SVM & RF

Andrea Filiberto Lucas

2024-06-05

Support Vector Machines (SVMs) & Random Forests (RFs) Analysis

Introduction

This study investigates the use of Support Vector Machines (SVMs) and Random Forests (RFs) in the context of respiratory sound classification. SVMs and RFs are two popular machine learning algorithms used for classification tasks due to their ability to handle complex data sets and resistance to over fitting.

Support Vector Machine (SVM)

SVMs are highly effective supervised learning methods for classification and regression tasks. They work by identifying the hyper plane that best separates different classes in the feature space, maximizing the margin between them. SVMs are particularly effective in high-dimensional spaces and can handle non-linear decision boundaries using kernel functions.

Random Forest (RF)

Random Forests are ensemble learning methods that construct a large number of decision trees during training and output the mode of the classes (classification) or the mean prediction (regression) of each tree. RFs are well-known for their ability to withstand over fitting, scale, and handle high-dimensional data. They are particularly useful for classification tasks that require complex interactions and non-linear relationships between features.

Loading Necessary Libraries

```
# Load necessary libraries
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(fs)
library(purrr)
library(dplyr)
library(seewave)
```

```
##
## Attaching package: 'seewave'
##
## The following object is masked from 'package:lubridate':
##
##     duration
##
## The following object is masked from 'package:readr':
##
##     spec
```

```
library(tuneR)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
##
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(caTools)
library(e1071)
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:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

Data Preparation

```
# Define the file path
file_path <- "../Respiratory_Sound_Database/audio_and_txt_files/"

# Check if the directory exists
if (!dir_exists(file_path)) {
  stop("The specified directory does not exist.")
}

# Extracting only .txt files to a list
list_of_files <- fs::dir_ls(file_path, recurse = TRUE, regexp = "\\.txt$")

# Check if any .txt files are found
if (length(list_of_files) == 0) {
  stop("No .txt files were found in the specified directory.")
}
```

```r
# Store the data read from files
data_store <- lapply(list_of_files, read.table)
```

Reading the Data

Reading the ".txt" Files

```r
# Combine data frames into a single data frame
single_dataframe <- purrr::map_dfr(data_store, as.data.frame)

# Extract numbers from file names
extracted_nos <- purrr::map_dbl(list_of_files, ~ as.numeric(stringr::str_extract(.x, "\\d+")))

# Calculate number of rows in each data frame
number_of_rows <- purrr::map_int(data_store, ~ nrow(.))

# Check if file exists and remove if it does
if (file.exists("./PatientID.txt")) {
  file.remove("./PatientID.txt")
  cat("The Existing PatientID.txt file removed.\n")
}
```

```
## The Existing PatientID.txt file removed.
```

```r
# Write patient IDs to file with corresponding number of repetitions
purrr::pwalk(list(extracted_nos, number_of_rows), function(patient_id, num_rows) {
  text_file <- file.path("./PatientID.txt")
  cat(rep(patient_id, num_rows), file = text_file, append = TRUE, sep = "\n")
})

# Delay - 0.5 seconds
Sys.sleep(0.5)

cat("Paitent IDs written to PatientID.txt file.\n")
```

```
## Paitent IDs written to PatientID.txt file.
```

```r
# Read patient IDs from file
PID_store <- read.table(file = './PatientID.txt')

# Calculate the number of rows in the PID_store dataframe
num_rows <- nrow(PID_store)
cat("Number of rows in PID_store dataframe:", num_rows)
```

```
## Number of rows in PID_store dataframe: 6898
```

Data Manipulation and Analysis: Selecting, Calculating, and Renaming Columns

```r
# Select columns V1 to V4 from single_dataframe and calculate V5 and V6
single_dataframe <- single_dataframe %>%
  select(V1, V2, V3, V4) %>%
  mutate(V5 = V2 - V1, # Calculate difference between V2 and V1
         V6 = case_when( # Create V6 based on conditions
           V3 == 1 ~ 1,
           V4 == 1 ~ 1,
```

