University of Sunderland

Name: ROTIMI, Ayodeji Moses

Module Code: CETM24

CETM24 - Data Science Fundamentals

STUDENT ID : 229783515

A Comparative Study on the Effectiveness of Random Forest and K-Nearest Neighbors in Speech Emotion Prediction with STFT Features

INTRODUCTION

In the digital age, Artificial Intelligence (AI) plays a vital role in simplifying daily tasks, with facial expressions often used to identify human emotions. Recent research has delved into using AI to detect emotions through facial expressions (Kittichaiwatthana & Kanjanawattana, 2020). However, facial expressions can differ even in similar emotional states, limiting the accuracy of facial emotion identification. To mitigate this, exploring emotion recognition through speech analysis has emerged as an alternative approach (Lugović et al., 2016).

Mel-Frequency Cepstral Coefficients (MFCC), Linear Predictive Cepstral Coefficients (LPCC), and Short-Time Fourier Transform (STFT) are some of the feature extraction techniques used in audio signal processing. MFCC focuses on the Mel-frequency scale, which approximates auditory frequency response. It involves taking the logarithm of the magnitude of the Fourier Transform of a signal, followed by a cosine transform. MFCC is sensitive to the quantization noise introduced during cepstral analysis. Unlike static features, such as MFCCs, the time-frequency representation of STFT is well-suited for tasks like emotion recognition. This is because it is able to reveal spectral patterns associated with different emotional states. STFT decomposes frequency content of a signal into short overlapping windows. STFT's ability to reveal pitch variations and harmonics, makes it superior in extracting valuable information for audio classification, which gives it an edge over other acoustic feature type (Farheen & Chatterjee 2021).

Moreover, our methodology incorporates the Boruta feature selection algorithm, Random Forest, and K-Nearest Neighbors algorithms to develop the speech model. Boruta operates by weighing the importance of actual features against randomly generated shadow attributes. It is particularly useful in scenarios where the relationship between features and the target variable is complex. While KNN algorithm works by assigning the most common class label among the K neighbors as the predicted label for the input data point because it is built on the principle that similar data points tend to share similar labels or values, the RF algorithm combines the outputs of multiple decision trees to arrive at a singular result (Hastie et al., 2009).

The report focuses on predicting emotions using the Toronto Emotional Speech Set (TESS), with seven emotion classes—Fear, Happy, Sad, Angry, Pleasantly Surprised, Disgust, and Neutral—for predicting emotions. Our approach involves three key steps: data preprocessing, audio feature extraction and modeling. We aim to refine the audio dataset using preprocessing techniques and then leverage signal processing methods to extract audio features essential for emotion prediction. Each audio sample undergoes a Hamming windowing technique to enhance its spectral characteristics. To extract significant features, we implement an acoustic function based on STFT principles. This function transforms the audio waveform into matrices representing both time and frequency domains, enabling us to capture characteristics essential for emotion identification within the audio data. The Boruta feature selection algorithm will eliminate irrelevant features from the extracted audio data, and Random Forest and K Nearest Neighbors algorithms will be utilized for modeling.

2.0 METHODOLOGY

2.1 Data Sourcing: In 2010, Dupuis & Pichora-Fuller developed the Toronto Emotional Speech Set (TESS), available on Kaggle. To address gender bias, the dataset features two English-speaking actresses, aged 26 and 64, with musical training. Each actress recorded 200 target words for seven emotions (anger, disgust, fear, happiness, pleasant surprise, sadness, neutrality). The dataset comprises 2800 ".Wav" audio files, organized by actress and emotion in folders. The audio files follow the WAV format**.**

2.2 Data Preprocessing

2.2.1 Feature Extraction: The extraction of audio features from WAV files is carried out using the "extract_features" function. The function performs windowing, Fourier transforms, and statistical computations on both time and frequency domains. A tryCatch block is utilized in the function to drop bad audio files. The process is carried out for all 14 subfolders in the Tess directory .

We adopted Farheen & Chatterjee's (2021) approach in the extraction of STFT. The feature extraction process involves windowing the audio signal by the signal is divided into short overlapping segments. Windowing helps in capturing local characteristics of the signal within each segment. Following windowing, the extraction takes place in both the time and frequency domains. In the time domain, the short overlapping segments of the signal are analyzed to understand its temporal characteristics. This is essential for capturing changes over time, where the temporal evolution of emotional expression is crucial.

