Crowdsourcing for language processing(CLAP)

Rajat Chaturvedi
140050027

May 2, 2018

Guides : Prof. Kameswari Chebrolu and Prof. Preethi Jyothi

1 Introduction

This Project report is for Crowdsourcing for LAnguage Processing Android App, developed by 4th Year CSE IIT, Bombay undergraduate students, Suman Swaroop and Rajat Chaturvedi, under the guidance of Prof. Kameswari Chebrolu and Prof. Preethi Jyothi, as part of their B.Tech Project (BTP II) in the 8th semester.

Crowd-sourcing is a promising method for fast and cheap transcription of large volumes of speech data. It allows a researcher to get a lot of work done with significantly less effort required from an individual, and usually the simplicity of the task allows a layman to contribute in research too. This is especially true for regional Indian languages. All methods for Natural Language Processing require some form of speech along with corresponding text. The unavailability of this data for regional Indian languages makes them very less attractive to the research community. But the presence of large number of speakers of these languages in the country, presents us with an opportunity to make use of Crowd-sourcing to generate the kind of data which will make study of these languages easier.

This app acts as a transcribing tool for large volume of speech data. Apart from this, it has a verification section where users verify and correct speech transcribed by others to reduce errors to the extent possible.Based on the verification and transcribing accuracy of a user, they are given scores.

2 Previous Work

Previously as part of our R&D project, our primary motive was to create an android app to enable users to transcribe speech audio to text in a given language, with substantial accuracy.

For this, the app must enable client to download audio files (mainly .wav files) from the server, play this file to the user, and then allow the user to type and submit the corresponding text in the required language and script. Also, in order to maintain a level of accuracy, there was another function 'Verify' added to the app, which would allow users to verify transcriptions made by other users. We included a primitive scoring system based on edit-distance, which allowed us to get a measure of correctness of transcriptions, as well as giving users a motivation to use the app. Any corrections made by a verifying User also gave us additional data.

For typing in different languages and scripts, we used the Google Indic Keyboard which supports many Indian regional languages along with various ways of typing text.

The server side used the Django-Python framework because of its simplicity and scalability. We used PostgreSQL as our database to store information related to Users, audio files and the texts. The client side could make various calls to the server APIs to fetch data and files, submit data, etc. The server responded to these calls by fetching/modifying data from the database and responding with appropriate HTTP response. Also we used Token Authentication to maintain security as well as login sessions.

In all, the various features supported by the initial version of our app were :

- Signup : Using First name, Last name, username, email and password.
- Login : Which used username/email and password to login. The session is created with an authentication token, which has to be sent along with any further request from this session. Also, the user is able to login only when it has been he/she has been verified/approved by the admin.
- Label : The label tab shows a list of audio files, which can be played on clicking. The player supports replay, pause/resume, change playback speed, etc. That screen also contains a textbox which allows

user to type using the Google Indic Keyboard. It also allows the user to skip a file, which gives a 0 score.

- Verify : This tab contains a list of already labelled audio files. Upon clicking one option, the user gets to a screen where he/she can play original file, view labelled text in an uneditable textbox, and can make any corrections if required in another editable textbox. Also, like Label, it allows user to skip a verification task.
- Profile : This screen allowed users to view their profile details along with total and average scores.

3 Backend

3.1 Introduction

The backend for the Speech To Text App is a Django-Python server. We chose Python because it is easy to write and the version 2.7 supports most of the requirements for our project. Also it comes pre-installed with most Unix and Linux distributions. The Django server is versatile, is one of the best servers available for scalability, handles multithreading, atomic requests, and comes with a handy user-friendly admin interface.

3.2 Database Schema

The database consists of the following tables, which are visible in the Django Admin interface.

- User : The base user model, having attributes 'username', 'email' and 'password'.
- AppUser : This is the model that derives from User model. This is the model that stores details about the user using the app, like 'First Name', 'Last Name'. Other details include 'Conversions' - number of labels that user has submitted, 'Verifications' - number of others' labels that user has verified, and their respective passes, i.e., numbers passed. There is also a 'Score' attribute that stores the sum of all conversions and verifications of the user.
- AudioFile : This model stores the Audio File which is going to be labelled. It contains the name of the file in the database, the language the audio is in, the text if available, as well as the link to the audio file. Other attributes include 'Assigned' - number of users the file is currently assigned to for labelling, 'Conversions' - number of conversions made, number of passes, and 'Verified Conversions' - number of conversions which have been verified by a respectable number of other users.
- AudioFileAppUser - This model is basically a join of AudioFile and AppUser. It is created whenever an AudioFile is assigned to an AppUser for labelling. When labelled, it stores the text and the time of labelling. It also has other attributes like 'Assigned' - number of assigned users (for verification), 'Verifications' and its passes, and the score obtained after labelling. The conversion status corresponds to '0' - Assigned, '1' - Passed, '2' - Converted.
- AudioFileAppUserVerify - This model is a join of AudioFileAppUser and AppUser (VerifyingAppUser), and is created when a labelled audio is allotted to an AppUser for verification. It stores details regarding verfications of existing labels. It contains the verified (or corrected) text by the VerifyingAppUser, as well as the verification status ('0' - Assigned, '1' - Passed, '2' - Correct, '3' - Incorrect/Corrected). It also stores the score for the verification.
- There are supplementary models like LoginLog - stores Login and Logout Logs, Tokens - used for authentication during Login, Groups - stores Groups in which Users may be divided to, Update audio Texts - add text to Audio Files which will then be used to calculate scores, Upload App Users - add a CSV file containing details of new Users, Upload Audio Files - Upload zip file of Audio Files.

