C Sai Sirisha

Day-1 (12/12/24)

Task Assigned: **Speech-to-Sentiment Detection**

- **Responsibilities:**
    - Implement a model to process user speech and convert it into text (using tools like SpeechRecognition or Whisper).
    - Perform sentiment analysis on the transcribed text to gauge user emotion.
    - Train/test the sentiment model for accurate detection of emotional context.

On day 1 research done on basic concepts involved.

Main steps involved to implement the given task:

1. Pipeline Overview
2. Speech Recognition
3. Sentiment Analysis
4. Real-Time Processing
5. Sentiment Insights for Deal Negotiation
6. Toolkits and Frameworks

1. Pipeline Overview

- Speech Recognition: Convert speech to text.
- Sentiment Analysis: Process the text to determine sentiment.
- Real-Time Processing: Integrate the components to operate in real time.

---

2. Speech Recognition

Use a pre-trained Automatic Speech Recognition (ASR) model to transcribe speech into text:

- Open-Source Tools:
    - Google Speech-to-Text API: Highly accurate for multiple languages.

    - Whisper by OpenAI: Open-source ASR model, supports real-time processing.
    - CMU Sphinx: Free, offline ASR system.
- Integration:
    - Process audio input (via microphone) and output transcriptions in real time.

---

3. Sentiment Analysis

Perform sentiment analysis on the transcribed text:

- Pre-Trained Models:
    - Hugging Face Transformers: Models like BERT or RoBERTa for text sentiment classification.
    - VADER (Valence Aware Dictionary and sEntiment Reasoner): Simple rule-based sentiment analysis tool.
- Custom Model Training:
    - Use labeled datasets (e.g., Sentiment140 or IMDb Reviews) to train a custom sentiment model.

---

4. Real-Time Processing

- Use frameworks like Flask, FastAPI, or Socket.io for real-time interaction.
- Pipeline:
    1. Capture live audio.
    2. Transcribe audio using ASR.
    3. Analyze sentiment of the text.
    4. Visualize results (dashboard or alerts for negotiations).

---

5. Sentiment Insights for Deal Negotiation

- Analyze sentiment trends during conversations.
- Provide insights like:
    - Emotion Trends: Visualize positivity, negativity, and neutrality over time.
    - Key Moments: Highlight sections where sentiment spikes (positive or negative).
    - Actionable Alerts: Trigger recommendations based on sentiment (e.g., if the sentiment is negative, suggest empathetic responses).

---

6. Toolkits and Frameworks

- Speech Processing: pyaudio, wave, or speech_recognition library.
- NLP Tools: transformers, NLTK, spaCy, TextBlob.
- Visualization: Dash, Plotly, or Streamlit for interactive dashboards.

Day 2 (13/12/24**)**

On day 2 research done on the tools that can be used and ways how the task can be implemented.

Step 1: Problem Framing

Combine **speech recognition** and **sentiment analysis** into a pipeline:

1. Speech → Text: Use an ASR (Automatic Speech Recognition) system.
2. Text → Sentiment: Use an NLP model to analyze sentiment.

Step 2: Data Collection

Speech Dataset

- Collect or use an existing dataset with speech and corresponding sentiment labels.
    - **Datasets:**
        - **MELD (Multimodal EmotionLines Dataset):** Emotion and sentiment analysis of dialogue in TV shows.
        - **CREMA-D:** Emotional sentiment dataset with audio.
        - **IEMOCAP:** Emotional speech with both sentiment and emotion labels.

Text Dataset for Sentiment

- Fine-tune your sentiment model on text datasets such as:
    - **Sentiment140** or **IMDb Reviews** for general sentiment classification.
    - Custom datasets tailored to sales or negotiation language (if available).

Step 3: Preprocessing

For Speech:

1. **Audio Features Extraction:**
    - Extract Mel-Frequency Cepstral Coefficients (MFCCs), spectrograms, or log-mel spectrograms.

- Use libraries like librosa or torchaudio.

For Text:

- Clean text (remove stopwords, special characters).
- Tokenize using models like BERT or GPT.

Step 4: Model Architecture

You can build either a **two-stage pipeline** or an **end-to-end model**.

Two-Stage Pipeline

1. **Speech-to-Text Model:**
   - Use a pre-trained ASR system like OpenAI Whisper, Wav2Vec2, or Google Speech-to-Text.
   - Convert audio to text.
2. **Text-to-Sentiment Model:**
   - Fine-tune a pre-trained transformer model (e.g., BERT, RoBERTa) for sentiment classification.

End-to-End Model

Directly predict sentiment from audio:

1. **Input:** Audio features (e.g., MFCCs, spectrograms).
2. **Model:** Use a combination of CNN (for feature extraction) and RNN or Transformer (for temporal context).
3. **Output:** Sentiment label (positive, negative, neutral).

Step 5: Training and Fine-Tuning

1. **Pre-Trained Speech Models:**
   - Use pre-trained speech models like Wav2Vec2 for feature extraction or fine-tuning.
2. **Custom Sentiment Model:**
   - Train or fine-tune sentiment models using your collected data.
3. **Loss Function:**
   - Cross-Entropy Loss for classification tasks.
4. **Optimizer:**

- Adam or AdamW with learning rate scheduling.