In the frequency domain, the STFT is applied to each segment. STFT breaks down an audio signal into short segments, analyzes the frequency content of each segment, and extracts statistical features to characterize the frequency domain properties of the signal. The time-varying frequency content is necessary, as emotions are often expressed through variations in pitch, intensity, and other frequency-related features.

timeM: the median of time curve captures the central trend in the audio sample's temporal behavior.

timeP1: Initial percentile of time curve captures the start of the emotional expression.

timeP2: terminal percentile of time curve value is the maximum time point in the audio waveform

timeIPR: interpercentile range of time curve represents the range between the initial and terminal percentiles. It captures the duration or span of the emotional expression.

freqM: median of frequency curve represents the central value of the frequency curve.It captures the prevalent pitch or tone of the emotional expression.

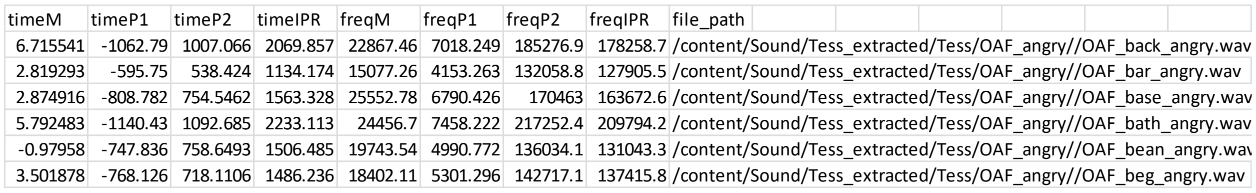
freqP1: initial percentile of frequency curve denotes the lowest frequency value within the audio file.

freqP2: terminal percentile of frequency curve is the highest frequency value in the audio.

freqIPR: interpercentile range of frequency curve captures the spread or dispersion of frequencies in the audio sample.

The extracted features for each emotion subfolder are saved as CSV files. At the end of the extraction process the files are imported back to R studio and appended as a single data frame using the “list.files” function.

Table 2: Head of the 2798 Extracted Audio Features(audio files, from Dupuis & Pichora-Fuller, 2010)



timeM	timeP1	timeP2	timeIPR	freqM	freqP1	freqP2	freqIPR	file_path
6.715541	-1062.79	1007.066	2069.857	22867.46	7018.249	185276.9	178258.7	/content/Sound/Tess_extracted/Tess/OAF_angry//OAF_back_angry.wav
2.819293	-595.75	538.424	1134.174	15077.26	4153.263	132058.8	127905.5	/content/Sound/Tess_extracted/Tess/OAF_angry//OAF_bar_angry.wav
2.874916	-808.782	754.5462	1563.328	25552.78	6790.426	170463	163672.6	/content/Sound/Tess_extracted/Tess/OAF_angry//OAF_base_angry.wav
5.792483	-1140.43	1092.685	2233.113	24456.7	7458.222	217252.4	209794.2	/content/Sound/Tess_extracted/Tess/OAF_angry//OAF_bath_angry.wav
-0.97958	-747.836	758.6493	1506.485	19743.54	4990.772	136034.1	131043.3	/content/Sound/Tess_extracted/Tess/OAF_angry//OAF_bean_angry.wa
3.501878	-768.126	718.1106	1486.236	18402.11	5301.296	142717.1	137415.8	/content/Sound/Tess_extracted/Tess/OAF_angry//OAF_beg_angry.wav

2.3 **Data Transformation**

A new column ('emotion') is stripped from the 'file_path' column with the forward slash as a delimiter by extracting a specific text section occurring after the underscore but before the dot in the filename. This extracted text represents emotions associated with the files. Subsequently, a 'second_index' column is created by further parsing the 'emotion' column using the underscore as a delimiter, which stores only the second part of the split. The ‘file_path’ and emotions columns are dropped while the ‘second_index’ column is then renamed as ‘emotion’.