3.3 Admin Interface

The admin interface can be found on 'speechtotext.cse.iitb.ac.in:8001', if you are accessing from inside IITB. A new user for admin interface can be created in the django shell. Some additional functions created by us :

- Export CSV : An export csv option for all the tables exists, which creates and outputs a CSV file for the selected rows in the table.

Figure 1: Database Schema

- Reset App User : In the App User table, 'Reset User History' option can be used to reset the details of the selected App Users, which will delete all their labelled files, their verified labels, as well as reset their score.
- Upload App Users : In this model, first we will need to upload a CSV file containing information about the Users that we are going to add. After uploading the CSV, the 'Import AppUsers' option will add the given App Users if their information is valid, and then delete the CSV file.
- Upload Audio Files : In this model, we add new audio files to the server. First, create an empty folder specifically named 'extracted'. In this folder, add whatever '.wav' or '.mp3' files as required. Create specifically a 'zip' of this folder and upload it along with the language. Now use the option 'Upload Audio File' to upload all the audio files.
- Update Audio Text : This model can be used to enter correct texts for existing audio files. Here we need to upload a .txt file containing the name of the audio file and the corresponding text in each row. The 'update text' option is used.

3.4 Some Implementation Details

- PostgreSQL database : For storing the data, we use a PostgreSQL database. The details about this database can be configured in the settings.py file in the project folder.
- Virtual environment : It is always advisable to use a virtual environment while running the server, preferable 'virtualenv' for python.
- Authentication : Authentication for Users is done using the Token Authentication provided by Django. Although this is not exactly Session-based, we try to create sessions based on tokens. Whenever a User logs in, a token is created and stored. For any other views other than Login, the request must have the token in its authentication header. As the User logs out, the token is destroyed.

- Serializers : For queries related to the User (Signup, Login, Logout), serializers are a good way to create a Json and validate users. We use the Serializer class provided by Django. In future, Serializers can be used for other models too. Currently, we have different Responses for different views, so there was no point in creating Serializers for each view. All Serializers are located in 'serializers.py'.
- Scoring : For the files that we already have correct texts for, we use Edit-Distance score, comparing the given text with the correct text. This view can be found in 'utils.py'.

4 Pilot

After completion of our BTP I, a pilot was launched for our app. Using a form floated on GPO, around 141 users in the first phase, and 181 more users in the later phase signed up as volunteers. They were assigned tasks based on their language of preference, choosing from Hindi, Marathi, Telugu, Bengali and Malayalam. Initially, users were required to complete 60 Label and 60 Verify tasks in order to receive a reward - a Microsoft sponsored Tshirt. Later, we increased the number of tasks to 100 and 150 respectively. The increase was necessary because the amount of effort we predicted initially was less.

