MLOps Course: Speech Recognition Projects Part 1

Note:

- This is part 1 of the specification for Speech Recognition (SR) projects. This document gives functional requirements that can be used to build the Proof of Concept (PoC) version of the project.
- Part 2 will be released in a week or so and will give more low-level technical details.
- I will create an assignment entry in Moodle to submit the PoC code as a zip file. It will have a deadline of Thursday, 23rd Jan EOD. Note that this is one day earlier than what was mentioned in the class.
- To ensure that students start working on it early, the PoC will have some notional score (5% of the course grade / 20% of project grade).
- Each project description will describe the minimum functionality required in the PoC to get full credit.

Prologue

For each of the projects below, there are three ways to implement them. For the PoC, it is sufficient to build a simple command line system. However, the final project should be implemented using ONE of the following ways:

1. **Browser based:** You can use HuggingFace **gradio** library to build a browser-based UI. You can also use other systems such as streamlit, taipy etc. Basically, you can use an audio input and output. Gradio has lot of great tutorials about how to use gradio audio both one-shot and real-time.
2. **Terminal UI based:** Recently, there has been increasing interest in developing sophisticated TUI that looks like GUI but for terminals. You can use **textual** library for developing sophisticated TUI. You can combine it with rich library to display complex and stylized output.
3. **GUI based:** Python is not that great for GUI development. You can use Tkinter that comes with Python. Or if you are ambitious, you can try PyQt6/PySide6 or any other equivalent library.

Python Libraries

- Speech to Text: I recommend openai-whisper. Other options include vosk, pocketsphinx, kaldi, Mozilla DeepSpeech, Coqui STT etc.
- Text to Speech: I recommend pyttsx3. Other options include gTTS (Google Text-to-Speech) and TTS (Mozilla's Deep Learning TTS).
- Python has a lot of libraries for audio related activities. My recommendation is to use one of sounddevice for controlling microphone. Pyaudio also works well.

Project 1: Basic Voice Command System

In this project, you will build a simple application that can accept some voice commands and perform the corresponding actions. Concretely, your application should be able to do three tasks. For each task, once the action is performed, it should provide auditory feedback using text to speech that the action has been performed. For example, if the voice command is “open browser”, it should open the browser and then say “browser has been opened”. If the system does not understand, then it should say, “I did not understand the command”.

1. Application Control

In the first task, you will select some application (such as a browser) and implement functionality to perform multiple actions triggered by voice command. The application choice is up to you. If you do not have a strong preference, you can use browser as the application. You should support at least 5 voice commands.

If you select browser as the application, some possible actions include “open browser”, “close browser”, “open tab”, “refresh tab”, “go to google”.

Linux has multiple applications that can be programmatically controlled. For example, you can control application such as terminal, media players (VLC, mpv etc), file manager (such as nautilus) etc.

If you are using browser as the application to control, I recommend **playwright-python**. Selenium is an older but equally good alternative.

2. Hardware Control

In the second task, you will do some basic hardware control. For simplicity, let us focus on keyboard, mouse and screen. You need to control all three in the final product.

You will use **pyautogui** for this task which is reasonably cross platform. Here are some common tasks to do:

- Keyboard: Turn on/off capslock
- Mouse: move mouse to top left or bottom right corner of the screen
- Take screenshot of the screen

3. Multi-Step Actions / Robotic Process Automation

In this task, you will do some complex multi-step operation where it is easier to create some kind of wow factor. The multiple steps can be done using a single application or a combination of applications. If you do not have any preference, I would recommend doing some complex

multi-step operation using browser and playwright-python library. You need to support three such as RPA actions.

Using browser as an example, here are some common RPA:

- When a command such as “search google for X” is uttered, open a browser, go to google.com for X, parse the page and print the title of the top-K results.
- Do the same for some e-commerce site such as amazon.in and print the product name, price and other details.
- Other possibilities include logging into Moodle LMS, hotel / flight search, restaurant reservation, job search, etc

Python Libraries:

- See individual tasks for library recommendation

PoC Requirements:

- At least one command from each of the three tasks [application, hardware, RPA] must be done.

Project 2: CEO Analytics Dashboard

In a big enterprise, there is a huge time lag between when a CEO wants X and when he/she gets it. Often, the CEO asks for X, then it goes via big management chain (SVP -> VP -> Director -> Manager -> data scientists). Many startups are building no-code/ low-code platforms that can answer sophisticated analytics queries required by the CEO.

In this project, we will build a toy version of this requirement. We will make the following assumptions.

- The analytics will be done over a single relational table that is stored as a csv file
- All the columns will have meaningful names (e.g. customer id instead of cust_id). This assumption ensures that speech recognition can easily recognize these fields. Otherwise, you have to write some complex rules mapping words to columns.
- We can also assume that the CEO knows the column names.
- You can choose any such table for your purposes – but make it some interesting from an analytics point of view (e.g. transaction table, revenue table etc)

You will be supporting three major tasks.

1. Reports

In this task, the CEO might ask for report A or report B and you will generate a PDF. You can constrain the vocabulary (e.g. give me the sales report / give me revenue report etc). The PDF should look semi-formal and contain logo, tables, charts and some text.

