UE18CS390B – Capstone Project Phase – 2

SEMESTER - VII

END SEMESTER ASSESSMENT

Project Title : Regional Voice Assistant
Project ID : PW22UD02
Project Guide : Dr. Uma D
Project Team : P Deepak Reddy
Chirag Rudresh
Adithya A S

Outline

- Abstract
- Team Roles and Responsibilities.
- Summary of Requirements and Design (Capstone Phase - 2)
- Design Description
- Modules and Implementation Details
- Project Demonstration and Walkthrough
- Test Plan and Strategy
- Results and Discussion
- Lessons Learnt
- Conclusion and Future Work
- References

Problem Statement

Developing a voice assistant that comprehends multilingual voice navigation queries by converting the audio query to text and translating the textual query into a monolingual sentence.

The novelty approach used in the implementation of this problem statement is the use of a Word Prediction model to reduce the search corpus and improve the accuracy of the multilingual translator DL module.

Abstract

Project Scope

- Taking input as an audio file containing a query sentence with a mixture of languages including English with Kannada.

- Translating the multilingual query to a single language for convenience.

- Feeding the translated sentence to an existing API which will generate appropriate output for the query.

Team Roles and Responsibilities

P Deepak Reddy Contributions:

- Generated more than 100 monolingual sentences.
- Converting mp3 and other forms of audio to wav.
- Study the properties of wav files.
- Understand librosa library and working with it
- Using librosa to visualise the analyse the wav files.
- Worked on splitting the model based on the silence factor.
- Split the wav files based on words manually and implement word similarity based on mfcc features.
- Splitting the wav file after applying smoothing to signal and split at minimas(which is obtained using a moving window method).
- Splitting of sentences to words based on the time window specified by google API.
- Literature survey.
- Research on current speech to text models.
- Research on different feature extraction methods for audio data.
- Study mfcc features in detail.
- Propose the idea of deep learning as a classification problem in speech recognition.
- Implement a deep learning model for a very small set of data.
- Write code for testing models.
- Proposed the idea of using multilingual POS tagging for word predictions to reduce the search space of Deep learning in order to improve accuracy.
- Record individual English words.

Team Roles and Responsibilities

Chirag Rudresh Contributions :

- Started with generating 61 monolingual sentences
- Research on voice assistant queries
- Converted these monolingual sentences to 289 multilingual sentences
- Recorded audio for 92 monolingual sentences
- Recorded audio for 289 multilingual sentences
- Research on speech recognition and translation tools available
- Testing python library for speech recognition
- Testing Top 3 translation APIs
- Tested Google Speech to Text tool
- Added further 67 monolingual sentences to dataset
- Converted these monolingual sentences to 291 multilingual sentences
- Research on speech to text models and methodology
- Testing POS tagging and NTLK toolkit for text processing
- Research on speech to text models and methodology
- Testing POS tagging and NTLK toolkit for text processing
- Built code to recognize language used using Google_trans_new library
- Research on word prediction based on neighbouring words
- Built model to predict word based on neighbouring words trained with English data set
- Built model to predict expected POS tag based on previous tags trained with English dataset
- Built model to predict word based on neighbouring words trained with multilingual dataset
- Research on prediction tag model and grammar correction model
- Testing grammar prediction model
- Testing Phase break detection and word placement prediction models on monolingual textual data.
- Recorded and collected audio multilingual recordings of 36 words in 7 different voice modulations.
- Building HMM model for word prediction
- Adding percentage probability metrics to predicted words in LSTM model

Team Roles and Responsibilities

Adithya AS Contributions :

- Started with generating 61 monolingual sentences
- Converted these monolingual sentences to 289 multilingual sentences
- Recorded audio for 92 monolingual sentences
- Recorded audio for 289 multilingual sentences
- Testing python library to recognise language of individual words
- Converting mp3 and other forms of audio to wav
- Wrote code with two different libraries to visualise the wav file to understand its property
- Worked on splitting of audio based on silence factor
- Built a model using Sequential function
- Tested google,IBM watson APIs to convert speech to text
- Added further 67 monolingual sentences to dataset
- Converted these monolingual sentences to 291 multilingual sentences
- Research on phonemes
- Built 10 different models on phonemes to test which combination gives the best accuracy
- Splitting of sentences to words based on values in the array
- Splitting of sentences to words by sending it to google api
- Recording voices for each individual words for 15 sentences
- Literature Survey
- Preprocessing
- Similarity Model

Summary of Requirements and Design

- Both Audio and textual dataset are required for training.
- The audio data should be in wav format.
- The query should contain word utterances with sufficient gaps.
- The input speech shall be simple command or a query in form of only a single sentence
- The speech query shall be less than 10 sec.
- It is expected that the user pronounces the words accurately.