4.1 Results from the pilot

Number of registered users: 322

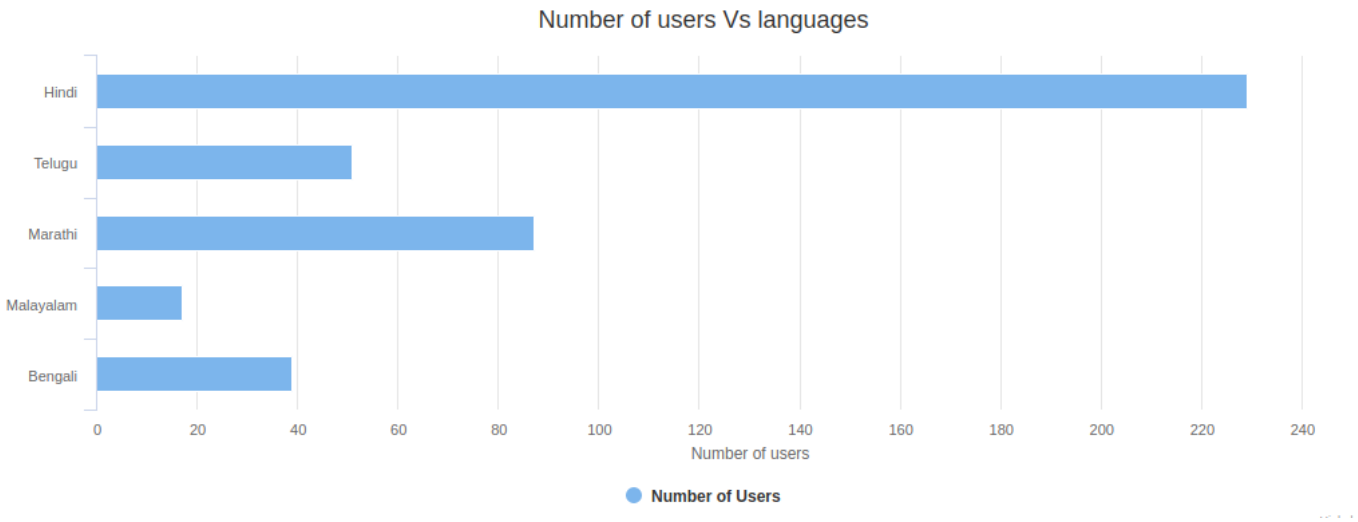

Figure 2: Number of users contributing in each language

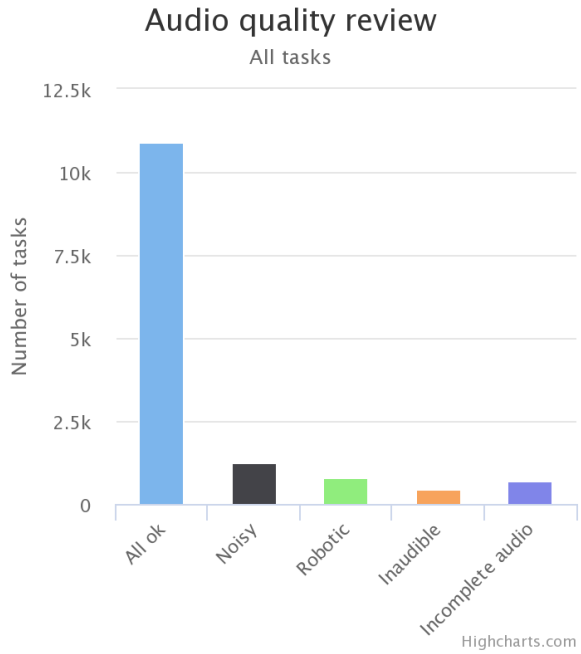

Figure 3: Audio Quality Review

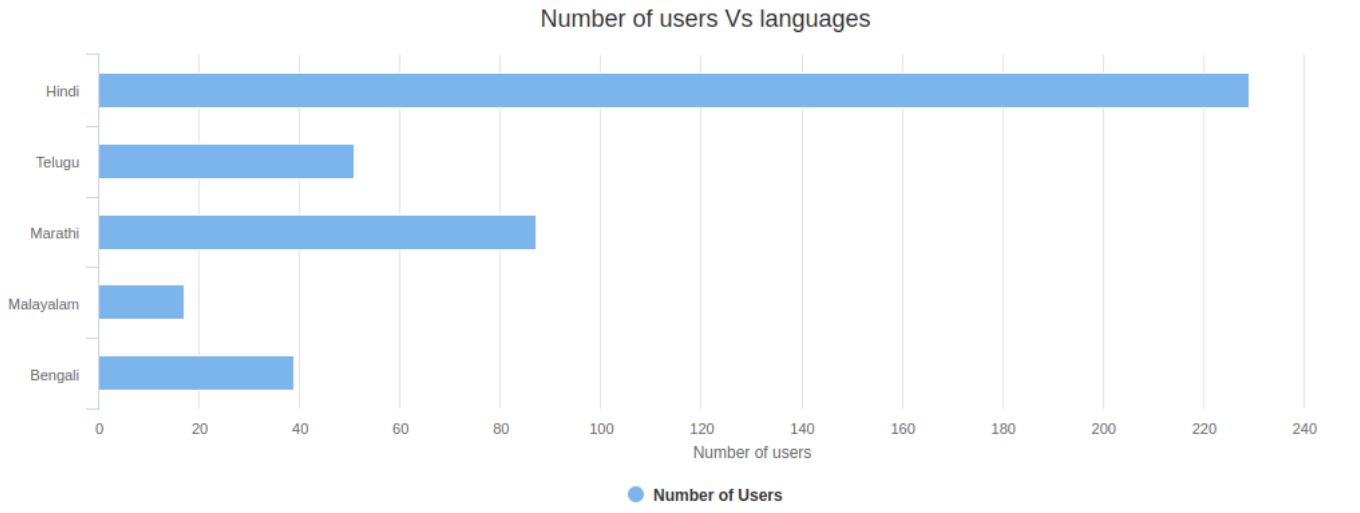

Figure 4: Number of users contributing in each language

5 Evaluation

5.1 Kaldi

In order to evaluate the quality of our data, we decided to use the Kaldi-speech framework to train an Automatic Speech Recognition (ASR) model. We selected Hindi as the primary language, and divided over 18 hours of Hindi data into Train, Test and Validation sets. The Kaldi framework takes speech recordings and texts as input, and trains an HMM-based model on this input. Also, since Hindi is a phonetic language, each sound is mapped uniquely to a phone. This list of all possible phones was also an input to the system. Then the speech from the test data is entered into this model to get the output. This output is compared with the ground truth data to evaluate the quality of the model. The metric used to measure the quality is called the Word-Error Rate (WER). WER gives the percentage of the number of words that were added, subtracted or substituted from the ground truth. The output is generated from an existing vocabulary of words, which in our case, was the list of words from the training data. The model tries to map each and every word from the audio signal to a word in the vocabulary. We had also filtered out those texts from the training data which appeared in the test data, so as to test on new data only.

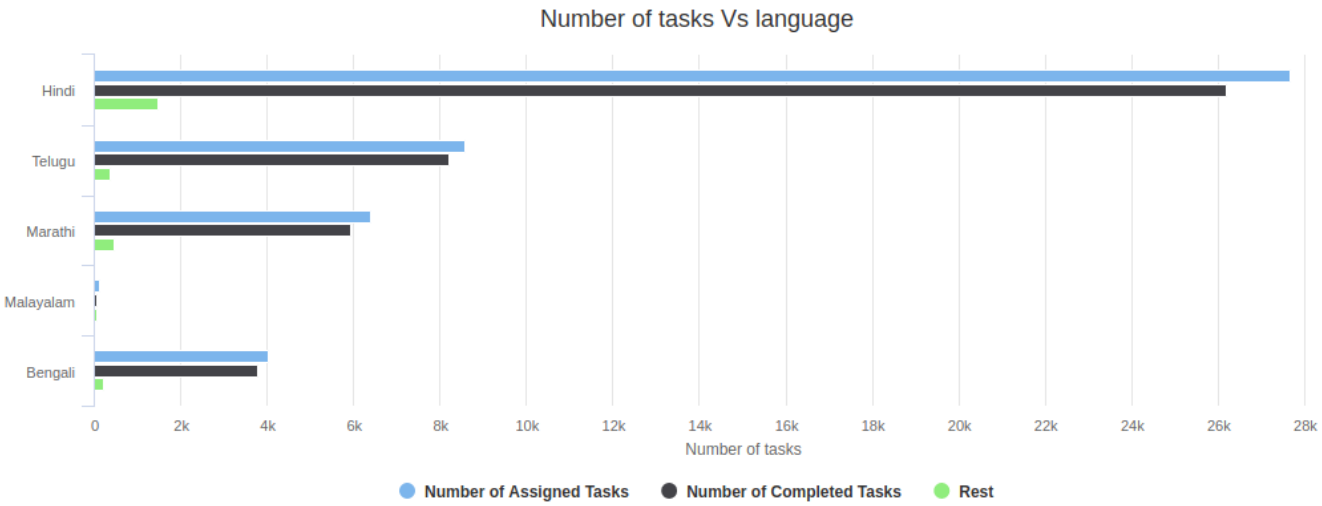

Figure 5: Number of tasks completed in each language

