IJIRR

International Journal of Information Research and Review
Vol. 03, Issue, 05, pp. 2395-2399, May, 2016

Research Article

SPEECH TO TEXT CONVERSION FOR CHEMICAL ENTITIES

***Farhaan Kaleem, Shruti Kanchan, Pradnya Kalbhor, Aditya Kakde and Sonali Patil**

Department of Computer Engineering, Savitribai Phule Pune University, Pune, Maharashtra, India

ARTICLE INFO

Article History:

Received 19th February 2016
Received in revised form
22nd March 2016
Accepted 13th April 2016
Published online 30th May 2016

Keywords:

Speech-to-text,
Text-to-speech,
Phoneme, Chemicals,
Speech Recognition,
Compounds.

ABSTRACT

In last many years, work has been done in audio processing. However it was not much used in the fields of Electronics and Computers due to its variety of speech signals and complexity. But now with the help of modern algorithms, it is possible to easily recognize the text from the given speech. In this project, we will develop an application, which will take the speech as input and give the text which will be specifically related to chemical names. It can also be further extended to provide a base for various applications like when the text is recognized it can give all the related information of that chemical. We know that there are applications already developed for audio processing and extracting text but they do not work that accurately with chemicals. Hence we make this application specially for chemical entities

INTRODUCTION

Speech recognition is the process of capturing spoken words using microphone or telephone and converting them into a digitally stored set of words. Speech to text conversion is one of the application of speech recognition. The main goal or objective of this paper is how to recognise the various sound in speech and provide the output as the text. Accuracy or correctness of speech recognition depends on various factors such as the size of the vocabulary, mode of speech like isolated, continuous, etc. To recognise different voice patterns, a technique based on Hidden Markov Model (HMM) is used, as it is one of the most efficient algorithm for speech recognition. Speech recognition system can be divided into different modules such as: Speech Acquistion, Speech Preprocessing, Hidden Markov Model and Text Storage. Finally we can extend this application to extract the chemical related text from the speech and provide the information about that chemical entity.

Related Work

There are some speech recognition techniques such as Natural Language Processing (NLP), etc. The idea from some of the base papers is used to extract text from the speech. Due to various reasons like variety of sounds in the speech, it becomes difficult to recognise the text from the input audio.

****Corresponding author: Farhaan Kaleem,***
Department of Computer Engineering, Savitribai Phule Pune University, Pune, Maharashtra, India

In past, many scientists were doing research on text extraction from input audio signals. As a result many algorithms were developed. Natural Language Processing (NLP) techniques can be used for this process. However these techniques do not work that effficiently when it comes to the chemical names. Therefore in this paper, we specifically stress on the chemical entities. The microphone input port with the audio codec receives the audio signal and produces the output as the text.

Method to extract Text from input audio signal

Overview

In this paper, the speech is taken as the input through microphone which is provided to the speech acquistion module. Then that speech is provided to the speech preprocessing module.

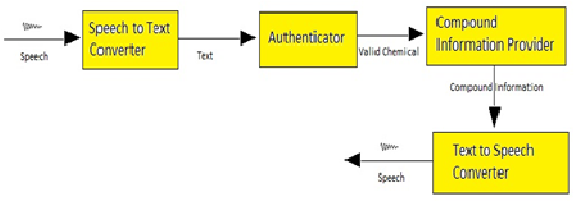

Fig. 1.1 Architechture of Proposed Method

Speech Acquistion

In this module, the microphone input port with the audio codec receives the signal, amplifies it, and converts it into

16-bit PCM digital samples at a sampling rate of 8 KHz. Here the analog audio signal is sampled on time and amplitude axes for digitisation of audio, so that further processing can be done on this digitised data. Speech signal is analysed in even interval which is usually 20ms [2]. Because the speech signal within this time interval is considered as stable. The final output of this module will be the digitised audio signal. This data is then stored in memory for further processing.

Speech Preprocessing

This digitised data is then given to the further module known as speech preprocessing, which extracts the features of the digitised audio signal like phonemes, unique patterns, etc. Preprocessing involves taking the speech samples as input, and breaking the samples into frames, and returning a unique pattern for each sample. The unique pattern can be achieved based on certain basic factors like phonemes.

The unique pattern can be achived by following steps:

- The digital samples are divided into overlapped frames.
- The system checks the frames for voice activity using endpoint detection and energy threshold calculations.
- The speech samples are passed through a pre-emphasis filter.
- The frames with voice activity are passed through a Hamming window. Hamming window returns only positive sample. It discards negative and zero values.
- The system finds linear predictive coding (LPC) coefficients for frames.
- From the LPC coefficients, the system determines the cepstral coefficients.

The cepstral coefficients serve as feature vectors.