Summary of Requirements and Design

Python libraries required:

- Numpy,Pandas
- MatplotLib
- Ipython.display
- Librosa
- Seaborn
- Tensorflow→Sequential,Adam,Dense,Dropout,Activation,Flatten
- Pickle
- AudioSegment

Summary of Requirements and Design

Top 5 Speech recognition APIs currently in use:

1. Google Speech API
2. IBM Watson API
3. SpeechAPI
4. Speech to Text API
5. Rev.AI API

Summary of Requirements and Design

Google API working and drawbacks

When the input language is chosen as English, the translator aims to translate the words using the English dictionary vocabulary. Thus, when another language (for example Kannada) is used in the sentence, that foreign word is mapped to the closest sounding English word irrespective of the meaning.

For example: When the input speech is ‘How to go to Shaale’ all the English words are recognised correctly but Kannada word ‘Shaale’ is mapped to the closest sounding English word ‘Charlotte’.

Google Translate API

The input provided for this API was an audio file containing the multilingual sentence and also the list of languages present in the statement.

Expected output was the converted multilingual text from the audio file but however google cloud requires the user to provide their credit card credentials, hence the test case failed to execute. Another drawback for the API to work perfectly is that it is necessary to give the languages present in the sentence which is not ideal in the real-world scenario.

Summary of Requirements and Design

Inbuilt Python Module

For speech to text conversion of the multilingual sentence inbuilt python libraries were used. The output could properly recognise only English words in the given sentence.

```
In [1]: import speech_recognition as sr

In [11]: filename = "54_3.wav"

In [12]: r = sr.Recognizer()

In [13]: with sr.AudioFile(filename) as source:
             # listen for the data (load audio to memory)
             audio_data = r.record(source)
             # recognize (convert from speech to text)
             text = r.recognize_google(audio_data)
             print(text)

MG Road reach Agar ks2 time aguthe
```

Detecting languages present in the multilingual sentence

Python functions were used to detect the languages present in the textual multilingual sentence. This function could properly recognise English words however in most cases failed to recognise regional languages.

Summary of Requirements and Design

Following are the dependencies, constraints,risks and the assumptions attributed to the regional voice assistant :

- Using existing APIs to generate the query's output once it is translated to a single language sentence.
- Ml/NLP algorithms such as character N-grams
- There is an assumption that API will generate an appropriate output to the query.
- Constraint on daily requests the existing API can handle
- It is assumed that words in user query are present in training data.

Design Description

The project consists of 4 modules :

- Splitting of query sentence
- pre-processing
- next-word predictor
- Deep Learning module

Design Description

The approach used for recognition and translation of multilingual audio query is as follows:

- The user query sentence is split into individual wav files which represent each word in the sentence and these chunks are then pre-processed.
- These pre-processed wav files are then passed to a deep learning model that uses a deep learning model to map the audio to text and generate the text output for the corresponding words.
- The accuracy of the speech-to-text model is further increased using a next-word prediction model and a POS tag prediction model. The deep learning model hence classifies each chunk among the next possible words

Design Description

The approach used for recognition and translation of multilingual audio query is as follows:

- The multilingual text query is passed through to Google Translation API to get the corresponding monolingual query which is then passed to the Search Engine to get the appropriate output.

- To make the prediction faster another approach was developed based on similarity of the waveform of the words. However the accuracy of prediction was less compared to deep learning models.

