Automated Speech Recognition Project Report

CS458 Natural Language Processing
Semester Project

Submitted by:
Rafsha Mazhar i170028
Daniyal Hassan i170411
Saad Zahoor i170046

Contents

Project Introduction

Lafz is a speech-to-text conversion tool for the Urdu language, developed as part of our semester project, using the Kaldi toolkit. The program uses Automated Speech Recognition technology and logic to train over a small dataset and then extends its functionality to new, similar audio files, previously untrained on. For now, *Lafz* is limited to listening to pre-recorded wav files and generating corresponding Urdu text, however, it can further be extended to convert speech to text in real time.

Project Motivation

In this time of technology, everything is being moved towards automation. Users' ease of access and usability is paid special attention to. So, in order to increase usability, a lot of systems are adopting Automatic Speech Recognition technologies. This not only gives users an alternative to typing out entire documents but is also of great aid to those who face difficulty in typing due to eyesight disabilities or the lack of limbs. This being said, a lot of work has already been done in this field in the English language.

A number of people are working towards developing accurate ASRs for the Urdu language as well but their progress is nowhere near all the work being done on ASRs in the English language. Additionally, systems of this sort, which use some sort of artificial intelligence and artificial learning techniques can never be fully accurate so there is always room for improvement. For *Lafz*, our aim is to explore the work already done in the field and then put in our best efforts to develop a highly accurate ASR for the Urdu language, building up on our researched work.

Project Team

We are a team of 3 members; Daniyal Hassan, Saad Zahoor and Rafsha Mazhar. Additionally, we sought the help and collaboration of Taha Firoz and Muhammad Saad regarding some issues faced during the development of our model.

Taha provided us with the downsampled version of the dataset we generated as part of our assignment. Getting a pre-downsampled dataset helped us save the time cost of downsampling.

Muhammad Saad helped us get a tarball of Kaldi files at a later stage during the project when we lost our system links due to a problem we faced while building our model. This problem is described in detail under the *Problems Faced during Implementation* section.

Together, with some help and guidance from our collaborators, we were able to successfully implement an ASR in Urdu. As we used the Kaldi toolkit to generate this model, we named our model *Lafz.*

Related Work

Before starting work, we researched thoroughly on Automated Speech Recognition systems, how they work, what toolkits they use, how we can build them and how much work has already been done in this field. We also looked into new projects ASR is being extended to and what the future of ASR may look like. A brief summary of our findings is given below. The rest of the information which we found and used for the implementation of our project is discussed further on, in the report, at the relevant places.

What has already been done in this field?

ASR is not a completely new technology. Work in this field dates back to 1784, when Wolfgang Von Kempelan created the Acoustic Mechanical Speech Machine, in Vienna. From there, with advancements in technology and the increasing atomisation of everything, the pace of development in this field picked up and ASRs are now being widely used in several applications and also come in-built in most of the mobile phones and laptops. In recent years, with the increasing work being done in the fields of AI, the work being done in the field of ASRs has significantly increased. Research in ASR systems has made significant development in the past few years and applications like microsoft translator, google voice search etc are proof of this.

Today, researchers on ASRs have several different choices for building a speech recognition system which include HTK, Julius, Sphinx engine and a Finite State Transducer (FST) based framework Kaldi. CMU Sphinx engine alone has various versions e.g. Sphinx4, Pocketsphinx, etc, which make use of the Hidden Markov models (HMM) to formulate speech recognition problems. Moreover research work has

been done on Kaldi ASR toolkit using Deep Neural Networks (DNN) based acoustic model of continuous speech recognition system for large vocabulary.

What is further being worked on in this field?

Previously the work done on ASR models was mostly based on the training, testing and decoding of the HMM-GMM model. Nowadays, Deep Neural Networks along with HMM, rather than GMM, are being used to improve performance. RBM, Auto Encoder and Deep Belief Networks (DBN) are some of the techniques being used to improve the performance of the system. RBM uses Gaussian distribution and DBN is composed of different layers of RBMs which are stacked together. Moreover work based on Long Short-Term Memory (LSTM) using recurrent neural networks is being done. Using this model, speech recognition is performed using the integration of ASRs and speech enhancement using the LSTM network.

References

We found a lot of useful information in these links and research papers, which helped us better understand the project scope, requirements and development process and hence helped us lay a base to our project.