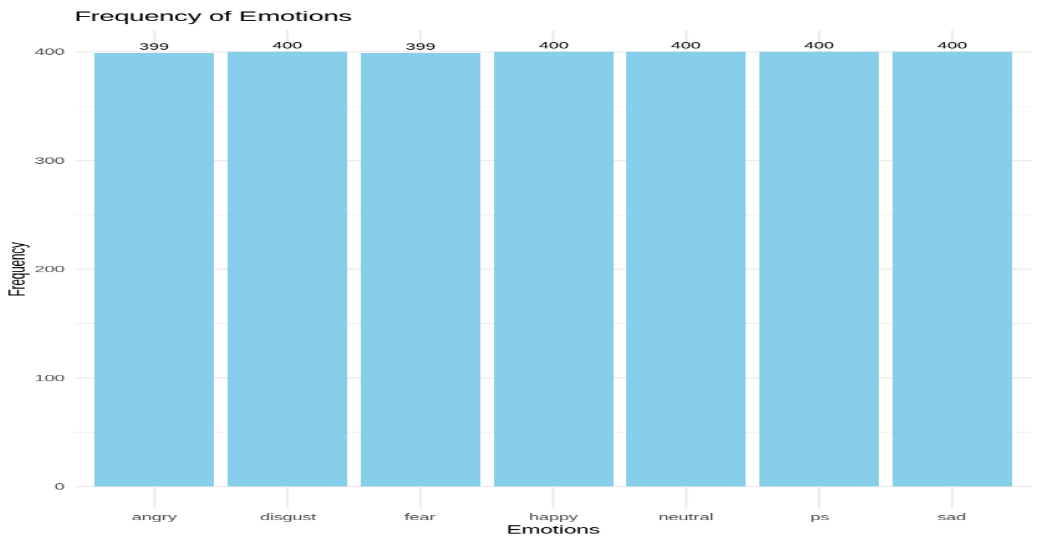


Figure 1: Distribution of Emotion Audio Classes

The total number of samples left after data extraction and transformation is 2798 because it was discovered that the fear and angry class had 1 bad audio file.

2.3.1 Data Scaling: standardization, particularly Z-score normalization, is essential before modeling for distance-based algorithms like KNN. If the features are on different scales, those with larger magnitudes might disproportionately influence the distance computations. In the report, the numeric columns are identified and selected from the data frame. Then, Z-score standardization is applied to the numeric columns. All the input features were identified as numeric, while the target column is identified as a character variable which was converted to a nominal factor variable.

2.3.2 Feature Selection

Boruta feature selection method is used on the scaled data frame. The feature selection algorithm helps to determine the significant variables for predicting speech emotions. Boruta works by comparing the importance of actual attributes against random shadow attributes. After 10 iterations taking approximately 43 seconds, Boruta confirmed all eight attributes as highly significant in predicting speech emotion.

Table 4: Boruta Attribute Statistics Table

meanImp	medianImp	minImp	maxImp	normHits	decision
66.3347	66.42802	61.61254	69.67934	1	Confirmed
40.09091	39.933	38.80782	41.72314	1	Confirmed
38.98109	39.04587	37.15997	40.03635	1	Confirmed
37.32822	37.50631	36.03804	38.3631	1	Confirmed
61.54842	61.30829	57.99828	66.41895	1	Confirmed
66.47674	67.24991	61.69844	68.80315	1	Confirmed
62.03521	61.93373	59.2792	64.17761	1	Confirmed
68.54485	68.55763	65.45642	72.03121	1	Confirmed

2.4 Model Development

We built Random Forest and KNN models to predict the target variable from the already transformed data frame. 80% of the data is used for training while the remaining 20 percent is used as test set.

2.4.1 Random Forest

RF adopts the Bagging (Bootstrap Aggregating) principle, which involves creating multiple subsets of the original training dataset through random sampling with replacement. By introducing randomness in both the data sampling and feature selection processes, RF reduces overfitting and enhance model generalization (Hastie et al., 2009).

The Random Forest model is trained using the training data, where it learns to predict emotions based on the 8 signal audio features. The trained RF model is then used to make predictions on the test data.We then use the confusion Matrix function to evaluate the RF model performance on the test set.