Modules and Implementation Details

Preprocessing:

```
def __init__(self,fileName):
    self.fileName=fileName

def normalise(self):    #Normalising the array
    signal,sample_rate=lib
    rosa.load(self.fileName)
    signal.scaleArray(-1,1)
    return signal
```

```
def trimmed(self):
                signal,sample_rate=librosa.lo
    ad(self.fileName)
    signal.removeSilence(beginning,end)
    return signal
```

```
def speedChange(self):
        signal,sample_rate=librosa.load(self.file
        Name)
        signal.changeFrameRate(targetLen=200
        00)
        return signal
```

Modules and Implementation Details

Splitting of sentence:

The moving average for the signal is calculated and minimas at windows of each 10000 were found out which would be the points at which the sentence will be splitted to chunks where each chunk represents a single word utterance.

def split(fileName)

Window_size = 10000

Signal,sample_rate = librosa.read("path to wav file")

X = Moving_average(winodw_size)

result = split_at_minimas(X)

return result

Modules and Implementation Details

Word Predictor:

def wordPredictor(sentence):

clean = sentence.lower()

tokenize = clean.word_tokenize()

vocab_size = len(tokenizer[train_len])

split()=Tokenizer().fit_text_to_sequen

ces(tokenize)

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
model = Sequential()
model.add(Embedding(vocabulary_size, seq_len, input_length=seq_len))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(50,activation='relu'))
model.add(Dense(vocabulary_size, activation='softmax'))
print(model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 3, 3)              201
_________________________________________________________________
lstm_4 (LSTM)                (None, 3, 50)             10800
_________________________________________________________________
lstm_5 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense_4 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_5 (Dense)              (None, 67)                3417
=================================================================
Total params: 37,168
Trainable params: 37,168
Non-trainable params: 0
_________________________________________________________________
None
```

Modules and Implementation Details

Speech to text:Methodology 1: Deep learning

```
def speech_to_text(audio_query):

    chunks = []
    chunks = split(audio_query)
    chunks.preprocess()
    cur_sentence = "<s>"
   for chunk in chunks:
        classes = word_predictor(cur_sentence)
        next_word = model.predict(chunk,classes)
        cur_sentence = cur_sentence + next_word
   return cur_sentence
```

```
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 100)	4100
activation_24 (Activation)	(None, 100)	0
dropout_18 (Dropout)	(None, 100)	0
dense_25 (Dense)	(None, 200)	20200
activation_25 (Activation)	(None, 200)	0
dropout_19 (Dropout)	(None, 200)	0
dense_26 (Dense)	(None, 100)	20100
activation_26 (Activation)	(None, 100)	0
dropout_20 (Dropout)	(None, 100)	0
dense_27 (Dense)	(None, 5)	505
activation_27 (Activation)	(None, 5)	0

Modules and Implementation Details

Speech to text:Methodology 2: Based on Cosine Similarity

def find_similarity(file1,file2):

```
audio1 = librosa.read(file1)
audio2 = librosa.read(file2)
audio1 = moving_average(audio1, window = 1500)
audio2 = moving_average(audio2, window = 1500)
return cosine_similarity(audio,audio2)
```

def predict(chunk,classes):

```
similarity=[]
for i in trainingData:
    if i in classes:
        similarity.append(find_similarity(i,chunk))
similarity.sort(reverse=True)
similarity=similarity[:20]
predictedWord=mostFrequentClass(similarity)
return predictedWord
```

Project Demonstration

Speech to text:Methodology 1: Deep learning

```
Test Accuracy 0.8148148059844971
WARNING:tensorflow:8 out of the last 9 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001CFB1
CB6558> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. Fo
r (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option tha
t relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/funct
ion#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
Predcited Label [2]
Predcited Class yenu
predicted word =  yenu
predictedSentence =   nanna next turn yenu
```

Project Demonstration

Speech to text:Methodology 2: Based on Cosine Similarity

```
actual_sentence =  nanna next turn yenu
4
[1] [[1]]
classes =  ['what', 'hathira', 'hathiradha', 'are', 'nanna', 'how', 'bheti']
{'nanna': 0.909999698361957, 'are': 0.8684239185365122, 'hathira': 0.8429909782236167, 'how': 0.8398534559567687, 'hathiradha':
0.8204536981585854, 'bheti': 0.772599791514836, 'what': 0.7635959130542181}
Before weights {'nanna': 18, 'are': 2}
After weights, {'nanna': 25.2, 'are': 3.1999999999999997}
Similarity Predicted Word  nanna
Weights Predicted Word  nanna
[1, 21] [[ 1 21]]
classes =  ['hathira', 'next', 'traffic']
{'next': 0.8885499415416608, 'traffic': 0.8581023125049895, 'hathira': 0.5544009349055259}
Before weights {'next': 15, 'traffic': 5}
After weights, {'next': 18.0, 'traffic': 5.0}
Similarity Predicted Word  next
Weights Predicted Word  next
[1, 30, 31] [[ 1 30 31]]
classes =  ['turn', 'places', 'plumbing']
{'turn': 0.9567368809176188, 'plumbing': 0.942401906497795, 'places': 0.8536701122846241}
Before weights {'turn': 13, 'plumbing': 7}
After weights, {'turn': 18.2, 'plumbing': 7.0}
Similarity Predicted Word  turn
Weights Predicted Word  turn
```