Automated Speech Recognition Systems

Some links and research papers which helped us set the foundations for our project:

- Basic understanding and architecture of an ASR
  https://www.researchgate.net/publication/337155654_A_Study_on_Automatic_Speech_Recognition

- A collection of resources entitled “Fifty Years of Progress in Speech and Speaker Recognition” - helpful for understanding and reading on the history of ASRs
  https://asa.scitation.org/doi/10.1121/1.4784967

- Evaluation techniques for our system
  https://core.ac.uk/download/pdf/29406959.pdf

- History of ASR
  https://medium.com/swlh/the-past-present-and-future-of-speech-recognition-technology-cf13c179aaf

- http://aghaaliraza.com/publications/2010--O-COCOSDA--An_ASR_System_for_Spontaneous_Urdu_Speech.pdf

- https://www.ijrte.org/wp-content/uploads/papers/v8i2S3/B11080782S319.pdf

- http://www.cs.cmu.edu/~awb/papers/ICNLSSP17_scharenborg.pdf

G2P

Some links and research papers that helped us better understand Seq-to-Seq Grapheme-to-Phoneme (G2P):

- https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43264.pdf
- https://www.aclweb.org/anthology/P16-1038.pdf
- https://www.mdpi.com/2076-3417/9/6/1143/htm

PocketSphinx

Some links and research papers that helped us better understand CMUSphinx:

- https://cmusphinx.github.io/wiki/tutorial/
- https://repository.lib.fit.edu/bitstream/handle/11141/1374/ELJAGMANI-THESIS.pdf?isAllowed=y&sequence=1 (chapters 1 and 2 used)
- https://www.cs.cmu.edu/~awb/papers/ICASSP2006/0100185.pdf

Kaldi

Some links and research papers that helped us better understand the Kaldi toolkit:

- https://publications.idiap.ch/downloads/papers/2012/Povey_ASRU2011_2011.pdf
- https://kaldi-asr.org/doc/index.html
- https://link.springer.com/article/10.1007%2Fs10772-020-09717-8

These articles and research papers gave us enough understanding of the project to enable us to proceed and develop our own approach for the implementation of our own ASR.

Our Approach

Basic Architecture of an ASR

From our research, we found out that the basic architecture of any speech recognition system consists of 3 modules. These modules are a phonetic dictionary, an acoustic model and a language model, all integrated together into one. This can be represented by a diagram:

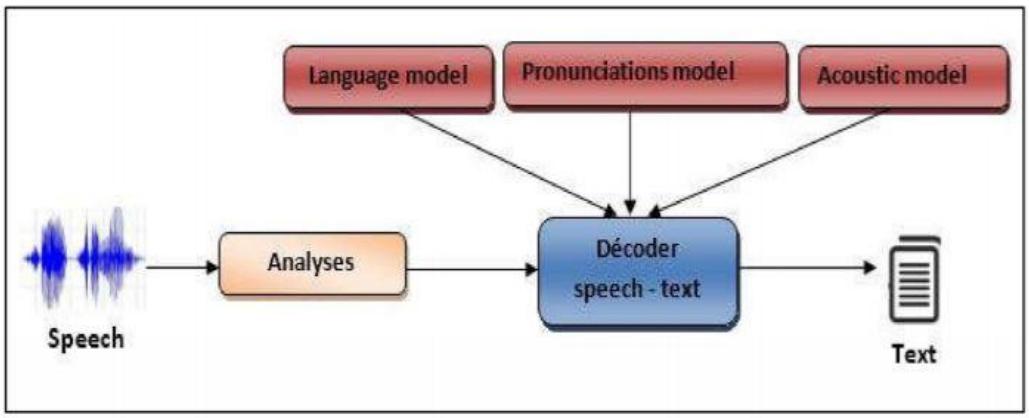

Figure 1. General Architecture of Automatic Speech Recognition Systems

(image taken from research paper already referred to above: https://www.researchgate.net/publication/337155654_A_Study_on_Automatic_Speech_Recognition)

Outline of Our Design

Similar to other ASR projects, we split our project into these 3 modules.

1. **Phonetic Dictionary:** It contains mapping of words to their phones. Initially, it may be trained over a small subset of the entire vocabulary but it must have the ability to extend over to new words present in the vocabulary of the specific language. The ASR uses this to map phones to words and vice versa, hence aiding in conversion of speech to text.

2. **Acoustic Model:** This model deals with the speech part of the ASR. It breaks the audio down into smaller parts and extracts features from it, both, with and without respect to context (monophones, triphones etc). The ASR then maps these to text using the help of the other two modules.
3. **Language Model:** The language model helps recognise patterns of words following other words according to the grammar of the specific language. It identifies the probabilities of various phrases occurring in a sentence or of different words being preceded and followed by other words in a grammatically correct sentence. This helps us limit the words a particular part of an audio can be mapped to, depending on the words preceding and following it. The language model is generally either created in the form of n-gram models or finite state automata.

