Speech Understanding (CSL7770)
Assignment 2
Submission Report

Setup -
To setup the environment for performing tasks given in this assignment, a virtual environment can be created. The packages and libraries necessary for all the functions of this assignment are mentioned in the requirements.txt file in the repository. The can be downloaded using the following commands -

Step 1 - Clone this repository into your local machine.

```
> cd <location>
> git clone https://www.github.com/SohamD34/Speech-Understanding-Assignment-2.git
> cd Speech-Understanding-Assignment-2/
```

Step 2 - Create a virtual Python environment.

```
> python3 -m venv b21ee067
```

Step 3 - Activate the environment and install the necessary Python libraries and packages.

```
> source b21ee067/bin/activate
> pip install -r requirements.txt
```

Question 1

In this question, the task is to perform speech enhancement to ensure that the speech of each speaker in a multi-speaker environment is enhanced and can be identified and seperated from the source easily. In this, we first perform speaker identification and verification, speaker separation and then speaker enhancement.

Dataset
For this task, the datasets provided were - ‘VoxCeleb1’ and ‘VoxCeleb2’. Both these datasets are audio-based datasets, containing audio samples (.wav or .mp3 format) associated with certain ids (like classes).

1. **Data ingestion and preparation**
   I first downloaded both the datasets in my working directory from the source (link) using gdown. The contents of the downloaded zip file were extracted using zipfile library and stored at the ‘data/voxceleb1’ and ‘data/voxceleb2’ locations in the working project directory.
   For the speaker verification task, we need to associate pairs of audios which are given in the ‘data/voxceleb_trial_pairs.csv’ file.
   All these steps mentioned above are performed in the ‘1. data_ingestion.py’ file.

2. **Speaker Verification**
   Now that we have the data sources ready, we move on to downloading the pretrained speaker verification WavLM model from this source (link). We store the checkpoint (.pth) file in ‘scripts/Question 1/models/wavlm_base/’ folder location.

We now define the WavLM architecture using WavLMConfig. This creates a model template with uninitialised weights. Now we shall load the checkpoint into our WavLM architecture. Once we have done this, we have a functional WavLM base model.

Now we shall move on to creating a PyTorch dataset to test our model. For this we load the trial pair information from the CSV file stored earlier and load audio files in pairs accordingly. The corresponding labels (0 or 1) denote 'not verified' or 'verified' respectively. The 'SpeakerVerificationDataset' class is custom defined using PyTorch to give a tuple containing (audio_path1, audio_path2, label). The audio files themselves are not loaded to avoid unnecessary memory overhead.

Once the dataset is ready, we now load it into a PyTorch DataLoader with batch size = 128. The loaded model is then evaluated on this dataset. For evaluation on a datapoint, we load both the audio files in that datapoint, extract embeddings of both the audio files using the 'last_hidden_state' of the model. The output of this task is the cosine similarity score between the embeddings of the two audio files.

The metrics used to evaluate the performance in this classification task is EER and TAR @ 1% FAR. The metrics obtained are -
EER (as %) = 42.71%
TAR at 0.01 FAR = 5.73%

All the above operations are done in the '2. Speaker_verification.py'. We can observe that the non-finetuned base model is performing poorly on this speaker verification task. So now, we shall have to finetune it.

3. **Finetuning the model**
   To improve the performance by enhancing the ability of the model to learn latent representations of audio samples, we need to finetune the model. For this we use VoxCeleb2 dataset and perform the LoRA based finetuning.

   We load the non-finetuned, base model checkpoint to initialise a WavLM model instance. Then we shall replace all the linear layers in this model with LoRA matrices (A and B). For this, I have used 'loratorch' packages available at (GitHub link). It repllaces all the linear layers in the architecture with LoRA computation layers.

   Then we shall use this LoRA induced model and train it on the VoxCeleb2 dataset. A problem we can face with audio datasets is that there can be audio samples of varying lengths, which result in variation in the sizes of their latent representations. To ensure uniformity we use a 'collate' function which handles variable length sequences by trimming.

   We load the audio samples dataset into the DataLoader which itself performs the collation task. The conditions used for training/finetuning are mentioned below -

   Learning_rate = 0.0001
   Optimiser - Weighted Adam
   Loss function - Arc Face Loss (scratch implementation)
   Num_epochs = 10

   The finetuned model is then again tested upon the VoxCeleb1 dataset for the verification task. The metrics obtained are as follows -
   EER (as %) = 7.69%

TAR at 0.01 FAR = 70.45%

Thus we can see a notable increase in the True Acceptance Rate (TAR) and a decrease in the Equal Error Rate (EER), compared to the non-finetuned model. The above tasks are performed in the '3. WavLM_finetuning.py' script.