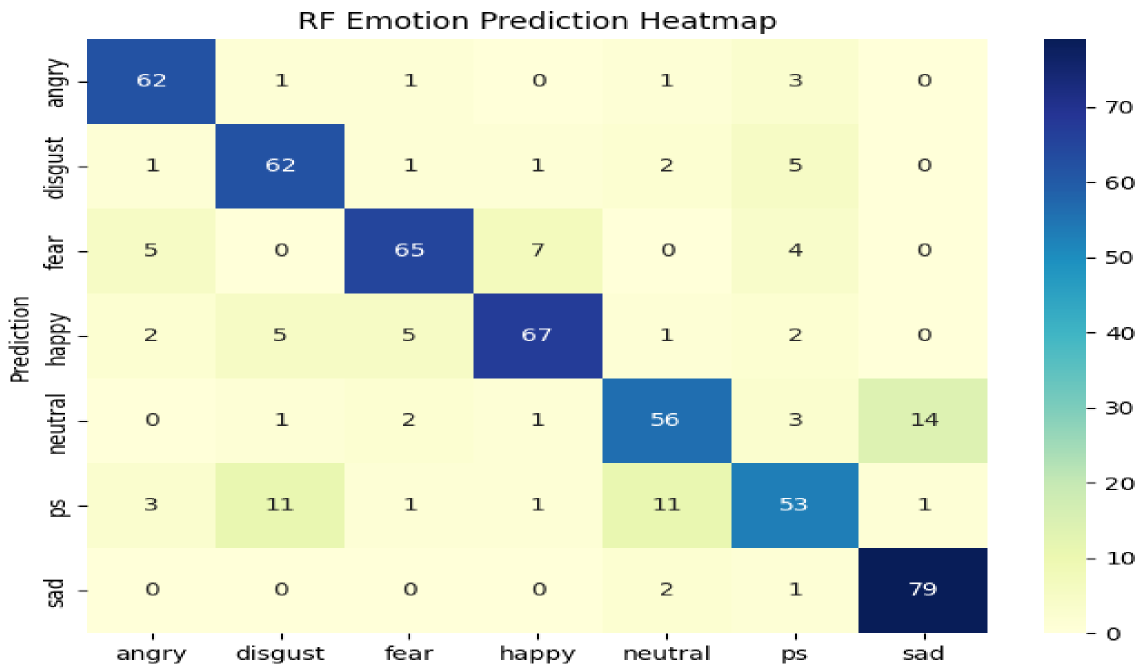


Figure 2: Confusion Matrix Heat Map of the Random Forest Prediction

Table 5: Performance of the RF model across classes

	Class						
Metrics	angry	disgust	fear	happy	neutral	ps	sad
Sensitivity	0.9118	0.8611	0.8025	0.8171	0.7273	0.6543	0.9634
Specificity	0.9768	0.9618	0.9784	0.9783	0.9635	0.961	0.9675
Pos Pred Value	0.8493	0.775	0.8667	0.8701	0.7671	0.7465	0.8404
Neg Pred Value	0.9872	0.9784	0.9658	0.9678	0.9553	0.9407	0.9933
Prevalence	0.1252	0.1326	0.1492	0.151	0.1418	0.1492	0.151
Detection Rate	0.1142	0.1142	0.1197	0.1234	0.1031	0.0976	0.1455

Detection Prevalence	0.1344	0.1473	0.1381	0.1418	0.1344	0.1308	0.1731
Balanced Accuracy	0.9443	0.9114	0.8904	0.8977	0.8454	0.8077	0.9654

2.4.2 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm operates on the principle that similar instances in a dataset tend to exhibit similar behaviors or outcomes. The algorithm stores the entire training dataset and classifies new instances based on the majority class of their K nearest neighbors, where K is a user-defined parameter. This flexibility allows KNN to adapt to various types of data. However, KNN's performance is sensitive to the choice of distance metric and the value of K, requiring careful tuning for optimal results (Hastie et al., 2009).

To predict emotions using KNN, using the 80-20 split we divide the pre-processed scaled data into training and testing subsets. Cross-validation is used to determine the best 'k' parameter for the KNN model.