Before creating these modules, we looked into various toolkits which aid in building speech recognition systems. Most of these toolkits require the 3 modules mentioned above as input and then integrate them together into an ASR system. However, each toolkit requires each module in its own specified format. So, before starting work on the modules, our first task was to research various toolkits and select an optimal one so we could tailor each module according to the required format of the toolkit. We shortlisted CMUSphinx (pocketsphinx) and Kaldi, among these, and eventually decided on taking Kaldi forward.

For the phonetic dictionary, we decided on building it in the CISAMPA format as it is compatible with Kaldi. For the language model and the acoustic model, our major work was to prepare the data, bring it to the correct format and arrange it according to the correct file hierarchy. Other than that, with properly formatted and arranged data, Kaldi automatically creates a language model in the FST format. After all these steps, we only had to train and test the model.

Motivation

Initially we shortlisted CMUSphinx and Kaldi for our speech recognition system. Both of these have been extensively used in the development of ASRs. Reading up on our two shortlisted toolkits, we realised that CMUSphinx did not have very good reviews and users had complained about the results not being as accurate as required, whereas Kaldi had great reviews and feedback. This made us select Kaldi between the two. Kaldi has recipes for building speech recognition systems with widely available databases and has pre trained models publicly released. Moreover Kaldi is a much more efficient model with a relatively lower Word Error Rate (WER) than CMUsphinx.

Implementation Details

Once we had selected our required toolkit and researched further on the toolkit itself, we started the actual coding process. With the help of Allah, the power of dua, never ending research and a lot of hard work, we were able to successfully implement our model in the following steps:

1. **Phonetic Dictionary**
   We started off with the phonetic dictionary. For this, we first needed to set G2P up and bootstrap it with a small dictionary. We, then, needed to train the G2P and extend it to our entire vocabulary.

   a. **G2P setup:** We set up Sequence-to-Sequence Grapheme-to-Phoneme (G2P) using the CISAMPA format on 4000 epochs.

   b. **Training G2P:** To train the G2P, we needed accurate data, mapping Urdu words to their phonemes. We found an Urdu phonetic inventory online (http://www.cle.org.pk/software/ling_resources.htm) from which we extracted words and their corresponding phones and created a file, mapping one word to its phone, in each line, in the form <word> <phone>. This file helped us bootstrap the dictionary for our G2P training.

   c. **Creating Dictionary:** The next step was to develop an accurate phonetic dictionary for Urdu, by extending the g2p training to new Urdu words, hence expanding our vocabulary. After extending the dictionary to our entire vocabulary, we first tested its accuracy in the interactive mode. We were pretty satisfied with the results so we moved on to the next step.

2. **Preparing Speech Corpus**
   The speech corpus we generated as part of our assignment was raw and unprepared. Some students had not submitted their entire part, hence resulting in missing files. We ran a code all over the folder to look for the missing files so we could skip over these files during training and testing. We used the audio files already downsampled and uploaded on the drive by Taha so we did not have to work on the downsampling.

3. **Setting up Kaldi**
   Next, we installed Kaldi and set it up on Google Colab.

4. **Arranging Files**

   Kaldi auto-generates a Language Model in the Final State Transducer (FST) form from the given datasets but in order to get this done, we had to arrange our files according to Kaldi's file hierarchy. Arranging the files not only generates the LM but also prepares the model to be trained on the dataset and eventually give results.

   a. **Research on Kaldi's File Directory:** In order to use Kaldi's tools on our data, we needed to arrange our data files according to Kaldi's file hierarchy. For this, we looked up Kaldi's file hierarchy and studied it to figure out our next action. The file hierarchy we found can be given as:

```
kaldi_b_urdu
|   run.sh
|   run-custom.sh
|   cmd.sh
|   cmd-custom.sh
|   path.sh
|   path-custom.sh
└──audio
|   └──speaker-1
|   |   |   000000.wav
|   |   |   000001.wav
|   |   |   ...
|   └──speaker-2
|   |   |   000000.wav
|   |   |   000001.wav
|   |   |   ...
|   └──speaker-3
|   |   |   000000.wav
|   |   |   000001.wav
|   |   |   ...
└──conf
|   |   decode.config
|   |   mfcc.conf
└──custom_scripts
|   |   mfcc_cmvn.sh
|   |   mmi_sgmm2.sh
|   |   mono.sh
|   |   prep_lm.sh
|   |   sgmm2.sh
|   |   tri1.sh
|   |   tri2.sh
|   |   tri3.sh
└──data
|   └──test
|   |   |   spk2gender
```

```
|   |   |   text
|   |   |   utt2spk
|   |   |   wav.scp
|   └──train
|   |   |   spk2gender
|   |   |   text
|   |   |   utt2spk
|   |   |   wav.scp
|   └──local
|   |   |   corpus.txt
|   |   |   score.sh
|   |   └──dict
|   |   |   |   lexicon.txt
|   |   |   |   nonsilence_phones.txt
|   |   |   |   optional_silence.txt
|   |   |   |   silence_phones.txt
└──steps
|   | <files from steps folder>
└──utils
|   | <files from utils folder>
```

b. **Arranging Data:** Next, we researched on the required formats of the phonetic, acoustic and language data. The next step was to bring data to this format, arrange it according to Kaldi's file hierarchy and make a tarball of the audio files.

