song_scores

2025-04-19

song_scores and plot_scores functions

```
# ---- PREREQUISITES ----
# 1. 'transforEmotion' Conda environment configured (transforEmotion::setup_miniconda()).
# 2. Required Python packages installed:
#    conda activate transforEmotion
#    pip install transformers torch librosa soundfile sentencepiece accelerate
#    (Ensure 'torch' is correctly installed for your CPU/GPU configuration).
# 3. R package 'transforEmotion' installed and loadable (if text analysis is required).

#' Emotion Analysis in Music (Audio and Lyrics)
#'
#' Combines audio-based emotion analysis (using CLAP) and text-based emotion analysis
#' (using an NLI model via transforEmotion::transformer_scores).
#' Can automatically transcribe lyrics from audio using Whisper or accept pre-supplied lyrics.
#'
#' @param audio_path Path to the audio file (e.g., .wav, .mp3).
#' @param audio_classes Character vector of emotion labels for AUDIO analysis (CLAP).
#' @param text_classes Character vector of emotion labels for TEXT analysis (NLI).
#'   If NULL, text analysis is skipped.
#' @param lyrics Text of the song lyrics (optional). If provided (not NULL),
#'   automatic transcription is skipped, and this text is used for textual emotion analysis.
#'   If NULL, the function may attempt transcription.
#' @param transcribe_audio Logical. If TRUE and `lyrics` is NULL, attempts to transcribe
#'   the audio (or segment) using the specified ASR model. Default: FALSE.
#'   Transcription can be time-consuming.
#' @param start_sec Start time (in seconds) of the segment for AUDIO analysis
#'   and, if `transcribe_audio=TRUE`, for TRANSCRIPTION. If NULL, starts from the beginning of the file
#' @param end_sec End time (in seconds) of the segment. If NULL, processes until the end of the file.
#' @param clap_model_id Identifier of the CLAP model (Hugging Face).
#'   Default: "laion/clap-htsat-unfused".
#' @param nli_model_id Identifier of the NLI model for text (Hugging Face),
#'   used by `transformer_scores`. Default: "joeddav/xlm-roberta-large-xnli".
#' @param asr_model_id Identifier of the ASR (Whisper) model for transcription
#'   (Hugging Face). Default: "openai/whisper-base". Larger models
#'   (e.g., "openai/whisper-large-v3") are more accurate but slower/computationally heavier.
#' @param asr_language Language code for Whisper (e.g., "english", "portuguese").
#'   Assists transcription accuracy. If NULL (default), Whisper attempts language detection.
#' @param verbose Logical. Print progress messages. Default: TRUE.
#'
#' @return A list containing:
#'   \item{audio_scores}{CLAP emotion scores for the audio/segment (named vector or NA upon failure).}
#'   \item{text_scores}{NLI emotion scores for the lyrics (output of `transformer_scores` or NULL if no
#'   \item{transcribed_text}{The text transcribed by Whisper (string or NULL if not transcribed/failed)
#'   \item{text_source}{Indicates the source of the analyzed text ("provided_lyrics", "transcribed", "t
#' @export
#'
#' @examples
#' \dontrun{
#' # --- Prerequisites ---
#' # Ensure reticulate points to the correct conda environment
#' # library(reticulate)
#' # use_condaenv("transforEmotion", required = TRUE)
#' # Ensure Python packages are installed in that environment.
#'
#' # --- Example Usage ---
#' my_song_path <- "path/to/your/song.mp3" # Replace!
#' emotion_labels <- c("joy", "sadness", "anger", "calmness", "neutral")
#'
#' # 1. Analyze audio and use provided lyrics
#' known_lyrics <- "Joyful, joyful, the sun is shining bright..." # Replace!