You can use **reportlab** or **weasyprint** to generate the stylized PDF. DO NOT hard code the analytics output. The code should run some realistic Python code on the csv file and generate it. I might vary the contents of CSV file to check if your code is generic enough 😊

2. Charts

In this task, the CEO will ask for some charts. For PoC, you can come up with generic request template (e.g. plot X, plot X vs Y etc). For the final version, you should be able to write a more generic prompts through rules. For example, "plot proportion of revenue by state" should show a pie chart where the plot is partitioned based on state and shows aggregate revenue. Your system should be able to generate – at the minimum – bar, line and pie chart.

You can use gradio's plot component to show any charts. Alternatively, you can generate a plot, save it to a file and then display it.

3. Analytics Queries

You will support some simple aggregate queries (count, average, max, min) with optional filtering/group-by criteria. You can come up with a generic enough vocabulary but they should be sufficiently expressive.

You can use pandas for generating the results of aggregate query but I would encourage you to experiment with polars which is much more elegant and which has already superseded pandas among the cognoscenti.

PoC Requirements:

- You have to at least one action for each of the three tasks.
- For chart/analytics, you can do some simple hard coding for the PoC (e.g. plot X vs Y and show total of X etc).

Project 3: Customer Service Assistant

In the near future, fully autonomous customer service will be interacting with customers. Few years ago, Google Duplex showed the tantalizing possibility. However, the big issue will be training these models. Similar to self-driving cars, there is an emerging effort to introduce AI into customer service workflow. At the beginning, it will only help with some common operations, then it might provide additional context in a non-intrusive manner for e.g. automatically show recent orders of customers when customer says I want to return an item etc., and transcribe the conversation to eventually start replacing the customer service agent. Given the huge economic potential, multiple startups are building such products.

In this project, you will be doing the following basic analysis:

1. Transcribe the call into text
2. Required phrases (greetings, disclaimers, closing statements): check if the agent mentions these things. In many industries such as insurance, it is imperative to ensure that the agent said some standard disclaimers and other compliance related features.
3. Prohibited phrases: profanity and inappropriate language detection
4. PII: ensure that the customer did NOT provide any personal details (e.g. credit card number, ATM PIN, account password etc). You can use some rules to check if this happened.
5. Transcriptions: if there are any profanity or PII, intelligently mask the output (i.e. replace with **** from the transcript. Also output time-stamped markers for compliance phrases. (e.g. the legal disclaimer was told between 15-19th second of the call). This is often required by the legal team.
6. Sentiment analysis.
7. Speaking speed analysis: you do not want the agent to speak too fast.
8. Basic categorization: topic/theme of the call, problem category etc. You can use rules. No need to build a separate classifier. This will be passed to bootstrap other AI.
9. You can perform speaker diarization that determines which of the two users are talking. There are interesting things to be done using this information.
    a. customer to agent speaking ratio (we don't want agent to talk more than customer)
    b. whether agent interrupted a customer (doing once or twice is okay, but if it is done a lot, it is a bad sign)
    c. time between customer saying something and agent responding (this is a VERY important metric called TTFT – time to first token)

Python Libraries:

- NLTK is sufficient for almost all of the requirements (splitting transcription to words, sentiment analysis etc). You can also try other tools such as HuggingFace transformers or spaCy or textblob.
- Speaker diarization: https://github.com/pyannote/pyannote-audio

PoC Requirements:

- You should be able to the first 4 sub-tasks from the list above.

[advanced] Project 4: Language Learning Game

Here be dragons: This is a much more advanced project than the three before. Doing this project well require learning some basic speech recognition related features. Feel free to email the instructor if you have any questions.

The idea is to build a tool for language learning that gives good feedback about the pronunciation. You can select a small set of words and phrases (say roughly 10). For each word/phrase, record 2-3 reference pronunciations (e.g. one per team member). Build a UI where the student will select the word/phrase (e.g. from a dropdown) and then records an audio. Your code should analyze the audio and give feedback about the pronunciation.

Here are some high-level features to experiment with:

- OpenAI transcription: Let's start with a simple heuristic. Pass the audio to OpenAI Whisper and if it returns the same word/phrase, then you can consider it as right pronunciation. This is not elegant, but will serve as a good baseline. In the next step, we will use more sophisticated audio features.
- The next simple feature is phoneme alignment. You can convert the student audio into a list of phonemes and compare it with the pre-processed list of phonemes for the word. Ideally, both should be same. You can use some string similarity metric to define how aligned the two pronunciations are.
- Prosody Analysis: You can compare the pitch/intonation using this method. You convert the audio into a signal and compare it with the reference audio using various methods include DTW (dynamic time warping).
- You can use other features such as pitch similarity, spectra similarity, rhythm match, MFCCs for overall pronunciation accuracy, energy patterns for rhythm and stress.

Python Libraries:

- For this project, it is recommended to use wav file format for storing student and reference audio. More complex file types such as mp3 do complex compression that can mess up your analysis.
- Wav2Vec: Pre-trained model for speech recognition and phoneme detection
- librosa: Audio processing library great for pitch, energy, and rhythm analysis
- Montreal Forced Aligner: For phoneme-level alignment between reference and student audio
- Pratt-Parselmouth for detailed acoustic analysis including formants and pitch

PoC Requirements:

- OpenAI based transcription should be done.