5. **Experimenting on Small Scale**
   We decided to extract small parts of our data and test Kaldi on it, side by side with working on writing a script to automatically bring our data into the required form and hence automate the data preparation part. We introduced this step as we figured that testing on a smaller dataset could verify our approach.

   a. **Data Preparation:** With the help of Kaldi's documentation (https://kaldi-asr.org/doc/data_prep.html), we manually prepared all the required sample files at a very small scale i.e around 12 training sentences and 4 test sentences.

   b. **Testing on Small Scale:** Testing the small-scale sample data turned out to be unsuccessful as we were not able to manually save all the required files in the correct, required format. Running Kaldi on these files gave us encoding and format errors so we had to abandon this experiment.

6. **Arranging the Larger Dataset**
   We had already started working on automating the data preparation process alongside the experimentation process. When our experiment failed, we focused fully on this and wrote a script to automate the data preparation process so we could proceed to actually training and testing Kaldi.

7. **Training on the Dataset**
   Once we were done organising the data in the required format, we fed it into Kaldi to automatically generate our language model and acoustic model and to prepare our ASR to be trained.

   The Kaldi toolkit allows deep training according to multiple models and does not allow you to skip through training models to a later, better one as each model builds up on the previous one. These models include Linear Discriminant Analysis - Maximum Likelihood Linear Transformation (LDA - MLLT) and Speaker Adaptive Training (SAT) which can help our model distinguish between speakers and normalise according to them, hence improving the accuracy.

   We have, so far, trained our code on the first four models and are currently extending it to the remaining two, mentioned below, using 90% of the dataset. The results we have so far achieved give a reasonable accuracy which we have discussed in detail, in the *Experimental Evaluation* part of the report. The models we used for training include the Monophone model, the 3 Triphone models, SGMM2 model and the MMI-SGMM2 model.

   a. **Monophones Model:** This trains the acoustic model on individual phones with no respect to context i.e. phones preceding and/or following a particular phone. Monophones alone do not provide a very accurate and reasonable result but this is a necessary step as the later models use files generated by this.

   b. **The 3 Triphones Models:** All 3 of these train the model according to triphones, looking into two surrounding phones, one preceding the phone-in-question and the other following it. Tri1 builds up on the monophone model and in turn provides the prereqs to build tri2 and tri3.

      Tri1(deltas+deltas-deltas) uses 1st and 2nd order derivative approximations on original features to help compute delta features which in turn help compute delta-delta features.

Tri2(LDA+MLLT) builds HMM states and normalises speakers by developing unique transformations for each user.

Tri3(LDA+MLLT+SAT) performs speaker adaptive training and normalises speaker ad noise.

c. **The SGMM2 and the MMI-SGMM2 Model:** These two models can be used to further enhance the accuracy of our trained model, by a huge factor.

8. **Testing**
   Finally we tested our trained model on the remaining 10% of the data. The remaining audio files, which we had not trained our model on, were used for the testing of our model. Result details are discussed at length in the *Results and Analysis* section of the report.

Checkpoints

Training on one model alone takes hours due to the large size of the dataset and considering we trained our model on 6 training models i.e. Monophones, Tri1, Tri2, Tri3, SGMM2 and MMI-SGMM2, training it all over again every time we have to run it is illogical. In order to save ourselves from having to train our model everytime we restart the program, we created checkpoints after each model's training and stored the trained files in our folders.

Testing Interface using PyKaldi

Once the model was set up, trained and tested and we were satisfied with the results, we set up PyKaldi in our model, in order to provide an interface to use our model on new, user-recorded wav files. PyKaldi allows a user to record an audio file in Urdu, at runtime, and then uses our trained model to decode it and translate it to text.

Problems Faced during Implementation

Since Kaldi was a new toolkit for us, we faced several issues which slowed us down during the development of our project. Some of these problems are explained below:

1. **Strict Format of Kaldi**
   Kaldi requires its data files to be in a specific format, which we prepared our data according to, but, while executing the code, we were still getting errors in the format and at times it was because of a mistake as small as an extra whitespace. Kaldi having such a strict structure and format kept us busy over fixing such small errors and wasted a significant amount of our time.

2. **Pinpointing Errors**
   Kaldi does not pinpoint errors well and just gives a very general direction towards the nature of the error. Because of this constraint, for each minor error, we had to look into the scripts, the code and the data files in detail, manually, and figure out the place of the error ourselves.