#' results1 <- song_scores(
#'   audio_path = my_song_path,
#'   audio_classes = emotion_labels,
#'   text_classes = emotion_labels,
#'   lyrics = known_lyrics,
#'   start_sec = 30, # Analyze audio from 30s to 60s
#'   end_sec = 60
#' )
#' print("Results (Provided Lyrics):")
#' print(results1)
#'
#' # 2. Analyze audio (0-30s segment) and TRANSCRIBE lyrics from THIS segment
#' results2 <- song_scores(
#'   audio_path = my_song_path,
#'   audio_classes = emotion_labels,
#'   text_classes = emotion_labels,
#'   transcribe_audio = TRUE,    # REQUEST TRANSCRIPTION
#'   start_sec = 0,
#'   end_sec = 30,
#'   asr_language = "english",   # Specifying language aids accuracy
#'   asr_model_id = "openai/whisper-large-v3" # Example with a larger model
#' )
#' print("Results (Automatic Transcription 0-30s):")
#' print(results2)
#'
#' # 3. Analyze only the audio (entire file), without text analysis
#' results3 <- song_scores(
#'   audio_path = my_song_path,
#'   audio_classes = emotion_labels,
#'   text_classes = NULL # Skip text analysis
#' )
#' print("Results (Audio Only):")
#' print(results3)
#' }
```

```r
song_scores <- function(audio_path,
                        audio_classes,
                        text_classes = NULL, # Allow skipping text analysis
                        lyrics = NULL,
                        transcribe_audio = FALSE,
                        start_sec = NULL,
                        end_sec = NULL,
                        clap_model_id = "laion/clap-htsat-unfused",
                        nli_model_id = "joeddav/xlm-roberta-large-xnli",
                        asr_model_id = "openai/whisper-base",
                        asr_language = NULL,
                        verbose = TRUE) {

  # --- Initial Checks ---
  if (!requireNamespace("reticulate", quietly = TRUE)) {
    stop("Package 'reticulate' is required.", call. = FALSE)
  }
  # Check for transforEmotion only if text_classes are requested
  if (!is.null(text_classes) && !requireNamespace("transforEmotion", quietly = TRUE)) {
    stop("Package 'transforEmotion' is required for text analysis.", call. = FALSE)
  }

  if (!file.exists(audio_path)) {
    stop("Audio file not found: ", audio_path, call. = FALSE)
  }
  if (!is.character(audio_classes) || length(audio_classes) == 0) {
    stop("'audio_classes' must be a non-empty character vector.", call. = FALSE)
  }
   if (!is.null(text_classes) && (!is.character(text_classes) || length(text_classes) == 0)) {
    stop("'text_classes', if provided, must be a non-empty character vector.", call. = FALSE)
  }
  if (!is.null(lyrics) && !is.character(lyrics)) {
     stop("'lyrics', if provided, must be a character vector.", call. = FALSE)
  }


  # --- Validate start_sec and end_sec ---
  offset_py <- reticulate::py_none()
  duration_py <- reticulate::py_none()
  segment_description <- "the entire audio file" # English equivalent

  if (!is.null(start_sec) && !is.null(end_sec)) {
    if (!is.numeric(start_sec) || start_sec < 0) stop("'start_sec' must be numeric >= 0.", call. = FALS
    if (!is.numeric(end_sec) || end_sec <= start_sec) stop("'end_sec' must be numeric > 'start_sec'.", c
    offset_py <- reticulate::r_to_py(as.numeric(start_sec))
    duration_val <- as.numeric(end_sec - start_sec)
    duration_py <- reticulate::r_to_py(duration_val)
    segment_description <- sprintf("the segment from %.2f s to %.2f s (duration: %.2f s)", start_sec, en
  } else if (!is.null(start_sec) || !is.null(end_sec)) {
    stop("Both 'start_sec' and 'end_sec' must be provided, or both must be NULL.", call. = FALSE)
  }

  if (verbose) message(paste("Analysis requested for", segment_description))
  # --- Check Python Modules ---
  required_py_modules <- c("transformers", "torch", "librosa", "accelerate")
  modules_available <- sapply(required_py_modules, reticulate::py_module_available)

  if (!all(modules_available)) {
    missing_modules <- names(modules_available[!modules_available])
    stop(
      "Required Python modules not found: ", paste(missing_modules, collapse = ", "), ".\n",
      "Please install them in the 'transforEmotion' environment: 'pip install ", paste(missing_modules,
      call. = FALSE
    )
  }

  # --- Import Python Modules ---
  if (verbose) message("Importing Python modules (transformers, torch, librosa)...")
