International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 09 | Sep 2019 www.irjet.net p-ISSN: 2395-0072

AUDIO DATA SUMMARIZATION SYSTEM USING NATURAL LANGUAGE PROCESSING

Pravin Khandare[1], Sanket Gaikwad[2], Aditya Kukade[3], Rohit Panicker[4], Swaraj Thamke[5]

[1]Assistant Professor, Dept. of Computer Engineering, Vishwakarma Institute of Technology, Pune, Maharashtra (India)
[2,3,4,5]UG Student, Vishwakarma Institute of Technology, Pune, Maharashtra(India)

Abstract - *This paper presents techniques for converting speech audio file to text file and text summarization on the text file. For the former case, we have used Python modules to convert the audio files to text format. For the latter case, Natural Language Processing's modules are used for text summarization. A Python toolkit named SpaCy is used for the English data functions. Summarization method involves important sentence obtained when the extraction is investigated. Weights are assigned to words according to the number of occurrences of each word in the text file. This technique is used for producing summaries from the main audio file.*

Key Words: NLTK (Natural Language Toolkit), Stop words, Frequency Table, Frequency Distribution, Tokenization, Stemming, Lemmatization, SpaCy.

1. INTRODUCTION

The most natural and effective mode of communication between human beings is available in audio.

Although, it is an easier way to perceive the information, there are some disadvantages associated with this type of communication. In case of audios that are available today, like podcasts, broadcasting of news on radio channels, speeches, etc. all the audio files data that are obtained from these sources are not effectively useful means of gathering information every time. In this case, it is always effective if the data is obtained from the condensed content i.e. focusing more on important points in the content rather than the entire content. To proceed with this technique, an audio file is taken as input and data summarization processes are implemented to get condensed output.

This paper particularly focuses on a two-level procedure for getting summarized output from input audio file. Namely, Speech to Text and then text summarization. The efficiency of the output though depends on the clarity of the audio file that is given as input to the former stage. Human speech recording has proved to be a clear and processable input, whereas pre-recorded podcasts have some amount of deviation in the original text and the text generated in the first phase.

There is an optional output available for the users that are getting an audio summary of the recording. This has been kept as a user preference whether they want the audio-based summary or not. All the summarized outputs are available in the form of files. Also, the original audio and text in the form of output files are maintained, in case the user wants to manually add a sentence from the text to the summary.

2. PHASE 1: SPEECH TO TEXT

2.1 Obtaining input

The input audio for processing is obtained by recording using the device's microphone. The recording process is facilitated by GUI based buttons. The recording starts when the button "Start Recording" is pressed and stops on the "Stop recording" button click event. The python modules used for GUI creation is tkinter. The audio recording and processing are done with the help of pyaudio module. Audio is recorded by continuously appending frames of audio. The final output of this phase is in the form of an audio file of the "wav" format. The format wav is chosen since it is suitable for further processing.

2.2 Need of partition

The audio file recorded needs to be converted into text. This process uses python module SpeechRecognition. This module requires a working internet connection to function as it uses Google's web-based speech to text engine. The time duration for which the engine accepts input is a maximum of 1 minute. So, if the duration of the audio file increases beyond 1 minute then we need to divide it into chunks of smaller duration.

The process of dividing the audio file in chunks of processable duration is carried out by using mathematical

formulae by calculating the size of the audio file by considering frame rate of audio file, the original duration of the audio file and the bit rate and then dividing the size of the audio file by the file split size. We then get the total number of numbered chunks and in wav format as well. The python module required for getting the audio file parameters is pydub.

The mathematical formulae used for the same are:

wav_file_size = (sample_rate * bit_rate * channel_count * duration_in_sec) / 8 …….. (1)

file_split_size = 1000…….. (2)

total_chunks = wav_file_size // file_split_size…... (3)

chunk_length_in_sec = math.ceil((duration_in_sec * 10000000) /wav_file_size) ……(4)

chunk_length_ms=chunk_length_in_sec*1000…... (5)

chunks = make_chunks(myaudio, chunk_length_ms)

……. (6)

The individual chunks of files are then processed further for getting the final text output of the original audio file. This text will then be processed for getting summary.