3. **Error in *path.sh***
   At a point during coding, everything was running fine and all of a sudden it all just terminated, without even giving an error message. This was quite confusing as there wasn't even any error message so we could not even look it up on the internet. After going through the script multiple times, we found that we had "echo"-ed somewhere, out of place, and removing it removed the error. The issue however stands that we did not even receive an error message and had a lot of difficulty pointing out the error, manually.

4. **System Links and Google Drive**
   Kaldi's file hierarchy structure contains system links rather than copies of folders. Since we were working on Google Colab and were accessing all our resources through Google Drive, we had to move the file hierarchy to the drive but doing so broke the system links. Since the system links were no longer there, we kept getting errors related to I/O.

5. **Speed of Copying from Google Drive**
   Since the system links got broken, we tried making tarballs of the audio files to fix the issue but copying a huge collection of data from Google Drive is a somewhat slow process so this slowed us down significantly.

6. **Errors in CISAMPA Phonetic Dictionary**
   When we generated our phonetic dictionary in the CISAMPA format and uploaded it, we found that at some points, the dictionary files had some unknown symbols in them. We were unable to identify the source or the cause of these symbols so we eventually had to remove each of those manually.

7. **CISAMPA Format Transcription**
   The Urdu letter "غ" is mapped to the symbol "7" in some formats and to the symbol "G_G" in others. We used the CISAMPA format for our phonetic dictionary, The dictionary our model generated used "7" for this which was again causing issues as 7, being a digit, was causing the half of the phone (the digit part of it) to be printed right-to-left in the files whereas the other half of the phone was being printed left-to-right as Urdu phones were supposed to. This conflict was generating a warning. Half of phones were being aligned to the right whereas half of them were being aligned to the left. As it turned out, the issue was UTF-8 as digits are part of Urdu as well as English language. This confused the editor to align them to right (like urdu) or to left (like english).

8. **Corrupted File Headers**
   For testing and training our model, we used the dataset we generated as part of our Assignment 3, which consisted of approximately 22k recordings. We planned on using 90% (20k recordings) of this data for training and the remaining 10% (2k recordings) for testing. However, we used the down-sampled version Taha linked us up with. The headers of some of these files got corrupted during the down-sampling process so we were only able to use 17k out of 20k recordings for training and 200 out of 2k recordings for testing. This gave a Word Error Rate (WER) which fluctuated between 19 - 22.

9. **Broken, Incomplete Words**
   The assignment dataset had instances where some sentences were broken down without really ending, at random places, resulting in half-cut or incomplete words at the end of some sentences. Since these broken, incomplete words do not really exist in the Urdu vocabulary, they couldn't be found in our dictionary. This caused our model to ignore a total of 22 words.

10. **Bug Within Kaldi's Official Script**
    Another thing that slowed us down was that we encountered a bug in Kaldi's own official script - *decode.sh* had a variable defined within it named 'local' which had not been exported. In another file, this variable 'local' was being called without using the "$" sign which is invalid in bash. We had to look deeply into the official scripts of Kaldi to fix this so it wasted a significant portion of our time.

11. **Session Time-out Problem during Training**
    We decided to train our model on the six training models already mentioned and each training model took several hours to train on our dataset. However, the

biggest issue we faced regarding this was while we were training on our 5th model, SGMM2. We had to put the model to train multiple times as after 2-3 hours of training, the session always got disconnected due to some reason. This wasted a significant amount of our time and kept us from moving on to training on our final model, MMI-SGMM2.

Experimental Evaluation

For the phonetic dictionary, we trained our G2P model at approximately 4000 epochs. We collected a corpus of around 10k lines and decided to train our model on 70% of these. Since we trained our model on around 7k lines, we had to switch to TPU for faster, optimised results. We, then, tested our phonetic dictionary model on the remaining 3k lines.

For the acoustic model of the ASR, we were given a dataset of around 20-22k recordings, with one sentence per recording. We used 90% of this data i.e. 22k recordings to train our model and used the remaining 10% i.e. 2k recordings to test it. We trained our model at 39 epochs which is the Kaldi standard.

Results and Analysis

Accuracy

Phonetic Dictionary

Testing our phonetic dictionary model gave us an accuracy of about 95%. The accuracy was calculated from an in-built function of G2P.

Complete Model

We trained our dataset on 6 different training models. For comparison, the results obtained from all 6 models are discussed below.

1. **Monophone Training**
   Training our model on the first training model i.e monophones, gave us an average Word Error Rate (WER) of 33.19.. We calculated the accuracy by subtracting the average WER from 100, which gave us an approximate accuracy of 66.81%.