  transformers <- reticulate::import("transformers", delay_load = TRUE)
  torch <- reticulate::import("torch", delay_load = TRUE)
  librosa <- reticulate::import("librosa", delay_load = TRUE)


  # --- Initialize Results List ---
  results <- list(
      audio_scores = setNames(rep(NA_real_, length(audio_classes)), audio_classes), # Initialize with N
      text_scores = NULL,
      transcribed_text = NULL,
      text_source = "none" # none, provided_lyrics, transcribed, transcribed_empty, transcribed_failed
  )

  # --- Python Code for CLAP (Adapted from previous function) ---
  reticulate::py_run_string("
import torch
import librosa
from transformers import AutoProcessor, AutoModel # Using AutoModel as in previous version
import numpy as np

def get_clap_scores_py(audio_fpath, text_classes, model_ident, offset=None, duration=None):
    try:
        processor = AutoProcessor.from_pretrained(model_ident)
        model = AutoModel.from_pretrained(model_ident)

        target_sr = processor.feature_extractor.sampling_rate if hasattr(processor.feature_extractor, '
        audio_array, sr = librosa.load(audio_fpath, sr=target_sr, mono=True, offset=offset, duration=du

        if audio_array.size == 0:
             # Translated error message
             return {'error': 'CLAP audio segment is empty.', 'scores': []}

        audio_array = np.array(audio_array)
        inputs = processor(text=text_classes, audios=[audio_array], return_tensors=\"pt\", padding=True

        with torch.no_grad():
            outputs = model(**inputs)
        logits_per_audio = outputs.logits_per_audio[0] # Assume 1 audio processed
        probs = torch.softmax(logits_per_audio, dim=0).numpy()

        return {'error': None, 'scores': probs.tolist()}

    except Exception as e:
         # Translated error message
        return {'error': f'CLAP Error: {str(e)}', 'scores': []}
  ")

  # --- Python Code for ASR (Whisper) ---
  reticulate::py_run_string("
import torch
import librosa
from transformers import pipeline
import numpy as np
import warnings

# Suppress specific pipeline warnings that can clutter the console
warnings.filterwarnings('ignore', message='.*Maximum duration.*')
warnings.filterwarnings('ignore', message='.*Using PipelineChunkIterator.*')

def transcribe_audio_segment_py(audio_fpath, model_ident, offset=None, duration=None, language=None):
    try:
        target_sr = 16000 # Whisper SR
        audio_array, sr = librosa.load(audio_fpath, sr=target_sr, mono=True, offset=offset, duration=du

        if audio_array.size == 0:
             # Translated error message
             return {'error': 'ASR audio segment is empty.', 'text': ''}

        audio_array = np.array(audio_array)
        device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

        # Use try-except for pipeline loading, as it might fail (memory etc.)
        try:
            pipe = pipeline(
                'automatic-speech-recognition',
                model=model_ident,
                device=device,
                chunk_length_s=30,
                stride_length_s=5
            )
        except Exception as pipe_e:
             # Translated error message
             return {'error': f'Error loading ASR pipeline ({model_ident}): {str(pipe_e)}', 'text': ''}

        generate_kwargs = {}
        if language:
          generate_kwargs['language'] = language

        # Use try-except for the transcription itself
        try:
            transcription_result = pipe(audio_array.copy(), generate_kwargs=generate_kwargs)
            transcribed_text = transcription_result['text'].strip() if transcription_result else ''
        except Exception as trans_e:
             # Translated error message
             return {'error': f'Error during ASR transcription: {str(trans_e)}', 'text': ''}

        return {'error': None, 'text': transcribed_text}

    except Exception as e:
        # Catch general errors like librosa.load failure
        # Translated error message
        return {'error': f'General ASR (Whisper) Error: {str(e)}', 'text': ''}
")


# --- 1. Audio Analysis (CLAP) ---
if (verbose) message(paste("Initiating AUDIO analysis with CLAP:", clap_model_id, "..."))