```
          TRUE ~ 0
        ))

# Rename columns for better readability
single_dataframe <- single_dataframe %>%
  rename("Start_rec" = V1,
         "End_rec" = V2,
         "Crackles" = V3,
         "Wheezes" = V4,
         "End-Start" = V5,
         "Normal_VS_Abnormal" = V6)

# Combine single_dataframe with PID_store
patient_data <- cbind(single_dataframe, PID_store)
```

Reading the ".wav" Files

Warning: Estimated Time of Completion (ETC): 4-5 minutes on a MacBook Air M2

```
# Open a connection to write frequency data
freq_file_conn <- file("./Freq.txt", "w")


# List all WAV files in the specified directory
list_of_WAV_files <- list.files(
  path = file_path,
  recursive = TRUE,
  pattern = "\\.wav$",
  full.names = TRUE
)

# Initialize an empty list to store frequency data
freq <- list()

# Iterate over each WAV file
for (i in seq_along(list_of_WAV_files)) {
  # Read the WAV file
  current_file <- list_of_WAV_files[i]
  wav <- readWave(current_file)

  # Calculate the spectrogram
  wave <- spectro(wav, plot = FALSE)

  # Extract the maximum frequency
  max_freq <- max(wave$freq)

  # Get the number of rows for the current WAV file
  num_rows <- number_of_rows[[i]]

  # Write the maximum frequency to the file for each row
  for (j in seq(from = 1, to = num_rows, by = 1)) {
    textfile <- file.path("./Freq.txt")
    printer <- file(textfile, "a+")
    write(c(max_freq), textfile, append = TRUE)
```

```
    close(printer)
  }
}

# Close the file connection
close(freq_file_conn)
cat("Frequency data written to Freq.txt file.\n")
```

```
## Frequency data written to Freq.txt file.
```

```
# Read frequency data from file
Freq_store <- read.table(file = './Freq.txt')

# Calculate the number of rows in the Freq_store dataframe
num_rows_freq <- nrow(Freq_store)
cat("Number of rows in Freq_store dataframe:", num_rows_freq, "\n")
```

```
## Number of rows in Freq_store dataframe: 6898
```

Integrating Frequency Data into Patient Records: Combining and Labeling Columns

```
# Combine patient_data  and Freq_store
patient_data  <- cbind(patient_data , Freq_store)

# Set column names
patient_data <- setNames(patient_data , c("Start_rec","End_rec","Crackles","Wheezes","Start_End","Normal
```

Integrating Demographic Information into Patient Records: Matching and Updating Sex and Age

```
# Read the demographic_info.txt file into a data frame
demographic_info <- read.table("../Respiratory_Sound_Database/demographic_info.txt")

# Group demographic_info by V3 and select relevant columns
container <- demographic_info %>%
  group_by(V3) %>%
  select(ID = V1, Age = V2, Sex = V3)

# Iterate through rows of patient_data and update Sex and Age based on ID match
for (i in 1:nrow(patient_data)) {
  matching_row <- container[container$ID == patient_data$ID[i], ]
  if (nrow(matching_row) > 0) {
    # If ID is found, update Sex and Age
    patient_data$Sex[i] <- matching_row$Sex
    patient_data$Age[i] <- matching_row$Age
  }
}

# Reorder columns in patient_data to place Normal_VS_Abnormal after Age
patient_data <- patient_data %>%
relocate(Normal_VS_Abnormal, .after = Age)
```

```r
cat("Patient_data has been updated!\n")
```

```
## Patient_data has been updated!
```

Preparing Patient Data Subset: Transformation & Conversion

```r
# Extract columns 5 to 10 from patient_data and store in patient_data_subset
patient_data_subset <- patient_data[5:10]

# Convert "M" to 1 and "F" to 0 in the Sex column
patient_data_subset$Sex <- as.integer(patient_data_subset$Sex == "M")

# Convert columns to numeric type
patient_data_subset$Start_End <- as.numeric(patient_data_subset$Start_End)
patient_data_subset$Sex <- as.numeric(patient_data_subset$Sex)
patient_data_subset$Age <- as.numeric(patient_data_subset$Age)
patient_data_subset$ID <- as.numeric(patient_data_subset$ID)
patient_data_subset$Frequency <- as.numeric(patient_data_subset$Frequency)