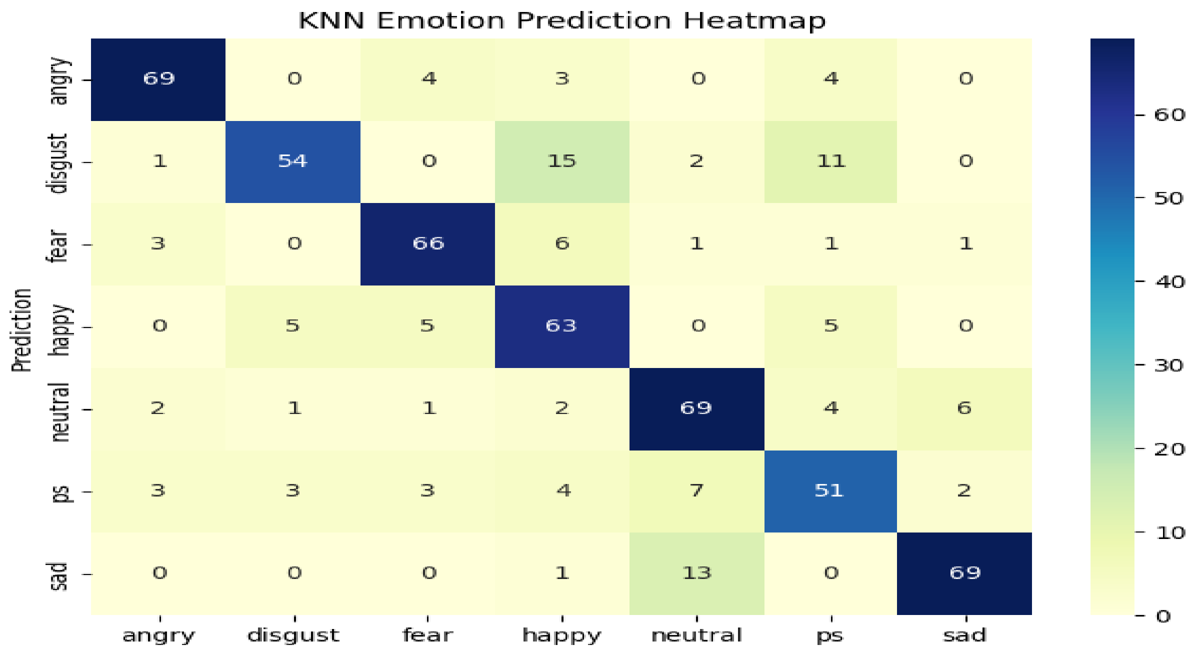


Figure 3: Confusion Matrix Heat Map of the KNN Prediction

Table 6: Performance of the KNN model Accross classes

Metric	angry	disgust	fear	happy	neutral	ps	sad
Sensitivity	0.8625	0.6506	0.8462	0.8077	0.8118	0.69863	0.8313
Specificity	0.9812	0.98113	0.973	0.9357	0.9516	0.94867	0.9811
Pos Pred Value	0.8846	0.85714	0.8354	0.6702	0.75	0.67105	0.8846
Neg Pred Value	0.9772	0.94165	0.9751	0.9678	0.9658	0.95455	0.971
Prevalence	0.1429	0.14821	0.1393	0.1393	0.1518	0.13036	0.1482
Detection Rate	0.1232	0.09643	0.1179	0.1125	0.1232	0.09107	0.1232
Detection Prevalence	0.1393	0.1125	0.1411	0.1679	0.1643	0.13571	0.1393
Balanced Accuracy	0.9219	0.81587	0.9096	0.8717	0.8817	0.82365	0.9062

Table 7: Overall statistics of the Random Forest and KNN Model

Overall Statistics	RF	KNN
Accuracy	0.8177	0.7875
95% CI	(0.7826, 0.8493)	(0.7513, 0.8207)
No Information Rate	0.151	0.1518
P-Value [Acc > NIR]	< 2.2e-16	< 2.2e-16
Kappa	0.7872	0.7521

Table 8: Comparison with Other Research **(**Reakaa and Haritha 2021)

EMOTIONS	SVM Sensitivity	KNN Sensitivity	Our RF Sensitivity
Disgust	0.62	0.53	0.8611
Fearful	0.67	0.59	0.8025
Happy	0.5	0.47	0.8171
Neutral	0.53	0.35	0.7273
Sad	0.55	0.45	0.9634

3.0 DISCUSSION AND CONCLUSION

The comparison between Random Forest (RF) and K-Nearest Neighbors (KNN) models shows that RF significantly outperforms KNN in key metrics . RF demonstrates superior performance in sensitivity, specificity, positive predictive value, and balanced accuracy across multiple classes (Table 5). Particularly in 'sad' and 'angry' classes, RF exhibits notably higher sensitivity and specificity. KNN, while competitive, generally lags behind RF in these key metrics, although it maintains a fair balance of sensitivity and specificity(Table 6). For certain classes like 'neutral' and 'sad' KNN presents better sensitivity and specificity. For the overall statistics result (Table 7), RF yielded a higher accuracy of 81.77% compared to KNN's 78.75%, suggesting that RF more accurately predicted the emotional classes. The 95% confidence intervals for RF's accuracy (0.7826 to 0.8493), surpasses KNN's (0.7513 to 0.8207). These collective metrics substantiate RF's as a better model compared to KNN for speech emotion prediction, as evidenced by its higher accuracy, Kappa value, and narrower confidence interval range.

The better model from the research was compared with work of Reakaa and Haritha 2021(Table 8). The comparison shows that sensitivity results obtained from the Random Forest (RF) model in our research significantly outperform those of both the Support Vector Machine (SVM) and K Nearest Neighbors (KNN) algorithms across various emotions. Notably, RF achieves sensitivity scores of 86.11%, 80.25%, and 81.71% for 'Disgust,' 'Fearful,' and 'Happy' emotions, respectively. These scores appears strikingly higher compared to SVM and KNN for the given classes, Therefore audio feature extraction with the use of STFT coupled with the adoption of the RF algorithm is a superior choice for speech emotion classification compared to the SVM and KNN algorithms.

REFRENCE

Dupuis, K., & Pichora-Fuller, M. K. (2010). Toronto emotional speech set (TESS) Collection. University of Toronto, Psychology Department. https://doi.org/10.5683/SP2/E8H2MF

Farheen, N., & Chatterjee, S. (2021). Speech Based Emotion Detection Using R. https://doi.org/10.1007/978-981-33-6881-1_8

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.

Lugović, S., Dunđer, I., Horvat, M., “Techniques and applications of emotion recognition in speech,” in 2016 39th international convention on information and communication technology, electronics and microelectronics (mipro), 2016, pp. 1278–1283.

