  test's multi_logloss:0.346459
## [18]:  test's multi_logloss:0.326272
## [19]:  test's multi_logloss:0.310409
## [20]:  test's multi_logloss:0.296284
## [21]:  test's multi_logloss:0.283724
## [22]:  test's multi_logloss:0.271669
## [23]:  test's multi_logloss:0.260958
## [24]:  test's multi_logloss:0.250245
## [25]:  test's multi_logloss:0.239644
## [26]:  test's multi_logloss:0.230264
## [27]:  test's multi_logloss:0.222856
## [28]:  test's multi_logloss:0.215456
## [29]:  test's multi_logloss:0.209634
## [30]:  test's multi_logloss:0.203187
## [31]:  test's multi_logloss:0.19814
## [32]:  test's multi_logloss:0.19477
## [33]:  test's multi_logloss:0.190398
## [34]:  test's multi_logloss:0.185885
## [35]:  test's multi_logloss:0.182588
## [36]:  test's multi_logloss:0.177791
## [37]:  test's multi_logloss:0.172963
## [38]:  test's multi_logloss:0.17027
## [39]:  test's multi_logloss:0.16608
## [40]:  test's multi_logloss:0.163439
## [41]:  test's multi_logloss:0.159829
## [42]:  test's multi_logloss:0.157721
## [43]:  test's multi_logloss:0.155107
## [44]:  test's multi_logloss:0.153784
## [45]:  test's multi_logloss:0.150913
## [46]:  test's multi_logloss:0.14881
## [47]:  test's multi_logloss:0.146883
## [48]:  test's multi_logloss:0.145029
## [49]:  test's multi_logloss:0.142752
## [50]:  test's multi_logloss:0.140551
## [51]:  test's multi_logloss:0.1377
## [52]:  test's multi_logloss:0.137249
## [53]:  test's multi_logloss:0.136323
## [54]:  test's multi_logloss:0.134558
## [55]:  test's multi_logloss:0.1333
## [56]:  test's multi_logloss:0.13346
## [57]:  test's multi_logloss:0.130618
## [58]:  test's multi_logloss:0.130078
## [59]:  test's multi_logloss:0.130093
## [60]:  test's multi_logloss:0.130498
## [61]:  test's multi_logloss:0.129996
## [62]:  test's multi_logloss:0.129824
## [63]:  test's multi_logloss:0.128687
## [64]:  test's multi_logloss:0.127766
## [65]:  test's multi_logloss:0.127872
## [66]:  test's multi_logloss:0.127134
## [67]:  test's multi_logloss:0.126798
## [68]:  test's multi_logloss:0.126435
## [69]:  test's multi_logloss:0.124821
```

```
## [70]:  test's multi_logloss:0.122732
## [71]:  test's multi_logloss:0.12283
## [72]:  test's multi_logloss:0.122579
## [73]:  test's multi_logloss:0.121438
## [74]:  test's multi_logloss:0.121018
## [75]:  test's multi_logloss:0.120896
## [76]:  test's multi_logloss:0.120183
## [77]:  test's multi_logloss:0.119296
## [78]:  test's multi_logloss:0.118764
## [79]:  test's multi_logloss:0.118859
## [80]:  test's multi_logloss:0.117877
## [81]:  test's multi_logloss:0.117577
## [82]:  test's multi_logloss:0.11721
## [83]:  test's multi_logloss:0.117384
## [84]:  test's multi_logloss:0.117254
## [85]:  test's multi_logloss:0.116496
## [86]:  test's multi_logloss:0.116479
## [87]:  test's multi_logloss:0.116928
## [88]:  test's multi_logloss:0.117271
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [89]:  test's multi_logloss:0.115571
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [90]:  test's multi_logloss:0.116195
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [91]:  test's multi_logloss:0.117427
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [92]:  test's multi_logloss:0.116701
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [93]:  test's multi_logloss:0.116937
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [94]:  test's multi_logloss:0.117191
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [95]:  test's multi_logloss:0.117626
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [96]:  test's multi_logloss:0.117224
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [97]:  test's multi_logloss:0.117451
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [98]:  test's multi_logloss:0.116991
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [99]:  test's multi_logloss:0.116914
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [LightGBM] [Warning] No further splits with positive gain, best gain: -inf
## [100]:  test's multi_logloss:0.116959
```