# Convert Normal_VS_Abnormal to a factor with levels 0 and 1
patient_data_subset$Normal_VS_Abnormal <- factor(patient_data_subset$Normal_VS_Abnormal, levels = c(0,
```

Data Splitting and Scaling for Modeling: Training and Test Set Preparation

```r
# Set seed for reproducibility
set.seed(123)

# Split the data into training and test sets
split_indices <- sample.split(patient_data_subset$Normal_VS_Abnormal, SplitRatio = 0.75)
training_set <- patient_data_subset[split_indices, ]
test_set <- patient_data_subset[!split_indices, ]

# Scale the features in the training_set & test_set, excluding the 6th column (target variable)
training_set[-6] = scale(training_set[-6])
test_set[-6] = scale(test_set[-6])
```

Model Creation and Prediction

Data Cleaning

```r
# Remove rows with missing values
training_set <- na.omit(training_set)
test_set <- na.omit(test_set)
```

SVM

```r
# Create SVM classifier
svm_classifier <- svm(Normal_VS_Abnormal ~ .,
                      data = training_set,
                      type = 'C-classification',
                      kernel = 'radial')
```

```r
# Make predictions using the SVM classifier
svm_predictions <- predict(svm_classifier, newdata = test_set[, -6])
```

RF

```r
# Train random forest classifier
classifier_randomForest <- randomForest(x = training_set[-6],
                                        y = training_set$Normal_VS_Abnormal,
                                        ntree = 15)

# Predict using the trained classifier
rf_prediction <- predict(classifier_randomForest, newdata = test_set[-6])
```

Evaluation

Confusion Matrix Generation: SVM and Random Forest Classifiers

```r
# Create Confusion Matrix for Random Forest predictions
cm_rf <- table(test_set[, 6], rf_prediction)

# Create Confusion Matrix for SVM predictions
cm_svm <- table(test_set[, 6], svm_predictions)
```

```r
# Display Confusion Matrix for Random Forest
cat("Confusion Matrix for Random Forest predictions:\n")
```

Display Confusion Matrix - SVM & RF

```
## Confusion Matrix for Random Forest predictions:
```

```r
print(cm_rf)
```

```
##    rf_prediction
##       0   1
##   0 709 193
##   1 243 566
```

```r
# Display Confusion Matrix for SVM
cat("\nConfusion Matrix for SVM predictions:\n")
```

```
##
## Confusion Matrix for SVM predictions:
```

```r
print(cm_svm)
```

```
##    svm_predictions
##       0   1
##   0 600 302
##   1 217 592
```

Accuracy Measurements

```r
# Calculate accuracy for SVM
accuracy_svm <- sum(diag(cm_svm)) / sum(cm_svm)
```

```r
# Calculate accuracy for Random Forest
accuracy_rf <- sum(diag(cm_rf)) / sum(cm_rf)

# Convert accuracy values to percentages
accuracy_svm_percent <- round(accuracy_svm * 100, 2)
accuracy_rf_percent <- round(accuracy_rf * 100, 2)

# Print out accuracy as percentages for SVM
cat("Accuracy for SVM:", accuracy_svm_percent, "%\n")
```

```
## Accuracy for SVM: 69.67 %
```

```r
# Print out accuracy as percentages for Random Forest
cat("Accuracy for Random Forest:", accuracy_rf_percent, "%\n")
```

```
## Accuracy for Random Forest: 74.52 %
```

Bar Chart

```r
# Create a data frame for accuracy
accuracy_df <- data.frame(Classifier = c("SVM", "Random Forest"), Accuracy = c(accuracy_svm_percent, ac

# Plot bar plot
barplot(accuracy_df$Accuracy, names.arg = accuracy_df$Classifier, col = c("blue", "red"), main = "Classi
```