clap_results_py <- tryCatch({
    reticulate::py$get_clap_scores_py(
        audio_fpath = audio_path,
        text_classes = reticulate::r_to_py(audio_classes),
        model_ident = clap_model_id,
        offset = offset_py,
        duration = duration_py
    )
}, error = function(e) {
    list(error = paste("R Error calling CLAP:", e$message), scores = list())
})

# Process CLAP results
if (!is.null(clap_results_py$error)) {
    warning("Audio analysis failed (CLAP): ", clap_results_py$error, call. = FALSE)
    # results$audio_scores remains initialized with NA
} else {
    scores_r <- reticulate::py_to_r(clap_results_py$scores)
    if (length(scores_r) == length(audio_classes)) {
        results$audio_scores <- setNames(scores_r, audio_classes) # Overwrite NA with valid scores
        if (verbose) message("Audio analysis completed.")
    } else {
        warning("Number of CLAP scores (", length(scores_r), ") differs from the number of classes ("
        # Keep NAs in results$audio_scores
    }
}


# --- 2. Text Preparation for Analysis ---
text_to_analyze <- NULL
perform_text_analysis <- !is.null(text_classes) # Only analyze text if text_classes were provided

if (perform_text_analysis) {
  if (!is.null(lyrics)) {
      if (verbose) message("Using provided lyrics ('lyrics').")
      # If lyrics is a vector of multiple strings, concatenate? Or analyze separately?
      # Assuming it's a single string or should be concatenated.
```

```r
   text_to_analyze <- paste(lyrics, collapse = "\n")
   results$text_source <- "provided_lyrics"
 } else if (transcribe_audio) {
   if (verbose) message(paste("Initiating audio TRANSCRIPTION with ASR (Whisper):", asr_model_id,

   asr_results_py <- tryCatch({
     reticulate::py$transcribe_audio_segment_py(
       audio_fpath = audio_path,
       model_ident = asr_model_id,
       offset = offset_py,
       duration = duration_py,
       language = if (!is.null(asr_language)) asr_language else reticulate::py_none()
     )
   }, error = function(e) {
     list(error = paste("R Error calling ASR:", e$message), text = '')
   })

   # Process ASR results
   if (!is.null(asr_results_py$error)) {
     warning("Audio transcription failed (ASR): ", asr_results_py$error, call. = FALSE)
     results$transcribed_text <- NULL # Failed
     results$text_source <- "transcribed_failed" # New state
   } else {
     results$transcribed_text <- reticulate::py_to_r(asr_results_py$text)
     if (!is.null(results$transcribed_text) && nchar(results$transcribed_text) > 0) {
       if (verbose) message("Transcription completed.")
       if(verbose) message(paste("Transcribed text:", substr(results$transcribed_text, 1, 100)
       text_to_analyze <- results$transcribed_text
       results$text_source <- "transcribed"
     } else {
        if (verbose) message("Transcription completed but resulted in empty or NULL text.")
        results$transcribed_text <- "" # Ensure it's an empty string, not NULL
        results$text_source <- "transcribed_empty"
     }
   }
 } else {
   if (verbose) message("No lyrics provided ('lyrics'=NULL) and transcription not requested ('trans
   results$text_source <- "none"
 }
} else {
  if (verbose) message("Text analysis not requested ('text_classes'=NULL).")
  results$text_source <- "none"
}


# --- 3. Text Analysis (NLI) ---
# Execute only if we have text AND text classes were provided
if (!is.null(text_to_analyze) && nchar(text_to_analyze) > 0 && perform_text_analysis) {
  if (verbose) message(paste("Initiating TEXT analysis with NLI:", nli_model_id, "..."))