```
# Predict using the LightGBM model
lgb_predictions <- predict(lgb_model, test_data)
lgb_predictions_class <- max.col(matrix(lgb_predictions, ncol = length(unique(train_labels)), byrow = TR

# Evaluate the LightGBM model
lgb_conf_matrix <- confusionMatrix(as.factor(lgb_predictions_class), as.factor(as.numeric(test_labels) +
print("LightGBM Confusion Matrix:")
```

```
## [1] "LightGBM Confusion Matrix:"
```

```
print(lgb_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4
##          0 18 14 18 19 14
##          1 20 18 17 17 16
##          2 20 15 15 22 15
##          3 21 11 17 23 17
##          4 23 19 16 21 18
##
## Overall Statistics
##
##                Accuracy : 0.2072
##                  95% CI : (0.1704, 0.2479)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : 0.8828
##
##                   Kappa : 0.0098
##
```

```
##  Mcnemar's Test P-Value : 0.7968
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4
## Sensitivity           0.17647  0.23377  0.18072   0.2255  0.22500
## Specificity           0.80994  0.80926  0.80055   0.8070  0.78297
## Pos Pred Value        0.21687  0.20455  0.17241   0.2584  0.18557
## Neg Pred Value        0.76731  0.83427  0.80952   0.7775  0.82133
## Prevalence            0.22973  0.17342  0.18694   0.2297  0.18018
## Detection Rate        0.04054  0.04054  0.03378   0.0518  0.04054
## Detection Prevalence  0.18694  0.19820  0.19595   0.2005  0.21847
## Balanced Accuracy     0.49321  0.52152  0.49064   0.5163  0.50398
```

```r
# SVM Model
dtm_svm <- dtm
dtm_svm <- DocumentTermMatrix(corpus, control = list(
  weighting = weightTfIdf,
  stopwords = TRUE,
  bounds = list(global = c(5, Inf), # Terms must appear in at least 5 documents
  dictionary = setdiff(Terms(dtm_svm), "portrayed")) # Exclude "portrayed"
))
# Convert the document-term matrix to a matrix
full_matrix <- as.matrix(dtm_svm)
# Split the data into training and testing sets
set <- createDataPartition(data$category, p = 0.8, list = FALSE)
train_data <- full_matrix[set, ]
test_data <- full_matrix[-set, ]
# Ensure target variable is correctly set
train_labels <- data$category[set]
test_labels <- data$category[-set]

# Train SVM model
model_svm <- svm(train_data, as.factor(train_labels), kernel = "linear")
```

```r
# Predict using the trained model
predictions_svm <- predict(model_svm, test_data)
# Calculate the confusion matrix
confMat_svm <- confusionMatrix(as.factor(predictions_svm), as.factor(test_labels))
print(confMat_svm)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      business entertainment politics sport tech
##   business            98             2        0     0    1
##   entertainment        0            72        0     0    0
##   politics             3             2       83     0    0
##   sport                0             0        0   102    0
##   tech                 1             1        0     0   79
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9775
##                  95% CI : (0.959, 0.9891)
##     No Information Rate : 0.2297
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9717
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: business Class: entertainment Class: politics
## Sensitivity                   0.9608               0.9351          1.0000
## Specificity                   0.9912               1.0000          0.9861
## Pos Pred Value                0.9703               1.0000          0.9432
## Neg Pred Value                0.9883               0.9866          1.0000
## Prevalence                    0.2297               0.1734          0.1869
## Detection Rate                0.2207               0.1622          0.1869
## Detection Prevalence          0.2275               0.1622          0.1982
## Balanced Accuracy             0.9760               0.9675          0.9931
##                      Class: sport Class: tech
## Sensitivity                1.0000      0.9875
## Specificity                1.0000      0.9945
## Pos Pred Value             1.0000      0.9753
## Neg Pred Value             1.0000      0.9972
## Prevalence                 0.2297      0.1802
## Detection Rate             0.2297      0.1779
## Detection Prevalence       0.2297      0.1824
## Balanced Accuracy          1.0000      0.9910
```