Project Demonstration

Speech to text:Methodology 2: Based on Cosine Similarity

```
[1, 30, 31, 45] [[30 31 45]]
classes =  ['yenu', 'saradi', 'open']
{'yenu': 0.9489688363667014, 'saradi': 0.8907351431505826, 'open': 0.8277233043671754}
Before weights {'yenu': 20}
After weights, {'yenu': 28.0}
Similarity Predicted Word  yenu
Weights Predicted Word  yenu
Similarity Predicted Sentence   nanna next turn yenu
Weight Predicted Sentence    nanna next turn yenu
s nanna next turn yenu
```

Walkthrough

Test Plan and Strategy

Splitting:

Sentences were run through the splitting module manually checked to see if they were splitted properly.

Word Predictor:

Each individual word predictor model were checked by running the query and seeing if the results were as expected.

DL Model:

Ratio of number of correctly predicted words to total number of words for each sentence was calculated and average of the entire process was also found. Model accuracy for each sentence was also noted.

Test Plan and Strategy

Similarity Model:

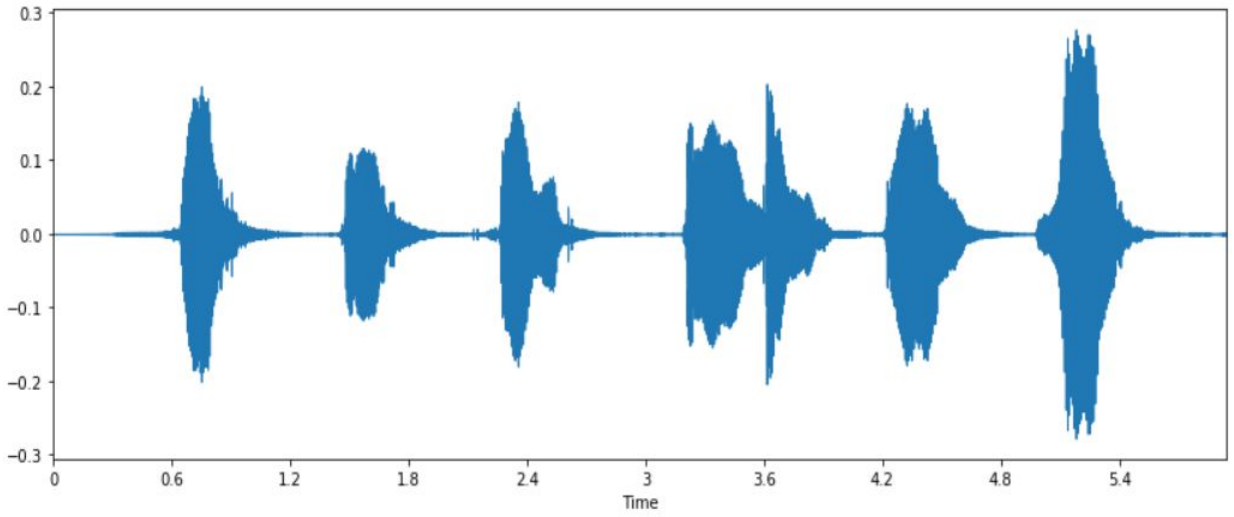
Matching from top 20: Frequency of top 20 classes were chosen and highest among the frequencies was chosen as the predicted word. Strategy similar to DL model was used to find accuracies.

Finding highest similarity: Average of similarity for each class was found and highest average was taken as the predicted word.Accuracy was found by using the same strategy as mentioned in the other two methods.