  # Call transformer_scores within tryCatch
  text_analysis_result <- tryCatch({
    # transformer_scores expects a vector of texts. Pass lyrics as a single element.
```

```
          transforEmotion::transformer_scores(
              text = text_to_analyze,
              classes = text_classes,
              transformer = nli_model_id
          )
      }, error = function(e) {
           warning("Text analysis failed (transformer_scores): ", e$message, call. = FALSE)
           return(NULL) # Return NULL in case of text analysis error
      })

      # Store the result (could be NULL if tryCatch failed)
      results$text_scores <- text_analysis_result
      if (!is.null(text_analysis_result) && verbose) {
          message("Text analysis completed.")
      }

  } else if (perform_text_analysis && results$text_source %in% c("none", "transcribed_empty", "transcri
      # If text analysis was requested but there was no text to analyze
      if (verbose) message("Text analysis skipped due to lack of textual content (lyrics/transcription)
  }


  # --- Final Return ---
  if (verbose) message("Processing completed.")
  return(results)
}


# ===========================================
#       Plotting Function (Translated)
# ===========================================
library(ggplot2)
```

```
## Warning: pacote 'ggplot2' foi compilado no R versão 4.4.3
```

```
#' Plot Audio and Text Emotion Scores
#'
#' Generates a bar plot comparing emotion scores derived from audio and text analysis.
#'
#' @param results A list object returned by the `song_scores` function.
#'
#' @return A ggplot object representing the bar plot.
#' @export
#'
#' @examples
#' \dontrun{
#' # Assuming 'results' is an object generated by song_scores()
#' # with both audio and text scores:
#' plot_obj <- plot_scores(results)
#' print(plot_obj)
#'
#' # If results only contains audio_scores:
#' results_audio_only <- song_scores(
```

```r
#'   audio_path = "path/to/your/song.mp3",
#'   audio_classes = c("happy", "sad", "neutral"),
#'   text_classes = NULL # Ensure text analysis is skipped
#' )
#' plot_audio_only <- plot_scores(results_audio_only)
#' print(plot_audio_only)
#' }
plot_scores <- function(results) {

  # --- Validate input ---
  if (!is.list(results) || is.null(results$audio_scores)) {