Results can be further studied from the following attached snippet of output of our model. The results also highlight the Sentence Error Rate (SER), insertion, deletion and substitution errors separately.

```
exp/mono/decode/wer_10
-%WER 32.07 [ 10461 / 32624, 701 ins, 2739 del, 7021 sub ] [PARTIAL]
%SER 67.91 [ 1515 / 2231 ]
exp/mono/decode/wer_11
-%WER 31.70 [ 10343 / 32624, 631 ins, 2893 del, 6819 sub ] [PARTIAL]
%SER 66.92 [ 1493 / 2231 ]
exp/mono/decode/wer_12
-%WER 31.71 [ 10344 / 32624, 568 ins, 3056 del, 6720 sub ] [PARTIAL]
%SER 66.34 [ 1480 / 2231 ]
exp/mono/decode/wer_13
-%WER 32.04 [ 10454 / 32624, 521 ins, 3306 del, 6627 sub ] [PARTIAL]
%SER 65.84 [ 1469 / 2231 ]
exp/mono/decode/wer_14
-%WER 32.72 [ 10675 / 32624, 491 ins, 3502 del, 6682 sub ] [PARTIAL]
%SER 66.02 [ 1473 / 2231 ]
exp/mono/decode/wer_15
-%WER 33.34 [ 10876 / 32624, 460 ins, 3675 del, 6741 sub ] [PARTIAL]
%SER 65.80 [ 1468 / 2231 ]
exp/mono/decode/wer_16
-%WER 34.10 [ 11126 / 32624, 429 ins, 3854 del, 6843 sub ] [PARTIAL]
%SER 65.89 [ 1470 / 2231 ]
exp/mono/decode/wer_17
-%WER 34.76 [ 11340 / 32624, 407 ins, 4013 del, 6920 sub ] [PARTIAL]
%SER 65.53 [ 1462 / 2231 ]
exp/mono/decode/wer_7
-%WER 35.74 [ 11660 / 32624, 1110 ins, 2275 del, 8275 sub ]
[PARTIAL]
%SER 72.70 [ 1622 / 2231 ]
exp/mono/decode/wer_8
-%WER 34.13 [ 11135 / 32624, 951 ins, 2423 del, 7761 sub ] [PARTIAL]
%SER 71.04 [ 1585 / 2231 ]
exp/mono/decode/wer_9
-%WER 32.81 [ 10703 / 32624, 808 ins, 2571 del, 7324 sub ] [PARTIAL]
%SER 69.52 [ 1551 / 2231 ]
```

2. **Tri1 (Deltas + Deltas-Deltas) Training**
   Extending our training to the Tri1 training model gave us an average WER of approximately 18.99 and the accuracy was calculated to be roughly 81.01%.

   A snippet from our output at this stage is given below:

```
exp/tri1/decode/wer_10
```

```
-%WER 19.71 [ 6429 / 32624, 1159 ins, 1146 del, 4124 sub ] [PARTIAL]
%SER 60.47 [ 1349 / 2231 ]
exp/tri1/decode/wer_11
-%WER 18.80 [ 6134 / 32624, 1044 ins, 1206 del, 3884 sub ] [PARTIAL]
%SER 58.85 [ 1313 / 2231 ]
exp/tri1/decode/wer_12
-%WER 17.97 [ 5862 / 32624, 943 ins, 1253 del, 3666 sub ] [PARTIAL]
%SER 57.87 [ 1291 / 2231 ]
exp/tri1/decode/wer_13
-%WER 17.43 [ 5686 / 32624, 857 ins, 1297 del, 3532 sub ] [PARTIAL]
%SER 57.24 [ 1277 / 2231 ]
exp/tri1/decode/wer_14
-%WER 16.98 [ 5541 / 32624, 786 ins, 1359 del, 3396 sub ] [PARTIAL]
%SER 56.57 [ 1262 / 2231 ]
exp/tri1/decode/wer_15
-%WER 16.73 [ 5458 / 32624, 722 ins, 1410 del, 3326 sub ] [PARTIAL]
%SER 56.12 [ 1252 / 2231 ]
exp/tri1/decode/wer_16
-%WER 16.59 [ 5411 / 32624, 666 ins, 1463 del, 3282 sub ] [PARTIAL]
%SER 55.85 [ 1246 / 2231 ]
exp/tri1/decode/wer_17
-%WER 16.41 [ 5353 / 32624, 611 ins, 1508 del, 3234 sub ] [PARTIAL]
%SER 55.49 [ 1238 / 2231 ]
exp/tri1/decode/wer_7
-%WER 24.43 [ 7971 / 32624, 1737 ins, 1001 del, 5233 sub ] [PARTIAL]
%SER 65.04 [ 1451 / 2231 ]
exp/tri1/decode/wer_8
-%WER 22.84 [ 7450 / 32624, 1532 ins, 1055 del, 4863 sub ] [PARTIAL]
%SER 63.33 [ 1413 / 2231 ]
exp/tri1/decode/wer_9
-%WER 21.06 [ 6870 / 32624, 1314 ins, 1104 del, 4452 sub ] [PARTIAL]
%SER 61.72 [ 1377 / 2231 ]
```

3. **Tri2(LDA + MLLT) Training**
   Further extending our training to the Tri2 training model gave us an average WER of approximately 19.37 and an accuracy of roughly 80.63%..