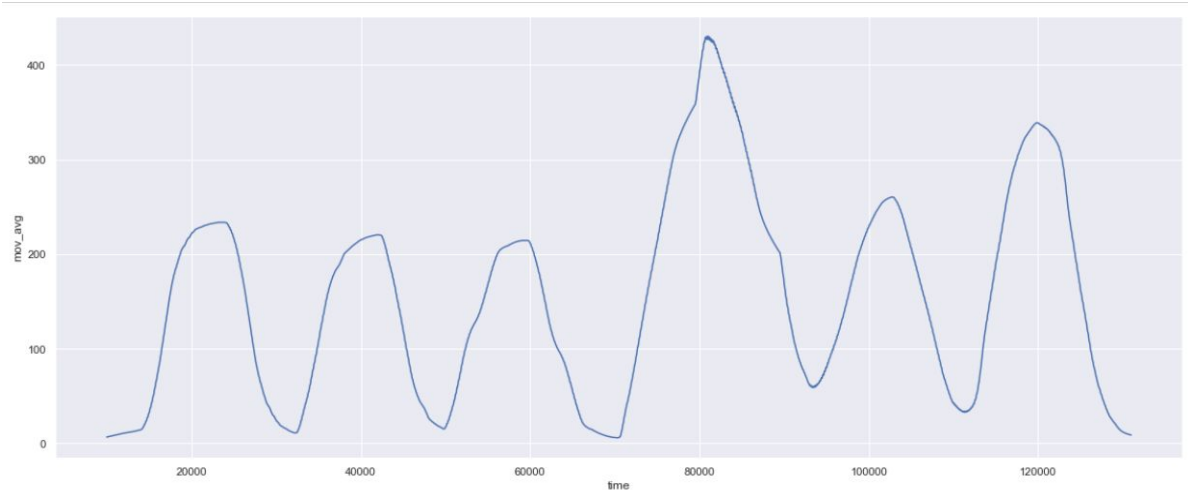
Results and Discussion

Splitting of sentence

This is the visualization of a wav file for the recording of the sentence “how to hang painting on gode”.



After smoothing, the minimas are defined clearly and hence split at these points:



Results and Discussion

Splitting module:

	Splitting Module	
Tested Sentence	**Actual Number of Words**	**Predicted Number of Words**
nanna next turn yenu	4	4
what is my next saradi	5	5
what is my mundina saradi	5	5
how is the datthane ahead	5	5
munde traffic hegidhe	3	3
how is the datthane munde	5	5
what are the nearby anila stations	6	6
what are the nearby gas kendragalu	6	6

The splitting algorithm was tested on 30 sentences out of which 28 were correctly splitted into respective words as shown in the table

Results and Discussion

Word Prediction:

```
model.fit(train_inputs,train_targets,epochs=500,verbose=1)
model.save("modelOne.h5")
```

```
Epoch 491/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2042 - accuracy: 0.8899
Epoch 492/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2360 - accuracy: 0.8300
Epoch 493/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2174 - accuracy: 0.8719
Epoch 494/500
4/4 [==============================] - 0s 5ms/step - loss: 0.2401 - accuracy: 0.8510
Epoch 495/500
4/4 [==============================] - 0s 5ms/step - loss: 0.2281 - accuracy: 0.8731
Epoch 496/500
4/4 [==============================] - 0s 5ms/step - loss: 0.1995 - accuracy: 0.8876
Epoch 497/500
4/4 [==============================] - 0s 5ms/step - loss: 0.2151 - accuracy: 0.8846
Epoch 498/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2099 - accuracy: 0.8877
Epoch 499/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2298 - accuracy: 0.8676
Epoch 500/500
4/4 [==============================] - 0s 4ms/step - loss: 0.2033 - accuracy: 0.9055
```

```
In [9]: from keras.preprocessing.sequence import pad_sequences
input_text = input().strip().lower()
encoded_text = tokenizer.texts_to_sequences([input_text])[0]
pad_encoded = pad_sequences([encoded_text], maxlen=seq_len, truncating='pre')
print(encoded_text, pad_encoded)
for i in (model.predict(pad_encoded)[0]).argsort()[-3:][::-1]:
  pred_word = tokenizer.index_word[i]
  print("Next word suggestion:",pred_word)
```

```
nanna
[24] [[ 0  0 24]]
Next word suggestion: places
Next word suggestion: mundina
Next word suggestion: next
```

Word Predictor model gave 90% accuracy and when predicted top 5 possible next words for a current sentence, the desired word was present in this predicted possible words.