2.3 Getting the text output

The function of generating text from the audio file recorded is aided by the python module Speech Recognition. The reason for selecting this specific module is due to its simplicity of usage and better efficiency than offline modules. The audio file or chunk of audio file is passed to an instance of the module that takes this wav file as source. The audio file is scanned and rectified for noise. The module uses Google's engine to recognize the audio and convert it into text file. This file stores the original text of the speech and is stored for operational purposes. This file is then processed under phase two to get the extractive summary of the text recorded in this file.

3. PHASE 2: TEXT SUMMARIZATION

This phase focuses on generating a text summary as an output for the input text that is generated in phase 1. This phase mentions the major steps taken in the algorithm of summarization. It uses SpaCy which is an open-source software library for advanced Natural Language Processing, written in the programming languages Python and Cython.

3.1 Tokenization of words and word frequency generation

At this point, the data has to be tokenized by breaking it down into words to generate the frequency of each word occurring in the provided data. After the list of tokens is generated, iterate through the list and check if the corresponding word is not present in the stop words list and if not, increase its frequency by 1.

```
In [10]: mytokens
Out[10]: ['What',
          'is',
          'a',
          'personal',
          'calling',
          '?',
          'It',
          'is',
          'God',
          "'s",
          'blessing',
          ',',
          'it',
          'is',
          'the',
          'path',
          'that',
          'God',
          'chose',
```

3.2 Weighted Frequency and Frequency Table Generation

At this point having generated the frequencies of each word in the given data, normalize the frequencies by extracting the maximum of all frequencies which was generated in the earlier step and divide frequency of each word by the maximum frequency.

```
In [15]: word_frequencies
Out[15]: {'What': 0.058823529411764705,
          'personal': 0.17647058823529413,
          'calling': 0.17647058823529413,
          '?': 0.11764705882352941,
          'It': 0.058823529411764705,
          'God': 0.11764705882352941,
          "'s": 0.11764705882352941,
          'blessing': 0.058823529411764705,
          ',': 1.0,
          'path': 0.23529411764705882,
          'chose': 0.058823529411764705,
          'Earth': 0.058823529411764705,
          '.': 0.8823529411764706,
          'Whenever': 0.058823529411764705,
          'fills': 0.058823529411764705,
          'enthusiasm': 0.058823529411764705,
          'following': 0.058823529411764705,
          'legend': 0.058823529411764705,
          'However': 0.058823529411764705,
          "n't": 0.17647058823529413,
          'courage': 0.11764705882352941,
          'confront': 0.058823529411764705,
          'dream': 0.23529411764705882,
          'Why': 0.058823529411764705,
          '\n': 0.11764705882352941,
          'There': 0.11764705882352941,
          'obstacles': 0.058823529411764705,
          'First': 0.058823529411764705,
          'told': 0.058823529411764705,
          'childhood': 0.058823529411764705,
          'onwards': 0.058823529411764705,
          'want': 0.20411764705882354,
          'impossible': 0.058823529411764705,
```

3.3 Sentence Tokenization

Now there is a need to generate scores of each sentence in the data to generate an optimal summary of the given data. For that first tokenize each sentence in the data.

3.4 Sentence scoring

Now, having generated a list of sentences and also a list of words. So, generate a score for each sentence by adding the weighted frequencies of the word that occur in a particular sentence. As it is not interested in a summary, it has to score only those sentences with less than 30 words.

3.5 Sentence Selection

In the earlier step, output is generated as dictionary of sentences with their scores. Now, select the top N sentences by passing our sentence scores dictionary along with its values and the required N sentences to the n-largest function in the NLTK library and it turns into a list with sentences according to their sentence scores.

3.6 Joining the sentences

Further, convert the sentences to strings and then Finally Join them to generate the finalized summary.