   A snippet from our output at this stage is given below:

```
exp/tri2/decode/wer_10
-%WER 20.38 [ 6650 / 32624, 1225 ins, 1142 del, 4283 sub ] [PARTIAL]
%SER 61.95 [ 1382 / 2231 ]
exp/tri2/decode/wer_11
-%WER 19.18 [ 6258 / 32624, 1074 ins, 1179 del, 4005 sub ] [PARTIAL]
%SER 60.20 [ 1343 / 2231 ]
exp/tri2/decode/wer_12
-%WER 18.35 [ 5987 / 32624, 965 ins, 1225 del, 3797 sub ] [PARTIAL]
%SER 58.90 [ 1314 / 2231 ]
exp/tri2/decode/wer_13
```

```
-%WER 17.70 [ 5773 / 32624, 864 ins, 1274 del, 3635 sub ] [PARTIAL]
%SER 57.82 [ 1290 / 2231 ]
exp/tri2/decode/wer_14
-%WER 17.15 [ 5596 / 32624, 795 ins, 1330 del, 3471 sub ] [PARTIAL]
%SER 57.37 [ 1280 / 2231 ]
exp/tri2/decode/wer_15
-%WER 16.86 [ 5501 / 32624, 749 ins, 1377 del, 3375 sub ] [PARTIAL]
%SER 56.70 [ 1265 / 2231 ]
exp/tri2/decode/wer_16
-%WER 16.66 [ 5436 / 32624, 705 ins, 1423 del, 3308 sub ] [PARTIAL]
%SER 56.21 [ 1254 / 2231 ]
exp/tri2/decode/wer_17
-%WER 16.53 [ 5393 / 32624, 664 ins, 1468 del, 3261 sub ] [PARTIAL]
%SER 55.98 [ 1249 / 2231 ]
exp/tri2/decode/wer_7
-%WER 25.10 [ 8189 / 32624, 1759 ins, 1020 del, 5410 sub ] [PARTIAL]
%SER 66.38 [ 1481 / 2231 ]
exp/tri2/decode/wer_8
-%WER 23.44 [ 7648 / 32624, 1567 ins, 1055 del, 5026 sub ] [PARTIAL]
%SER 65.08 [ 1452 / 2231 ]
exp/tri2/decode/wer_9
-%WER 21.72 [ 7087 / 32624, 1388 ins, 1089 del, 4610 sub ] [PARTIAL]
%SER 63.47 [ 1416 / 2231 ]
```

4. **Tri3(LDA + MLLT + SAT) Training**
   Further extending our training to the Tri3 training model gave us an average WER of approximately 20.95. The accuracy was calculated to be roughly 79.05%.

   A snippet from our output at this stage is given below:

```
exp/tri3/decode.si/wer_10
-%WER 21.47 [ 7004 / 32624, 1149 ins, 1329 del, 4526 sub ] [PARTIAL]
%SER 61.68 [ 1376 / 2231 ]
exp/tri3/decode.si/wer_11
-%WER 20.61 [ 6724 / 32624, 1039 ins, 1388 del, 4297 sub ] [PARTIAL]
%SER 60.91 [ 1359 / 2231 ]
exp/tri3/decode.si/wer_12
-%WER 20.10 [ 6559 / 32624, 957 ins, 1436 del, 4166 sub ] [PARTIAL]
%SER 60.56 [ 1351 / 2231 ]
exp/tri3/decode.si/wer_13
-%WER 19.64 [ 6407 / 32624, 898 ins, 1483 del, 4026 sub ] [PARTIAL]
%SER 59.57 [ 1329 / 2231 ]
exp/tri3/decode.si/wer_14
-%WER 19.45 [ 6346 / 32624, 860 ins, 1542 del, 3944 sub ] [PARTIAL]
%SER 59.39 [ 1325 / 2231 ]
exp/tri3/decode.si/wer_15
-%WER 19.25 [ 6281 / 32624, 815 ins, 1606 del, 3860 sub ] [PARTIAL]
```

```
%SER 58.99 [ 1316 / 2231 ]
exp/tri3/decode.si/wer_16
-%WER 19.13 [ 6241 / 32624, 776 ins, 1654 del, 3811 sub ] [PARTIAL]
%SER 58.58 [ 1307 / 2231 ]
exp/tri3/decode.si/wer_17
-%WER 19.07 [ 6223 / 32624, 736 ins, 1707 del, 3780 sub ] [PARTIAL]
%SER 58.63 [ 1308 / 2231 ]
exp/tri3/decode.si/wer_7
-%WER 25.25 [ 8239 / 32624, 1597 ins, 1189 del, 5453 sub ] [PARTIAL]
%SER 64.77 [ 1445 / 2231 ]
exp/tri3/decode.si/wer_8
-%WER 23.89 [ 7795 / 32624, 1441 ins, 1238 del, 5116 sub ] [PARTIAL]
%SER 63.51 [ 1417 / 2231 ]
exp/tri3/decode.si/wer_9
-%WER 22.59 [ 7370 / 32624, 1295 ins, 1287 del, 4788 sub ] [PARTIAL]
%SER 62.30 [ 1390 / 2231 ]
```