Results and Discussion

Speech to text:

Methodology 1: Deep learning

DL Module		
Actual Sentence	**Predicted Sentence**	**Average Accuracy**
nanna next turn yenu	nanna next turn yenu	0.83
what is my next saradi	what is my next saradi	0.71
what is my mundina saradi	what aagothe any stalagalu saradi	0.68
how is the datthane ahead	how service the yaavaaga ahead	0.73
munde traffic hegidhe	munde traffic hegidhe	0.78
how is the datthane munde	how service the yaavaaga ahead	0.79
what are the nearby anila stations	what does the nearby anila stations	0.84
what are the nearby gas kendragalu	what aagothe the nearby gas kendragalu	0.84
what are the nearby anila kendragalu	what aagothe the nearby good kendragalu	0.85

Model Accuracy	0.85
Prediction Accuracy	0.71

Results and Discussion

Speech to text:

Methodology 2: Based on Cosine Similarity

Actual Sentence	Predicted Sentence(using average similarity of each class)	Accuracy	Predicted Sentence(using maximum count among top 20 most similar recordings)	Accuracy
nanna next turn yenu	nanna next turn yenu	1	nanna next turn yenu	1
what is my next saradi	what is any next yenu	0.6	how is my next saradi	0.8
what is my mundina saradi	are is any mundina yenu	0.4	what is my mundina yenu	0.8
how is the datthane ahead	what is some yaavaaga ahead	0.4	what aagothe my nearest ahead	0.2
munde traffic hegidhe	are traffic hegidhe	0.67	are traffic hegidhe	0.67

The average accuracy of 0.59 was achieved when prediction was done using average similarity of each class and 0.64 was achieved when using the highest occurring class among the top 20 most similar recordings.

Documentation

Github Repository Link:

https://github.com/adithya-1/Capstone-Project.git

IEEE conferences:

- International Workshop on Statistical Methods and Artificial
  Mar 22, 2022 - Mar 25, 2022
  Porto
  Deadline - 20 December, 2021
- 2022 6th International Conference on Data Mining, Communications and Information Technology (DMCIT 2022)
  Apr 28, 2022 - Apr 30, 2022
  Shanghai, China
  Deadline - Dec 30, 2021
- Springer Journal Special Issue on Data Science for Next-Generation Recommender Systems
  Deadline - Dec 31, 2021

Lessons Learnt

We have made a good understanding on audio processing and important elements of machine learning in audio domain.

What we could have done differently knowing what we know now:

- Instead of the deep learning model classifying each word, we could have tried to make the deep learning model classify each syllable.
- Similarity of the audio files using spectrograms can be explored.

Give an overview of issues that has been overcome in this project.

- Created our own data-set since there did not exist any ready-made one.
- There existing models for splitting an audio sentence based on words were not effective for splitting the words in our dataset. Hence we have come up with our own splitting algorithm.

Conclusion

Model	Accuracy
CMU Sphinx(HMM model trained and tested with our data)	57%
Similarity measure using Neural Network	64%
Google Translate(Kannada words recognition)	35.4%
Deep learning model(using MFCC features)	71%
Deep learning model(using MFCC features, for kannada words recognition)	66.6%

Future work

We can see our novelty method is able to perform to a good standard, but there are a few improvements that can be added which could further increase the accuracy, these are:

- Increasing the data set, both audio and textual, with a variation in voice such as age, gender, noise level, etc.
- The similarity approach can be improved using similarity index comparison of the spectrograms of the words, instead of the previously mentioned method which compares the wav-forms.
- The prediction of words and their parts of speech in a particular sentence by leveraging different and more useful language features.

References

[1] Multilingual Speech Recognition with a single end-to-end model - Shubham Toshniwal, 15th February 2018

[2] Multilingual Text-to-Speech Software Component for Dynamic Language Identification and Voice Switching ,September 2016 Paul Fogarassy,Costin Pribeanu

[3] Automatic Language Identification Using Deep Neural Networks,2016,Ignacio Lopez-Moreno, Javier Gonzalez -Dominguez, Oldrich Plch

Thank You