    stop("Input 'results' must be a list containing at least 'audio_scores'.", call. = FALSE)
  }
  if(all(is.na(results$audio_scores))) {
     warning("Audio scores are all NA. Plotting may not be informative.", call. = FALSE)
     # Create empty plot or just return NULL? Let's proceed but it might look weird.
  }


  # --- Prepare Data ---
  # Extract audio scores and classes (handle potential NAs)
  valid_audio_indices <- !is.na(results$audio_scores)
  audio_classes <- names(results$audio_scores)[valid_audio_indices]
  audio_values <- results$audio_scores[valid_audio_indices]

  plot_data_list <- list()

  if(length(audio_values) > 0) {
    plot_data_list$audio <- data.frame(
      class = audio_classes,
      score = audio_values,
      modality = rep("Audio", length(audio_classes))
    )
  }


  # Check if text scores exist and are valid
  # text_scores structure is list(vector), so check [[1]]
  if (!is.null(results$text_scores) &&
      is.list(results$text_scores) &&
      length(results$text_scores) > 0 &&
      !is.null(results$text_scores[[1]]) &&
      !all(is.na(results$text_scores[[1]]))) {

    valid_text_indices <- !is.na(results$text_scores[[1]])
    text_classes <- names(results$text_scores[[1]])[valid_text_indices]
    text_values <- results$text_scores[[1]][valid_text_indices]

     if(length(text_values) > 0) {
       plot_data_list$text <- data.frame(
         class = text_classes,
         score = text_values,
         modality = rep("Text", length(text_classes))
```

```r
        )
      }
  }

  # Combine data if both modalities are present
  if (length(plot_data_list) == 0) {
      stop("No valid scores found in 'results' to plot.", call.=FALSE)
  } else if (length(plot_data_list) == 1) {
      plot_data <- plot_data_list[[1]]
  } else {
      # Check if classes align before binding - might need more sophisticated merging
      # For simplicity now, just bind rows assuming classes might differ or overlap
      plot_data <- do.call(rbind, plot_data_list)
  }

  # Ensure 'class' is a factor for consistent ordering on x-axis
  # Use levels from both audio and text if available
  all_classes <- unique(plot_data$class)
  plot_data$class <- factor(plot_data$class, levels = all_classes)


  # --- Generate Bar Plot ---
  p <- ggplot(plot_data, aes(x = class, y = score, fill = modality)) +