Piyapong Kittichaiwatthana, P. P., Kanjanawattana, S., “Facial Expression Recognition using Deep Learning,” 2020, p. 41.

Reakaa, S D, and Haritha J. (May 2021). "Comparison study on speech emotion prediction using machine learning." Journal of Physics Conference Series, 1921(1), 012017. DOI:10.1088/1742-6596/1921/1/012017. [CC BY 3.0]

Song, I., Kim, H.-J., Jeon, P. B., “Deep learning for real-time robust facial expression recognition on a smartphone,” in 2014 IEEE International Conference on Consumer Electronics (ICCE), 2014, pp. 564–567.

Appendix:

```The audio recordings consits of 200 target words per emotion by two actresses aged 26 and 64. These recordings are made up of seven emotions: anger, disgust, fear, happiness, pleasant surprise, sadness, and neutrality. Overall, there are 2800 audio files in the dataset with ".Wav" file extension. The audio data is arranged such that each actress and her range of emotions are attached to their respective file names and folders.```

~~~
```{r}
# Unzip the Tess.rar file into the Tess_extracted directory
system('"C:\\Program Files\\WinRAR\\WinRAR.exe" x "C:\\Users\\Desktop\\Tess.rar" "C:\\Users\\Desktop\\Tess_extract\\"')


```
~~~

~~~
```{r}
# View the names of the subfolders in the TESS audio directory
files <- list.files("C:/Users/Desktop/Tess_extract/Tess")
print(files)
```
~~~

To determine the dimension of the audio file, the list.files function goes through a loop that iterates over each WAV file in the Tess folder and subfolders. The code extracts the file name, obtains the file's length in bytes, and adds both the file name and its length to their respective lists. It then aggregate the collected file names and lengths into a data frame named "file_info"

~~~
```{r}
main_directory <- "C:/Users/Desktop/Tess_extract/Tess"


# Get all .wav files within the main directory and its subdirectories
wav_files <- list.files(path = main_directory, pattern = "\\.wav$", recursive = TRUE, full.names =
TRUE)


# Initialize empty lists to store file names and their lengths
file_names <- character()
file_lengths <- numeric()


# Loop through each .wav file
for (file in wav_files) {
  # Extract the file name without the path
  file_name <- basename(file)


  # Store file name
~~~

~~~
  file_names <- c(file_names, file_name)

  # Get file length in bytes
  file_length <- file.info(file)$size

  # Store file length
  file_lengths <- c(file_lengths, file_length)
}

# Combine file names and their lengths into a data frame
file_info <- data.frame(File_Name = file_names, File_Length = file_lengths)
head(file_info)
```

```{r}
# Install the needed audio packages
packages <- c("tuneR", "seewave", "signal")
install.packages(packages)
```
~~~

The extraction of audio features from WAV files is carried out using the "extract_features" function. The function performs windowing, Fourier transforms, and statistical computations on both time and frequency domains. A tryCatch block is utilized in the function to handle bad audio files. The code navigates through the "Yaf-fear" folder housing audio files, retrieves these files, apply the "extract_features" function to each, and aggregate the extracted features into a DataFrame named "features_df"

The process is carried out for all 13 other subfolders in the Tess directory housing the 6 more emotions.

~~~~
```{r}

library(tuneR)

library(dplyr)

library(seewave)

library(signal)


# Function to extract features from an audio file

extract_features <- function(file_path) {

  # Attempt to read the audio file, handle errors

  tryCatch({

    audio <- readWave(file_path)


    hamming_window <- hamming(length(audio))
~~~~

```
windowed_audio <- audio * hamming_window


timeM <- median(windowed_audio@left, na.rm = TRUE)

timeP1 <- quantile(windowed_audio@left, probs = 0.1, na.rm = TRUE)

timeP2 <- quantile(windowed_audio@left, probs = 0.9, na.rm = TRUE)

timeIPR <- timeP2 - timeP1


freq_domain <- abs(fft(audio@left))

freqM <- median(freq_domain, na.rm = TRUE)

freqP1 <- quantile(freq_domain, probs = 0.1, na.rm = TRUE)

freqP2 <- quantile(freq_domain, probs = 0.9, na.rm = TRUE)

freqIPR <- freqP2 - freqP1


return(data.frame(

  timeM = timeM,

  timeP1 = timeP1,

  timeP2 = timeP2,

  timeIPR = timeIPR,

  freqM = freqM,

  freqP1 = freqP1,

  freqP2 = freqP2,
      freqIPR = freqIPR,
      file_path = file_path  # Adding file path for reference
    ))
  }, error = function(e) {
    cat("Error reading file:", file_path, "\n")
    return(NULL)  # Return NULL for the problematic file
  })
}

# Directory containing audio files and subdirectories
directory_path <- "C:/Users/Desktop/Tess_extract/Tess/YAF_fear/"

# Get a list of all audio files within the main directory and its subdirectories
audio_files <- list.files(directory_path, pattern = "\\.wav$", full.names = TRUE, recursive =
TRUE)

# Extract features for each audio file and combine into a DataFrame
features_list <- lapply(audio_files, extract_features)

# Filter out NULL elements resulting from problematic files
features_list <- Filter(Negate(is.null), features_list)
```

~~~
# Combine valid results into a DataFrame
features_df <- do.call(rbind, features_list)


# Show the resulting dataframe
head(features_df)
```
~~~

For every feature extracted for the emotion subfolder, the obtained dataframe is saved in a new directory named Tess as a csv file.

