Sentiment Recognition for Feedback

December 1, 2025

1 Introduction

In this part a comprehensive and technically detailed explanation of a Python-based application designed to capture live audio, transcribe speech to text using OpenAI's Whisper model, analyze the sentiment of the resulting transcript using the HuggingFace Transformers pipeline, and present the results through a graphical user interface (GUI). The system also logs the sentiment results with timestamps.

The explanation covers the following aspects:

- Description of the machine learning models used (Whisper and DistilBERT).
- Training methodology and architecture of each model.
- Detailed analysis of the source code.
- Explanation of the interplay between audio processing, transcription, NLP analysis, and GUI threading.

2 Models Used

2.1 Whisper Speech-to-Text Model

The Whisper model, developed by OpenAI, is a state-of-the-art automatic speech recognition (ASR) system. It is based on a Transformer encoder-decoder architecture similar to neural machine translation models. The model used in the application is the `base` version, which contains approximately 74 million parameters.

Whisper was trained on 680,000 hours of multilingual and multitask supervised data collected from diverse web sources. This dataset includes speech in multiple languages,

transcripts, background noises, accents, and various environmental contexts, which makes Whisper highly robust.

Architecture

Whisper uses:

- A Transformer encoder that processes log-Mel spectrograms extracted from the audio waveform.
- A Transformer decoder that autoregressively generates text tokens.

The encoder learns audio representations, while the decoder learns to map these representations to natural language text. The model also supports tasks such as language identification and timestamp prediction.

2.2 Sentiment Analysis Model (DistilBERT)

For sentiment classification, the system uses the HuggingFace Transformers `pipeline` with the `sentiment-analysis` task. By default, this pipeline loads the `distilbert-base-uncased-finetuned-` model.

Training Details

DistilBERT is a distilled, smaller version of BERT, trained using knowledge distillation to retain most of BERT's performance while being lighter and faster. The model used here is fine-tuned on the Stanford Sentiment Treebank (SST-2), a well-known dataset for binary sentiment classification.

The model outputs:

- `POSITIVE`
- `NEGATIVE`

with associated confidence probabilities.

3 Code Explanation

This section explains each part of the Python code in detail, covering imports, helper functions, model loading, and GUI construction.

3.1 Imports

```
import sounddevice as sd
import numpy as np
import torch
import whisper
from transformers import pipeline
import threading
import tkinter as tk
from tkinter import ttk, messagebox, font
from datetime import datetime
```

- `sounddevice`: Records audio from the system's microphone.
- `numpy`: Processes audio arrays.
- `torch`: Backend framework used by Whisper.
- `whisper`: Loads and runs the Whisper ASR model.
- `pipeline`: Provides a simplified interface for sentiment analysis.
- `threading`: Prevents the GUI from freezing during audio recording and inference.
- `tkinter`: Creates the graphical interface.
- `datetime`: Generates timestamps for logging.

3.2 Configuration Settings

```
SAMPLE_RATE = 16000
RECORD_SECONDS = 3
LOG_FILE = "feedback_log.txt"
```

- Audio is recorded at 16 kHz, matching Whisper's expected input.
- The program captures exactly 3 seconds of audio.
- Logs are stored in `feedback`$log.txt$.

3.3 Model Loading

```
whisper_model = whisper.load_model("base")
sentiment_analyzer = pipeline("sentiment-analysis")
```

- Loads the Whisper ASR model with the `base` architecture.
- Initializes the sentiment analysis pipeline, which loads DistilBERT.

3.4 Helper Functions

3.4.1 Audio Recording

```
def record_audio(duration=RECORD_SECONDS, sr=SAMPLE_RATE):
    print("Recording...")
    audio = sd.rec(int(duration * sr),
                   samplerate=sr,
                   channels=1,
                   dtype='float32')
    sd.wait()
    return audio.flatten()
```

This function:

- Initiates audio recording for a specified duration.
- Captures mono audio at 16 kHz in 32-bit float format.
- Waits until recording finishes.
- Returns a 1D NumPy array suitable for Whisper.

3.4.2 Speech Transcription

```
def transcribe_audio(audio):
    print("Predicting...")
    result = whisper_model.transcribe(audio, fp16=False)
    return result.get("text", "").strip()
```

The function:

- Calls Whisper to transcribe speech into text.
- Extracts the `text` field from the output dictionary.

3.4.3 Sentiment Analysis

```
def analyze_sentiment(text):
    if not text:
        return "NEUTRAL"

    filler_words = ["hmm", "mm", "uh", "uh-huh",
                    "huh", "ah", "okay", "hmmm"]
    words = text.lower().replace(",", "")
                      .replace(".", "")
                      .strip().split()

    if all(w in filler_words for w in words):
        return "NEUTRAL"

    res = sentiment_analyzer(text)[0]
    return res["label"].upper()
```

Explanation:

- Returns `NEUTRAL` if the text is empty.
- Removes punctuation and converts text to lowercase.
- Checks whether the transcript contains only filler vocalizations.
- If not, passes the text to the sentiment analyzer.
- Returns either `POSITIVE` or `NEGATIVE`.

3.4.4 Logging

```
def log_feedback(text, sentiment):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(LOG_FILE, "a", encoding="utf-8") as f:
        f.write(f"{timestamp} | Sentiment: {sentiment} ")
```

This function appends a timestamped sentiment label to the log file.

3.5 Main Feedback Function

```
def on_button_click():
    audio = record_audio()
    text = transcribe_audio(audio)
    sentiment = analyze_sentiment(text)
    log_feedback(text, sentiment)
    messagebox.showinfo("Detected Sentiment",
                        f"Sentiment: {sentiment}")
```

This is the central workflow:

1. Records audio.
2. Transcribes speech into text.
3. Performs sentiment analysis.
4. Logs results.
5. Displays a popup window with the sentiment classification.

```
root.mainloop()
```

Starts the Tkinter event loop, keeping the GUI responsive.

4 Conclusion

The system integrates modern speech recognition and natural language processing models into an interactive graphical application. Whisper enables high-quality real-time transcription, while DistilBERT provides fast and reliable sentiment classification. Through the use of multithreading, the application maintains responsiveness during computation-heavy tasks. This combination of models and engineering techniques results in a robust, efficient live feedback system.