AIML Project Checkpoint Report

Speech-to-Text using HMMs on LibriSpeech

Abhinav Chowdary Bikkina
Mohana

April 14, 2025

1. Introduction

This project explores the task of Automatic Speech Recognition (ASR) using simplified modeling techniques in Python. Rather than using large toolkits like Kaldi, we aim to build an educational ASR pipeline using the `hmmlearn` library on a small subset of the LibriSpeech dataset.

Our goal is to understand the step-by-step construction of a phoneme-level speech recognizer, starting from audio preprocessing, feature extraction, phoneme modeling using Hidden Markov Models (HMMs), and simple decoding. We specifically focus on short utterances and clean audio to keep the setup interpretable and manageable.

2. Objective

The key objectives of this project are:

- Use a small and manageable portion of LibriSpeech data (approximately 1 hour) for training and testing.
- Build a basic phoneme recognition system using Hidden Markov Models (HMMs).
- Use `hmmlearn` for HMM training and decoding.
- Avoid large-scale toolkits like Kaldi for model training to keep the project lightweight.
- Understand the full ASR pipeline and gain intuition for each step from audio to text.

3. Understanding the Research Paper

We are referring to the research paper titled *"LibriSpeech: An ASR Corpus Based on Public Domain Audio Books"* by Vassil Panayotov et al.

Our understanding of the paper is as follows:

- The LibriSpeech corpus was created from public domain audiobooks from LibriVox, containing around 1000 hours of read English speech.
- The authors used the Kaldi toolkit to perform alignment of audio with corresponding transcripts, segment audio into short utterances, and remove low-quality or mismatched data.
- Feature extraction was performed using Mel-Frequency Cepstral Coefficients (MFCCs), and then enhanced with techniques like frame splicing, Linear Discriminant Analysis (LDA), and Semi-Tied Covariance (STC) transformations.
- For modeling, both Gaussian Mixture Model - HMMs (GMM-HMMs) and Deep Neural Networks (DNNs) were trained on Feature-space Maximum Likelihood Linear Regression (fMLLR) adapted features.
- Decoding was done using Weighted Finite State Transducer (WFST) based graphs, which combine lexicon, acoustic models, and n-gram language models to identify the most probable word sequence.

- Models trained on LibriSpeech outperformed those trained on smaller datasets like WSJ, showcasing the benefits of large-scale and diverse training data.

Overall, this paper describes an efficient and scalable industrial-grade ASR system. In our project, we aim to replicate its fundamental logic using simpler tools, enabling a hands-on understanding of each component of the ASR pipeline.

4. Planned Workflow

Our project workflow is designed to mimic a simplified version of the LibriSpeech + Kaldi system using the following steps:

1. **Data Collection**: Select a 1-hour clean subset from LibriSpeech (preferably short utterances from `train-clean-100`).
2. **Preprocessing**:
   - Convert audio files to 16kHz mono WAV format.
   - Tokenize and clean corresponding transcripts.
3. **Feature Extraction**:
   - Extract 13-dimensional MFCC features using `librosa` or `python_speech_features`.
   - Normalize features per utterance.
4. **Text to Phoneme Conversion**:
   - Use `g2p-en` (Grapheme to Phoneme tool) to convert each word in the transcript to its phoneme sequence.
   - Generate a dictionary mapping words to phonemes.
5. **HMM Training**:
   - Use `hmmlearn` to train 3-state HMMs for each phoneme based on MFCC sequences.
   - Store a phoneme-wise model bank.
6. **Decoding**:
   - For a new audio file, extract MFCCs and compute log-likelihoods against trained phoneme models.
   - Use a simple Viterbi decoding scheme to find the most likely phoneme sequence and optionally reconstruct the word using a reverse lexicon.

5. Future Plans and Enhancements

Here are some enhancements that are feasible and aligned with our goals:

- **Language Model Integration (Bigram)**: We plan to integrate a bigram phoneme-level language model to improve decoding. This will help avoid improbable phoneme combinations and improve sequence plausibility.
- **Word-Level HMMs for Isolated Recognition**: We will experiment with training HMMs on entire words for a limited vocabulary. This simplifies decoding and can be useful in command-based applications or keyword spotting.
- **Error Analysis with Kaldi**: We will use a small Kaldi-trained model for reference and compare our decoding results to understand performance gaps and limitations of our minimal system.

These steps will help improve our system's robustness and deepen our understanding of different modeling strategies.

6. Current Progress

So far, the following steps have been completed for the speech recognition pipeline:

6.1. Dataset Download and Extraction

The dataset `train-clean-100` was downloaded from the OpenSLR website using the URL `http://www.openslr.org/resources/12/train-clean-100.tar.gz`. The dataset was then extracted from the downloaded `tar.gz` file to a directory named `train-clean-100`.

6.2. FLAC to WAV Conversion

The dataset consists of `.flac` files, which were converted to `.wav` format for easier processing. A custom function was implemented to handle the conversion, ensuring that the directory structure is preserved during the conversion process. The `flac` files were read using the `SoundFile` library, and the audio data was written to new `.wav` files.

6.3. MFCC Feature Extraction

The next step involved extracting Mel-Frequency Cepstral Coefficients (MFCCs) from the `.wav` files. MFCCs are commonly used features in speech recognition tasks and serve as a compact representation of the speech signal. The `python_speech_features` library was used to compute the MFCCs, with the number of coefficients set to 13, which is typical for speech recognition tasks. MFCC features were extracted for each audio file and stored in a list of dictionaries.

6.4. Data Visualization

To better understand the audio features, both the waveform and the MFCCs were visualized using `matplotlib`. The waveform of the audio was plotted alongside the MFCC spectrogram to give a clear view of the signal's structure and its feature representation.

6.5. Saving MFCC Features

The extracted MFCC features were serialized and saved to a JSON file. This JSON file contains the paths to each `.wav` file and the corresponding MFCC feature vectors. This will serve as the input data for subsequent model training.

6.6. Train/Val/Test Data Splitting

Finally, the dataset of MFCC features was split into three parts: training, validation, and testing. This was done using the `train_test_split` function from the `scikit-learn` library, with 70% of the data allocated for training, and the remaining 30% split equally between validation and testing. The splits were saved into separate JSON files for ease of use in the training process.

The data is now ready for the next stages of the pipeline, including phoneme-related feature extraction and model training.

7. Tools and Libraries

We will use the following tools in our project:

- **Python**
- `librosa`, `scipy.io.wavfile` – Audio loading and resampling
- `python_speech_features` – MFCC extraction
- `hmmlearn` – HMM implementation and training
- `g2p-en` – Grapheme-to-Phoneme conversion
- `numpy`, `pandas` – Data manipulation
- `matplotlib` – Visualization of HMM training and decoding

8. Google Drive Link

`https://drive.google.com/drive/folders/1tN4clMtGKxAG6WftNyuloZo5-ZBEjaeN?usp=drive_link`

9. References

1. Research Paper : `https://www.danielpovey.com/files/2015_icassp_librispeech.pdf`
2. LibriSpeech Dataset: `https://www.openslr.org/12`
3. Kaldi ASR Toolkit: `http://kaldi-asr.org`
4. hmmlearn: `https://hmmlearn.readthedocs.io/`
5. g2p-en: `https://github.com/Kyubyong/g2p`