Step 6: Real-Time Integration

1. Implement the model using a real-time framework (e.g., Flask, FastAPI).
2. Integrate ASR output with the sentiment analysis model.
3. Optimize for latency by using lightweight models or quantized versions.

Tools and Libraries

- **Speech Recognition:** transformers, torchaudio, or SpeechRecognition.
- **NLP Models:** Hugging Face, spaCy.
- **Deep Learning Frameworks:** PyTorch, TensorFlow, or Keras.

Day 3(14/12/24)

To implement Speech-to-Sentiment Detection using an LLM (Large Language Model), you can integrate speech-to-text processing with sentiment analysis powered by the LLM.

1. Convert Speech to Text

Use tools like Whisper or other speech recognition systems to transcribe audio into text.

2. Perform Sentiment Analysis Using an LLM

Use a pre-trained LLM (e.g., OpenAI GPT-4, Hugging Face Transformers) to perform sentiment analysis on the transcribed text.

Steps:

1. Use the LLM's API or locally fine-tune it for sentiment detection.
2. Prompt the LLM with appropriate instructions to classify the sentiment.

3. Fine-Tune the Model for Domain-Specific Sentiment

While general-purpose LLMs perform well, fine-tuning on domain-specific datasets (e.g., sales conversations) improves accuracy.

Steps:

1. **Dataset Preparation:**
    - Collect labeled data (audio-transcription pairs with sentiment labels like Positive, Negative, Neutral).
    - Example datasets: Customer review datasets, manually labeled sales call transcripts.
2. **Fine-Tuning Process:**
    - For OpenAI GPT: Use fine-tuning endpoints with labeled text data.
    - For Hugging Face: Fine-tune transformer models (e.g., BERT) using frameworks like PyTorch or TensorFlow.

4. Real-Time Integration

To provide real-time sentiment feedback:

1. **Chunk Audio Input:**
    - Break speech into smaller segments for incremental transcription and sentiment updates.
2. **Pipeline:**
    - Transcribe each chunk using Whisper.
    - Analyze sentiment using the LLM or fine-tuned model.
    - Update the sales dashboard dynamically.

5. Evaluation

- Use metrics like accuracy, precision, recall, and F1-score to evaluate sentiment detection.
- Validate the pipeline with test data from real or simulated sales calls.

6. Deployment

- Deploy the pipeline as a microservice using frameworks like Flask or FastAPI.
- Host it in a cloud environment (e.g., AWS, Azure, or Google Cloud) for scalability.

Sample Pipeline Architecture:

1. **Input:** User speech.
2. **Speech-to-Text:** Whisper transcribes speech into text.
3. **Sentiment Analysis:** LLM or fine-tuned model detects sentiment.
4. **Output:** Sentiment labels and real-time insights for negotiation strategies.

This integration leverages LLMs for sophisticated language understanding and aligns well with your project's goals of enhancing sales calls with real-time sentiment-driven insights.

Day 4(15/12/24)

Assignment 1: Integrate LLM Sequence by Sequence

This involves using the LLM to process input text data in a step-by-step manner.

Steps:

1. Define Input: Provide a text sequence as input to the LLM.
2. Call LLM for Analysis: Send the input text to the LLM for processing, such as sentiment detection, summarization, or classification.
3. Handle Output: Use the LLM's response for further processing or display.

Assignment 2: Integrate LLM with VOICE

This involves combining voice input with LLM processing. The steps are:

1. **Speech-to-Text:** Convert the audio input into text using a speech recognition library (e.g., Whisper, Google Speech API).
2. **Text Processing via LLM:** Send the transcribed text to the LLM for analysis or action.
3. **Output Result:** Display or return the processed output.

Assignment 3: Integrate LLM with VOICE and Prompt Engineering

This extends the previous integration with advanced **prompt engineering** to tailor the LLM's output.

Prompt Engineering Tips:

- **Contextual Prompts**: Add context or background information to the prompt for better results.
- **Instruction Clarity**: Be explicit about the task.
- **Formatting Expectations**: Specify the desired format of the output.

Assignment	**Core task**	**Tools/technologies**
Assignment 1	Process text sequences with LLM only.	OpenAI GPT, Hugging Face
Assignment 2	Add voice-to-text conversion for LLM input.	Whisper, Google Speech API
Assignment 3	Integrate voice-to-text and optimize with prompt engineering.	Whisper, OpenAI GPT

Day 5 (16/12/24)

Implementation of assignment 1

1. Prerequisites

Ensure you have:

- An OpenAI API key (or access to another LLM like Hugging Face).
- Python installed with required libraries:
    - `openai` for OpenAI GPT models.
    - Optional: `transformers` for Hugging Face models if you choose that route.

2. Implementation Using OpenAI GPT

Step 1: Import Necessary Libraries

Step 2: Set Up OpenAI API Key

Get your API key from OpenAI and set it.

Step 3: Define a Sequence

The sequence is your text input.

Step 4: Send the Sequence to the LLM

Use a pre-trained LLM to analyze the text

3. Implementation Using Hugging Face Transformers

Step 1: Import Hugging Face Library

Step 2: Load Pre-Trained Sentiment Analysis Model

Step 3: Analyze the Sequence