    geom_bar(stat = "identity", position = position_dodge(preserve = 'single'), color = "black") +
    scale_fill_manual(values = c("Audio" = "#4c9a9e", "Text" = "#b3bef2"), name = "Modality") +
    scale_y_continuous(labels = scales::percent_format(accuracy = 1), limits = c(0, 1)) + # Format y-ax
    labs(
         title = "",
         x = "Class",
         y = "Probability"
         ) +
    theme_classic(base_size = 12) + # Slightly larger base font size
    theme(axis.text.x = element_text(angle = 45, hjust = 1), # Rotate x-axis labels if many classes
          plot.title = element_text(hjust = 0.5)) # Center title


  return(p)
}
```

Lyrics Transcription and Song Analysis (audio and lyrics)

```r
library(reticulate)
```

```
## Warning: pacote 'reticulate' foi compilado no R versão 4.4.3
```

```r
use_condaenv("transforEmotion", required = TRUE)

my_song <- "C:/Users/vinic/Downloads/Gostava.mp3"
emo_classes <- c("alegre","neutro","triste")
```

```
results <- song_scores(
  audio_path = my_song,
  audio_classes = emo_classes,
  text_classes = emo_classes,
  transcribe_audio = TRUE,
  asr_model_id = "openai/whisper-large-v3",
  asr_language = "portuguese",
  #lyrics = lyrics,
  start_sec = 14, # Analisar áudio dos 30s aos 60s
  end_sec = 53
)
```

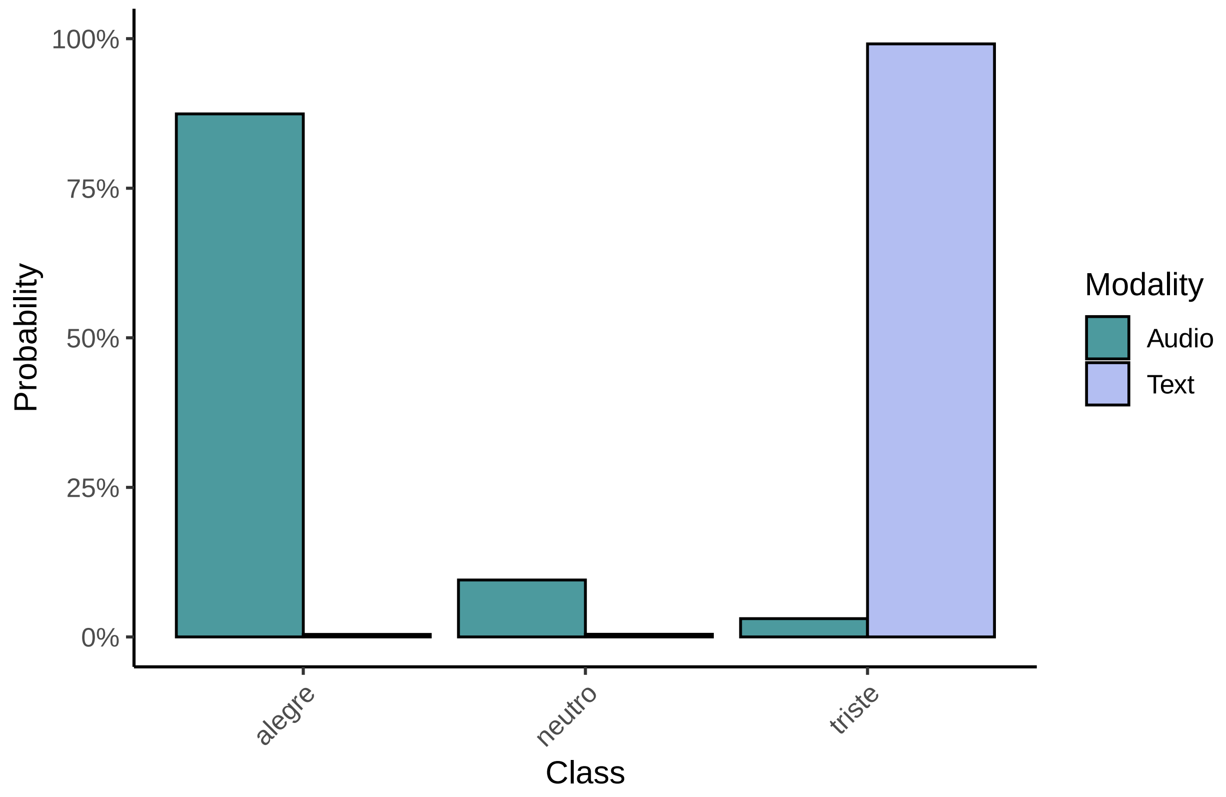
```
## Analysis requested for the segment from 14.00 s to 53.00 s (duration: 39.00 s)

## Importing Python modules (transformers, torch, librosa)...

## Initiating AUDIO analysis with CLAP: laion/clap-htsat-unfused ...

## Audio analysis completed.

## Initiating audio TRANSCRIPTION with ASR (Whisper): openai/whisper-large-v3 ... (This may take time!)

## Transcription completed.

## Transcribed text: Nem sei por que você se foi Quantas saudades eu senti E de tristezas vou viver E a

## Initiating TEXT analysis with NLI: joeddav/xlm-roberta-large-xnli ...

##
## Installing modules for 'transforEmotion'...

## Importing transformers and torch modules...

## Obtaining scores...

## Text analysis completed.

## Processing completed.
```

```
print(results)
```

```
## $audio_scores
##     alegre     neutro     triste
## 0.87424743 0.09516805 0.03058450
##
## $text_scores
## $text_scores$...
##      alegre      neutro      triste
## 0.004230120 0.004449461 0.991320431
```