~~~
```{r}
  #Export the data frame as a csv file
  write.csv(features_df, file = "C:/Users/Desktop/YAF_fear.csv", row.names = FALSE)
```
~~~

R code to list the content of all the CSV files in the Voice folder path and append the CSV files into a single data frame.

~~~
```{r}
library(tidyverse)
folder_path <- "C:/Users/Desktp/Voice"
~~~

~~~
# List all CSV files in the folder
csv_files <- list.files(folder_path, pattern = "\\.csv$", full.names = TRUE)


# Read and combine all CSV files into a single dataframe
combined_data <- csv_files %>%
  map_df(read_csv)
head(combined_data)
dim(combined_data)
```
~~~

Here a new column(emtion) is created but it first splits the 'file_path' column by the forward slash delimeter by capturing a part of the text that comes after the underscore but before the dot in the filename. Then, it creates a new column, 'second_index,' by further splitting the 'emotion' column based on the underscore and keeping only the second part. Finally, it updates the dataset by removing unnecessary columns like 'file_path' and 'emotion' while renaming 'second_index' to 'emotion,'.

~~~
```{r}
# Load required library
#library(tidyverse)
# Split the file_path column and extract emotions
df <- combined_data %>%
~~~

~~~
  mutate(emotion = str_split(file_path, "\\/") %>%
         map_chr(., ~str_extract(.x[[length(.x)]], "(?<=_)\\w+(?=\\.)")))


library(stringr)


# Split the 'emotion' column and extract the second part
df$second_index <- str_split(df$emotion, "_") %>% sapply(function(x) x[2])


# Drop the file_path and emotions column
df <- df[, !(names(df) %in% c("file_path","emotion"))]
# rename the second_index column to emotion
df <- df %>%
  rename(emotion = second_index)
dim(df)
head(df)
```


```{r}
#install.packages('caret')
library(caret)
```
~~~

~~~
```{r}
#library(tidyverse)
data <- df
dim(data)
str(data)
data <- data %>% mutate_if(is.character, as.factor)
```

```{r}
str(data)
```

```{r}
library(readr)
library(dplyr)
library(purrr)
```

```{r}
#install.packages("caTools")
~~~

~~~
library(caTools)

#install.packages("randomForest")

library(randomForest)

#install.packages("caret")

library(caret)

```
~~~

~~~
```{r}

install.packages("Boruta")

install.packages("mlbench")

library(Boruta)

library(mlbench)

```
~~~

Here the numeric columns are identified and selected from the dataframe. Then, standardization is applied to the numeric columns, which involves centering the data around zero and scaling by the standard deviation. The scaling technique transforms the numeric values to have a mean of zero and a standard deviation of one. The scaled columns is combined with the original 'emotion' column to create a new dataframe named 'scaled_data.'

~~~
```{r}

data <- df
~~~

~~~
data$emotion <- as.factor(data$emotion)

numerical_columns <- data[, sapply(data, is.numeric)]


# Standardize numerical columns (excluding the 'emotion' column)

scaled_data <- as.data.frame(scale(numerical_columns))


# Combine scaled numerical columns with the 'emotion' column

scaled_data <- cbind(scaled_data, emotion = data$emotion)

str(scaled_data)

```
~~~

Boruta feature selection method is used on the'scaled_data' dataframe. The feature selection algorithm helps to determine influential variables for predicting speech emotions. Boruta works by comparing the importance of actual attributes against random shadow attributes. After 10 iterations taking approximately 43 seconds, Boruta confirmed all eight attributes as highly significant in predicting speech 'emotion'