5.1.1 Results

With a basic monophone model, we got a WER of 90%. This was expected since monophone models do not capture any context and hence are prone to errors. Then we used a triphone model, which uses 3 phones for prediction, thus capturing the phones surrounding the current phone. This gave better results than the monophone model, but since this was also a primitive model, the WER was 70%. After tweaking some hyper parameters, such as changing the language model order, we were able to achieve a WER of 50% for the triphone model. After this, we tried to use the Speaker-Adapted Training model, which tries to differentiate each speaker's characteristics from the speech, so that the trained model is not affected by different speakers speaking differently. This model allowed us to get a WER of 22%.

6 Future Ideas

6.0.1 Kaldi - Evaluation

- Kaldi also provides a Deep Neural Net based model, which is expected to perform even better than our current model. We tried to implement this, but ran into some problems with CUDA versions and library issues, and then didn't have enough time.
- We haven't incorporated the information from Verify tasks yet into the model. This can be used to increase confidence in good data and decrease confidence in erroneous data
- Methods to remove errors due to mistranscribed text in the ground truth will also help in getting a better estimate of the quality of the data.

References

[1] *Kaldi ASR*.

[2] *Highcharts for creating graphs*.

[3] *Stackoverflow*.

[4] *Official Django Developer Guide*.

[5] *Django REST Framework*.

[6] IIT Bombay. *POS Tagger*.

[7] Google Developers. *Google Cloud Speech Api*.