```
##
##
## $transcribed_text
## [1] "Nem sei por que você se foi Quantas saudades eu senti E de tristezas vou viver E aquele adeus nã
##
## $text_source
## [1] "transcribed"
```

```
plot_scores(results)
```



Analysis of music with provided lyrics

```
# 1. Analisar áudio e usar letra fornecida
lyrics <- "Nem sei por que você se foi, quantas saudades eu senti
E de tristezas vou viver, E aquele adeus não pude dar
Você marcou na minha vida Viveu, morreu na minha história
Chego a ter medo do futuro E da solidão que em minha porta bate
E eu Gostava tanto de você Gostava tanto de você"

results_lyrics <- song_scores(
  audio_path = my_song,
  audio_classes = emo_classes,
  text_classes = emo_classes,
```

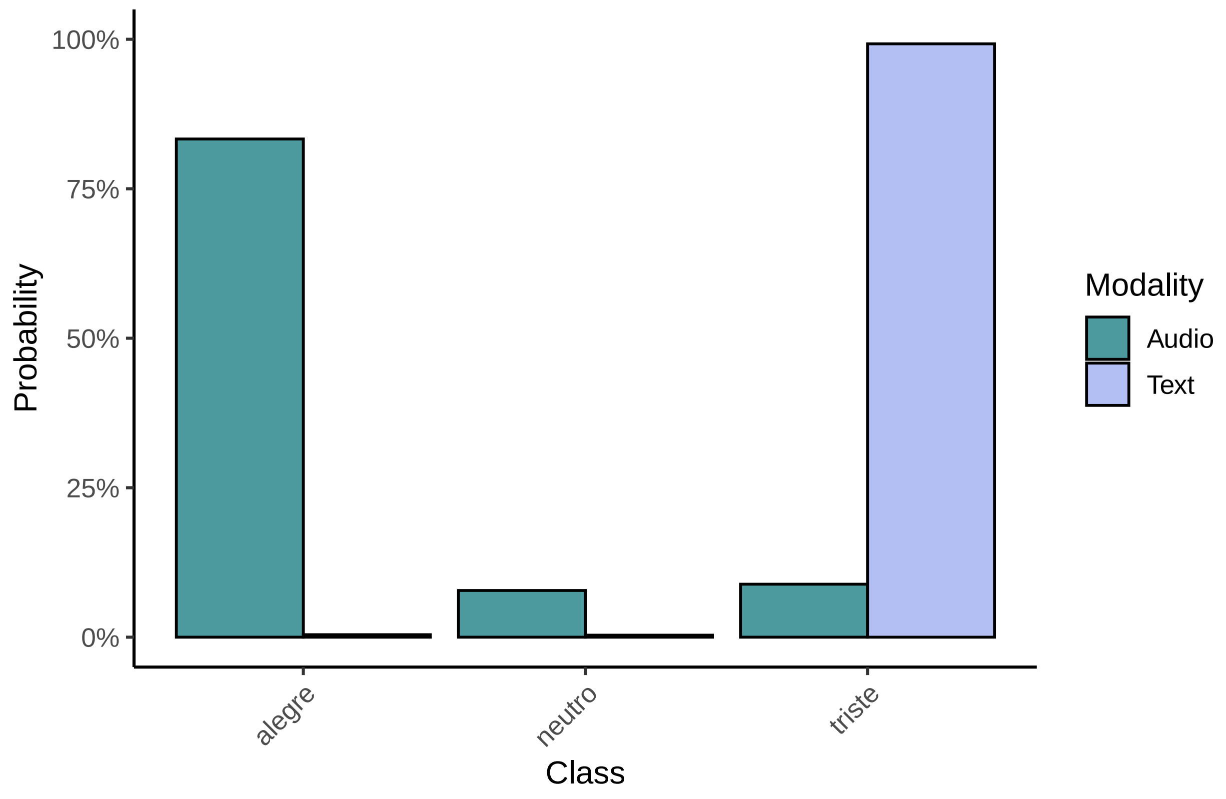
```r
  transcribe_audio = FALSE,
  asr_model_id = "openai/whisper-large-v3",
  asr_language = "portuguese",
  lyrics = lyrics,
  start_sec = 14, # Analisar áudio dos 30s aos 60s
  end_sec = 53
)
```

```
## Analysis requested for the segment from 14.00 s to 53.00 s (duration: 39.00 s)

## Importing Python modules (transformers, torch, librosa)...

## Initiating AUDIO analysis with CLAP: laion/clap-htsat-unfused ...

## Audio analysis completed.

## Using provided lyrics ('lyrics').

## Initiating TEXT analysis with NLI: joeddav/xlm-roberta-large-xnli ...

## Obtaining scores...

## Text analysis completed.

## Processing completed.
```

```r
print(results_lyrics)
```

```
## $audio_scores
##     alegre     neutro     triste
## 0.83329254 0.07808751 0.08861996
##
## $text_scores
## $text_scores$...
##      alegre      neutro      triste
## 0.004181641 0.003377766 0.992440581
##
##
## $transcribed_text
## NULL
##
## $text_source
## [1] "provided_lyrics"
```

```r
plot_scores(results_lyrics)
```



Author: Frederico Pedrosa fredericopedrosa@musica.ufmg.br

Reference

Tomasevic A, Golino H, Christensen A (2024). "Decoding emotion dynamics in videos using dynamic Exploratory Graph Analysis and zero-shot image classification: A simulation and tutorial using the transforEmotion R package."*PsyArXiv*. doi:10.31234/osf.io/hf3g7 https://doi.org/10.31234/osf.io/hf3g7, https://osf.io/preprints/psyarxiv/hf3g7.

Yin, W., Hay, J., & Roth, D. (2019). Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. arXiv preprint arXiv:1909.00161.