4. **Evaluating Sepformer and combination with Speaker Identification model**
In this, the first part is the evaluation of Sepformer model on separation task on a mixture of sound utterances. For this, we first create the mixture dataset using a combination of pair of audio samples - one from VoxCeleb1 and the other from VoxCeleb2.

We first split the 100 datapoints into 2 sets - training (80) and testing (20). Then in each pair, we choose an overlap region between the two audios and superimpose (by adding) the overlap regions of the two audios.
The combined audio (of increased length) is saved at 'data/mixture/' directory. And a dataset containing information about

- Mixture_id,
- Speaker1,
- Speaker2,
- Utterance1,
- Utterance2,
- Mixture_path,
- Source1_path,
- Source2_path,
- Offset,
- Overlap_length,
- Mixture_length,

is created. This metadata is also stored as a CSV file in the 'data' folder. The above tasks are performed in the 'mixture.py' script.

In order to test the Sepformer model on this mixture data, we now load the mixture dataset. The Sepformer model is loaded from 'speechbrain' library with loaded parameters. The model separated sources and returned vectors representing the two separated audio samples.

The metrics used for evaluation of the separation task are - Signal to Interference Ratio (SIR), Signal to Artefacts Ratio (SAR), Signal to Distortion Ratio (SDR) and Perceptual Evaluation of Speech Quality (PESQ). These metrics are loaded from 'mir_eval' library.

During testing in the test() function, we get the 'pred_s1' and 'pred_s2' source audio vectors. From these two predicted sources, we take the overlap region out and concat the sources. Then we convert these into vector embeddings using the WavLM model we have. The embeddings of the original sources and the embeddings of the predicted source files are then checked for similarity and then labelled whether the separation is correct or not.

This process is repeated with the non-finetined model WavLM model as well as the finetuned WavLM model. The accuracy scored for correct separation are as follows -

Sepformer + Base, non-finetuned WavLM = 0.7154
Sepformer + Finetuned WavLM = 0.7912

Question 2

The task in this question is to perform a comparative analysis of the Indian languages using MFCC and spectrogram based features.

Dataset

The dataset is a Kaggle dataset containing audio files of 10 Indian languages - Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Punjabi, Tamil, Telugu and Urdu.

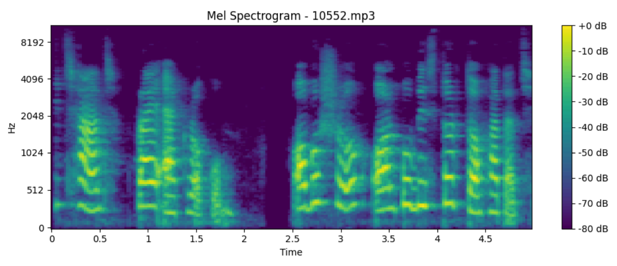
TASK A

1. Data Ingestion
   The dataset is downloaded from Kaggle using the Kagglehub API. It can also be downloaded directly as a ZIP file and later unzipped. The folder is stored at ‘data/audio_dataset/Language Detection Dataset’. It contains 10 folders corresponding to the different languages.

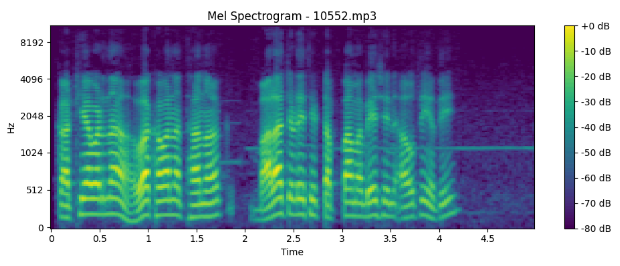
2. Mel Feature Extraction and Spectrograms
   We read all the audio files in each of the language folders and extract 13 Mel Feature Cepstral Coefficients (MFCCs) for all the audio files. For each coefficient, we find out the mean and standard deviation as the features we shall use for our downstream tasks. The language of each audio file is assigned as its label.

   This entire data - the 13 MFCC means, 13 MFCC standard deviations - are used as the input features (X) and the assigned language is the label (Y). This information is stored as a CSV file for ease of use and readability.