3.7 Getting audio output

For this part of the program, use of gTTS module is done. It stands for Google Text to Speech. It is a web-based engine for processing an input text file and generating output in audio format. The language needs to be specified for the purpose. In this case, English language is chosen and the input text file is the previously generated summarized text output file. After processing on the given parameters, the gTTS module gives output in the form of an mp3 file. This file can be played on a simple button click event. Hence the user satisfaction is taken care of in the end.

3.8 Final Set of Outputs

Hence the final set of outputs include:

a. Main recorded audio file.

b. Chunks of the above file, if needed.

c. Text file of recorded audio.

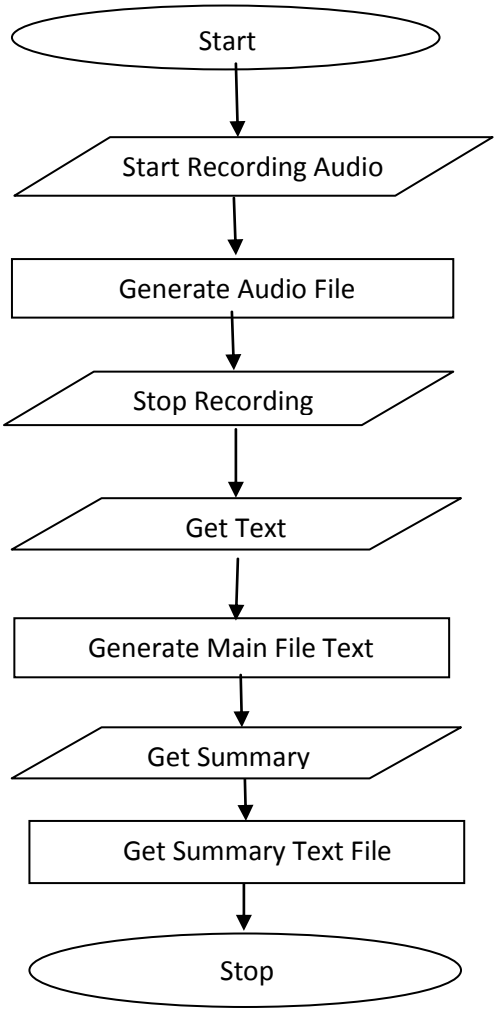
d. Summarized text output file.

e. Summarized audio file.

4. CONCLUSION

In the process of generating a summary as discussed in this paper, the primary input taken for processing is an audio file. The audio file is generated by recording the human speech which is being spoken or is already recorded. The duration of the recording is completely controlled by the user in the form of a buttoned GUI. The audio file in wav format is then converted into a text file which is then used as input for text summarizer processing. The final output of the project is the summarized text file of the contents of the primary recording.

FLOWCHART



REFERENCES

1. Partha Mukherjee, Soumen Santra, Subhajit Bhowmick, Development of GUI for Text-to-Speech Recognition using Natural Language Processing

2. Sadaoki Furui, Fellow, IEEE, Tomonori Kikuchi, Yousuke Shinnaka, and Chiori Hori, Member, IEEE, Speech-to-Text and Speech-to-Speech Summarization of Spontaneous Speech

3. Rajesh S.Prasad, Dr. U.V.Kulkarni, Machine Learning in Evolving Connectionist Text Summarizer

4. M. Saadeq Rafieee, Somayeh Jafari,"Considerations to Spoken Language Recognition for Text-to-Speech Applications"

5. Guangbing Yang, Erkki Sutinen,"Chunking and Extracting Text Content for Mobile Learning: A Query-focused Summarizer Based on Relevance Language Model"

6. Yihong Gong ,Xin Liu,"Creating generic text summaries"

7. Yogita H. Ghadage, Sushama D. Shelke," Speech to text conversion for multilingual languages"

8. Abhijit V. Bapat, Lalit K. Nagalkar, "Phonetic Speech Analysis for Speech to Text Conversion"

9. R. Carlson, G. Fant, C. Gobl, B. Granstrom, I. Karlsson, Q.-G. Lin, "Voice source rules for text-to-speech synthesis"

10. F. Violaro, O. Boeffard, "A hybrid model for text-to-speech synthesis"