~~~
```{r}

# Set the seed for reproducibility

set.seed(111)


# Apply Boruta feature selection on 'scaled_data' using 'emotion' as the target variable
~~~

```
# Parameters:

# emotion ~ .: The formula for target and features

# data = scaled_data: The dataset to perform feature selection on

# doTrace = 2: Trace information during the Boruta process

# maxRuns = 500: Maximum number of iterations for the Boruta algorithm

boruta <- Boruta(emotion ~ ., data = scaled_data, doTrace = 2, maxRuns = 500)


# Print the summary or results of Boruta feature selection

print(boruta)


# Plot the Boruta feature importance graph

# Parameters:

# boruta: The Boruta object containing feature selection results

# las = 2: Label axis style (2 denotes perpendicular labels)

# cex.axis = 0.7: Size of axis labels

plot(boruta, las = 2, cex.axis = 0.7)
```

use the Tentative Rough Fix (TRF) function on the 'boruta' object, which was previously generated through Boruta feature selection. Tentative Rough Fix helps in refining the selection made by Boruta,as it further categorizes the attributes as either confirmed, tentative, or rejected.

```{r}

# Apply the Tentative Rough Fix algorithm on the 'boruta' object

bor <- TentativeRoughFix(boruta)


# Print the summary of the Boruta analysis

print(bor)


# View the attribute statistics in the Boruta object

attStats(boruta)
```

Since the 8 independent features have been confirmed to be relevant for predicting emotions we would build a Random Forest model to predict the target variable from the already transformed audio data frame. Inorder to reproduce similar result set.seed function is used. 80% of the data is used for training while the remainng 20 percent is used as test set. The Random Forest model is trained using the training data, where it learns to predict emotions based on the 8 signal audio features except for the 'emotion' target column. The trained RF model is then used to make predictions on the test data.

We then use the "confusionMatrix" function to evaulate the RF model performance on the test set, which compares the emotions predicted by the model with the actual emotions present in the test dataset

```{r}
```

```
# Assign the scaled dataset to 'data'

data <- scaled_data


# Load necessary libraries

library(caTools)

library(randomForest)

library(caret)


# Set seed for reproducibility

set.seed(222)


# Split the data into training and testing sets (80-20 split)

ind <- sample(2, nrow(data), replace = TRUE, prob = c(0.8, 0.2))

train <- data[ind == 1,]

test <- data[ind == 2,]


# Train a Random Forest model excluding the 'emotion' column

rf <- randomForest(emotion ~ . - emotion, data = train, proximity = TRUE)


# Make predictions on the test set using the trained model

speechpred <- predict(rf, test)
```

```
# Calculate and display the confusion matrix for evaluation

confusionMatrix(speechpred, test$emotion)
```

To predict emotions using KNN the data is first processed by by converting the character column to factors and setting seed for reproducibility.Using the 80-20 split we divide the dataset into training and testing subsets.the numeric columns in the dataset are being standardized using Z-score normalization. The preProcess function with the method argument set to c("center", "scale") performs this standardization.

the scaled numeric data is then merged with the 'emotion' column to create the final training and testing sets.

Cross-validation is used to determine the best 'k' parameter for the KNN model. To know how the developed model performs the predictions on the test set is evalueted with confusion matrix.

```{r}
# Display dimensions and structure of the dataset

data <- df
```

```
str(data)

# Convert character columns to factors
data <- data %>% mutate_if(is.character, as.factor)

# Set seed for reproducibility
set.seed(245)

# Split the dataset into training and testing sets (80-20 split)
data_rows <- floor(0.80 * nrow(data))
train_indices <- sample(c(1:nrow(data)), data_rows)
train_data <- data[train_indices,]
test_data <- data[-train_indices,]

# Identify columns containing numeric data
numeric_cols <- sapply(train_data, is.numeric)

# Scale only the numeric columns in the training and testing sets
preProcValues <- preProcess(train_data[, numeric_cols], method = c("center", "scale"))
trainTransformed <- predict(preProcValues, train_data[, numeric_cols])
```

```
testTransformed <- predict(preProcValues, test_data[, numeric_cols])

# Combine scaled numeric data with 'emotion' column to create training and testing sets

trainset <- cbind(trainTransformed, emotion = train_data$emotion)

testset <- cbind(testTransformed, emotion = test_data$emotion)

# Train a k-Nearest Neighbors model using cross-validation (tuning 'k' parameter)

knnModel <- train(

  emotion ~ .,

  data = trainset,

  method = "knn",

  trControl = trainControl(method = "cv"),

  tuneGrid = data.frame(k = c(2,3,4,5,6,7,8,9,10))

)

# Get the best 'k' value from cross-validation

knnModel$bestTune$k

# Train the final KNN model using the best 'k' value

best_model <- knn3(

  emotion ~ .,

  data = trainset,

  k = knnModel$bestTune$k

)

# Make predictions on the test set and calculate confusion matrix for evaluation
```

```
predictions <- predict(best_model, testset, type = "class")

confusionMatrix(predictions, testset$emotion)
```