5. **SGMM2 and MMI-SGMM2 Training**
   The training on these two models is still under process.

Results of Newly Recorded Sentences

At the very end, we tested our model on some newly recorded sentences, some of them having words which already existed in our dictionary and others having new words which were not present in our dictionary.

The results can be observed below:

Sentence: Islamabad Pakistan ka capital hai aur bahut khubsurat shehr hai.
Result: اسلام آباد پاکستان کا دارالحکومت ہے اور ہیں بہت خوبصورت شہر ہے

Sentence: Mere paas phone nahi hai.
Result: میرے پاس موبائل فون نہیں ہے

Sentence: Ye project Rafsha, Saad aur Daniyal ne mil kar kia.
Result: یہ پروجیکٹ بخشا ساتھ اور داڑھی والے مل کر کیا

Sentence: Bahut shukriya, Khuda Hafiz.
Result: اب وقت شکریہ خدا حافظ

Sentence: Facebook aik jadeed social media platform hai.
Result: فیس بک ایک جدید سوشل میڈیا کا قو وہ شاید س

Sentence: Apne daant ko brush karo..
Result: اپنے داد کو پیش کروں

Analysis

It can be seen that training the model on monophones only gave us an accuracy of 66.81% but the accuracy dramatically increased to 81.01% once we were done training on Tri1. However the slight decrease is accuracy after training our model on Tri2 and Tri3 is slightly confusing for us right now. For now, we cannot say for sure, but we hope that extending the training to SGMM2 and MMI-SGMM2 would overcome this deficiency and increase the accuracy.

The results of newly recorded sentences show that the model is pretty accurate on decoding sentences which use the words which already exist in the dictionary. However, for words which do not exist in the model's dictionary, the accuracy is slightly lower. The model still decodes it to some similar word though, like it decoded the word *'daant'* to *'daad'* since the former didn't exist in the dictionary.

Limitations and Drawbacks

Some limitations and constraints of our model include:

1. For now, our model is limited to generating text against stored wav files and cannot generate text from in-real-time speech.

2. Fetching something from colab or google drive is very slow.

3. Since colab session is only of 12 hours it is not possible to upload the files or train the dataset in one session.

4. We cannot access the dataset from the drive in one attempt since the dataset is huge and the drive keeps timing out.

5. The model is otherwise very accurate but it is not as accurate on words which do not exist in the dictionary.

Conclusion

Achievements

We were able to successfully train our model on four out of the six chosen training models and achieved an accuracy of approximately 79-81%.

Deficiencies

Due to the time constraint and session disconnection issues already discussed in the *Problems Faced during Implementation* section, we weren't able to extend the training to the last two models i.e SGMM2 and MMI-SGMM2. Extending the training to these two can further improve the accuracy significantly.

In addition, including a larger corpus or extending our test and train sets by including the Rumi corpus can further improve this accuracy.

How could we have done better?

Most of the problems we faced during the development of this project revolved around issues regarding time constraint and issues regarding Kaldi's scripts and environment. In the future, we can avoid such issues by firstly, managing our time well from the very start. Secondly, although we did research on Kaldi in depth before starting the project, we need to look into issues others faced using this model and their feedback on our chosen toolkits at an even deeper level, the next time, in order to avoid such problems.

Additionally, as already mentioned, we can still improve our accuracy by either extending our dataset or extending our training to the other models or by doing both.

Future Possible Improvements or Enhancements

After some additional research work and further training, *Lafz* has the tendency to be extended such that it is able to recognise speech in-real-time and convert it to text without the user having to first record the entire sentence; similar to how Google Documents uses the ASR technology for automated, in-real-time typing.