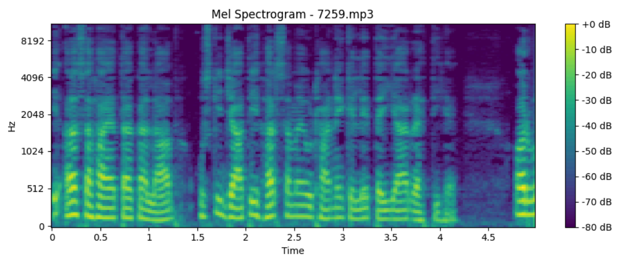
   Similarly all the audio files are converted into the corresponding spectrograms using the ‘librosa’ library. The spectrograms are stored as images in the ‘scripts/Question 2/spectrograms/’ directory. Some of the spectrograms from different languages can be observed below -



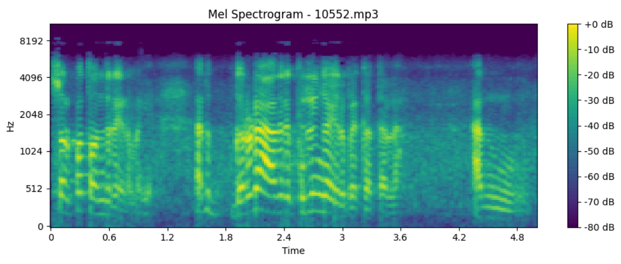
Bengali



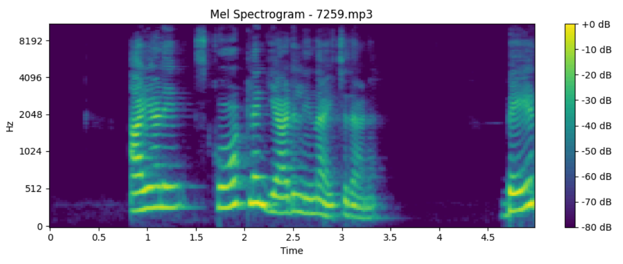
Gujarati



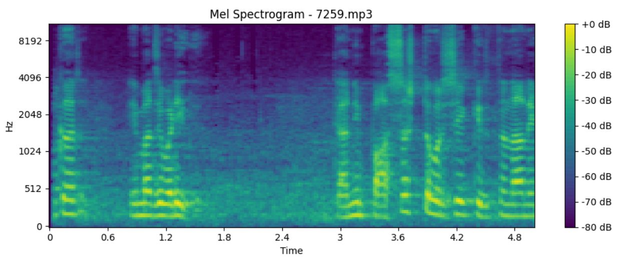
Hindi



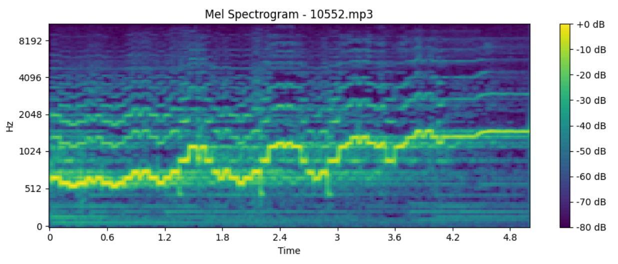
Kannada



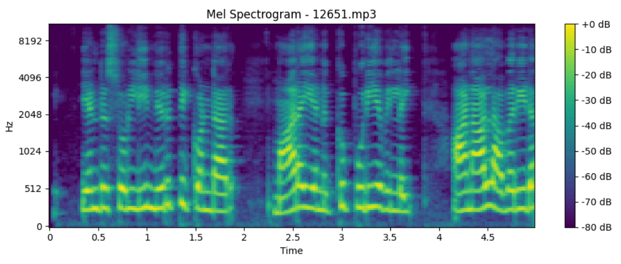
Malayalam



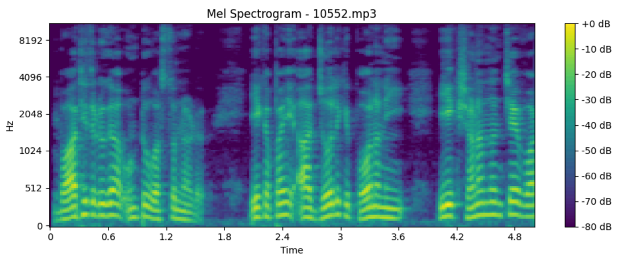
Marathi



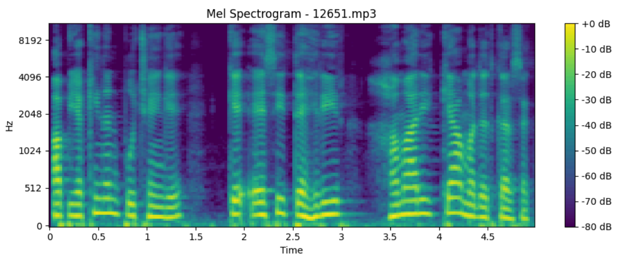
Punjabi



Tamil



Telugu



Urdu

Each language has distinct phonetic and phonological properties, such as:

- Vowel and consonant inventories
- Syllable structure
- Prosody (intonation, stress, rhythm)
- Phoneme durations and transitions

MFCCs reflect these differences indirectly through -

- Spectral Envelope: Different phonemes produce distinct spectral shapes. MFCCs capture these via the shape of the power spectrum.
- Formants: MFCCs roughly encode the formant structure, important for vowel identity.
- Coarticulation Effects: Transition patterns between MFCC frames may reveal language-specific phoneme sequences.
- Phonotactics: Frequency and combination of sounds affect temporal dynamics, visible in MFCC feature distributions over time.

For example, in the above spectrograms, we can see that Tamil and Telugu (languages with similar sounds) share a common vowel and consonant sound inventory. They have a similar syllable structure and rhythms. This can be observed in the similarity in the spectrograms and the similarity of the MFCC features.

On the other hand, Marathi and Punjabi are two entirely different sounding languages with a varied set of syllable structure and prosodic patterns. Hence the spectrograms and MFCC patterns are also quite different.

TASK B - Classification

In this part, we first load the Mel features dataset from the CSV file. For feature preprocessing, we perform Standard Scaling on all the 26 features and Label Encode the language labels.
The data is then split into training and testing sets in the ratio 80:20.

The models used for this task are -
1. Support Vector Machine/Classifier (SVC)
2. Random Forest Classifier
3. K-Nearest Neighbour Algorithm

We perform Grid Search optimization for hyperparameter tuning of all these models. The results for the hyperparameter tuning can be observed at 'logs/question2.txt'.

The best estimator model for each case is then trained and used for evaluation on testing dataset. The training, testing accuracies and the classification reports are as follows.

Support Vector Machine
Best Parameters for SVC: {'C': 1, 'kernel': 'rbf'}
Training Accuracy: 0.8909
Test Accuracy: 0.8647

Class no/ID	Precision	Recall	f1-score	Support
0	0.97	0.98	0.97	5423
1	0.41	0.37	0.39	5373
2	0.97	0.99	0.98	5177
3	0.98	0.97	0.98	4448
4	0.99	0.99	0.99	4758
5	0.98	0.98	0.98	5125
6	0.42	0.46	0.44	5299
7	0.99	0.99	0.99	4800
8	0.99	0.99	0.99	4606
9	0.98	0.98	0.98	6355

Random Forest Classifier (RFC)
Best Parameters for RFC: {'max_depth': 15, 'n_estimators': 50}
Training Accuracy: 0.8953
Test Accuracy: 0.8040

Class no/ID	Precision	Recall	f1-score	Support
0	0.89	0.96	0.92	5423
1	0.22	0.21	0.22	5373
2	0.93	0.98	0.95	5177
3	1.00	0.92	0.95	4448
4	0.99	0.96	0.97	4758
5	0.96	0.95	0.95	5125
6	0.24	0.24	0.24	52999
7	0.98	0.98	0.98	4800
8	0.98	0.95	0.96	4606
9	0.93	0.96	0.94	6355

K-Nearest Neighbour (KNN)
Best Parameters for KNN: {'n_neighbors': 9, 'weights': 'uniform'}
Training Accuracy: 0.8967
Test Accuracy: 0.8340

Class no/ID	Precision	Recall	f1-score	Support
0	0.97	0.97	0.97	5423
1	0.28	0.27	0.27	5373
2	0.97	0.99	0.98	5177
3	0.96	0.97	0.97	4448
4	0.99	0.98	0.99	4758
5	0.98	0.98	0.98	5125
6	0.27	0.27	0.27	5299
7	0.99	0.99	0.99	4800
8	0.99	0.99	0.99	4606
9	0.98	0.98	0.98	6355

References -

1. Mir Eval - https://github.com/mir-evaluation/mir_eval?tab=readme-ov-file
2. Mir_Eval docs - https://mir-eval.readthedocs.io/latest/
3. LoRA-Torch - https://github.com/Baijiong-Lin/LoRA-Torch
4. Sepformer - https://paperswithcode.com/method/sepformer