We know that the frequency of speech of humans lies within specific range. An adult male will have a fundamental frequency from 85 to 180 Hz, and that of a typical adult female from 165 to 255 Hz. Thus pre-emphasis filter is used to extract only that audio, which lies in this frequency range. Linear predictive coding (LPC) is a tool used mostly inaudio signal processing and speech processing for representing the spectral envelope of a digital signal of speech in compressed form, using the information of a linear predictive model.

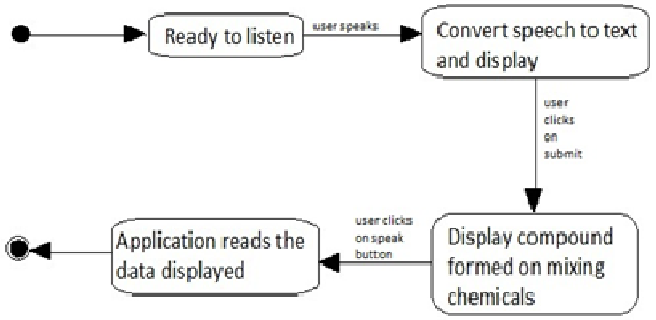

Fig. 2. State Transtion diagram

CMUSphinx toolkit

CMUSphinx toolkit is a leading speech recognition toolkit with various tools used to build speech applications. CMU Sphinx toolkit has a number of packages for different tasks and applications. It's sometimes confusing what to choose. To cleanup, here is the list:

- Pocketsphinx — lightweight recognizer library written in C.
- Sphinxbase — support library required by Pocketsphinx
- Sphinx4 — adjustable, modifiable recognizer written in Java
- Sphinxtrain — acoustic model training tools

Speech recognition using Sphinx toolkit consists basically of four steps:

- Building the dictionary
- Building the language model
- Adapting or building acoustic model
- Tuning the performance

Building the dictionary

There are various tools to help you to extend an existing dictionary for new words or to build a new dictionary from scratch. If your language already has a dictionary it's recommended to use since it's carefully tuned for best performance. If you starting a new language you need to account for various reductions and coarticulations effects. They make it very hard to create accurate rules to convert text to sounds. However, the practice shows that even naive conversion could produce a good results for speech recognition. For most of the languages you need to use specialized grapheme to phoneme (g2p) code to do the conversion using machine learning methods and existing small database. Nowdays most accurate g2p tools are Phonetisaurus: http://code.google.com/p/phonetisaurus. If TTS is used, you often need to do phoneset conversion. TTS phonesets are usually more extensive than required for ASR. However, there is a great adavantage in TTS tools because they usually contain more required functionality than simple G2P. For example, they are doing tokenization by converting numbers and abbreviations to spoken format. For English you can use simplier capabilities by using on-line webservice: http://www.speech.cs.cmu.edu/tools/lmtool.html

Building the language model

There are several types of models that describe language to recognize - keyword lists, grammars and statistical language models, phonetic statistical language models. You can chose any decoding mode according to your needs and you can even switch between modes in runtime.

Keywords-List

Pocketsphinx supports keyword spotting mode where you can specify the keyword list to look for. The advantage of this mode is that you can specify a threshold for each keyword so that keyword can be detected in continuous speech. All other modes will try to detect the words from grammar even if you used words which are not in grammar. Threshold must be specified for every keyphrase. For shorter keyphrase you can use smaller thresholds like 1e-1, for longer threshold must be bigger, up to 1e-50. If your keyphrase is very long, larger than 10 syllables, it is recommended to split it and spot for parts separately. Threshold must be tuned to balance between false

alarms and missed detections, the best way to tune threshold is to use a prerecorded audio file.

Tuning process is the following:

- Take a long recording with few occurrences of your keywords and some other sounds. You can take a movie sound or something else. The length of the audio should be approximately 1 hour
- Run keyword spotting on that file with different thresholds for every keyword, use the following command: pocketsphinx_continuous -infile <your_file.wav> -keyphrase <"your keyphrase"> -kws_threshold \ <your_threshold> -time yes
- From keyword spotting results count how many false alarms and missed detections you've encountered
- Select the threshold with smallest amount of false alarms and missed detections

Grammars

Grammars describe very simple type of the language for command and control, and they are usually written by hand or generated automatically within the code. Grammars usually do not have probabilities for word sequences, but some elements might be weighed. Grammars could be created with JSGF format and usually have extension like .gram or .jsgf.

Language Models

Statistical language models describe more complex language. They contain probabilities of the words and word combinations. Those probabilities are estimated from a sample data and automatically have some flexibility. For example, every combination from the vocabulary is possible, though probability of such combination might vary. For example if you create statistical language model from a list of words it will still allow to decode word combinations though it might not be your intent. Overall, statistical language models are recommended for free-form input where user could say anything in a natural language and they require way less engineering effort than grammars, you just list the possible sentences. Overall, modern speech recognition interfaces tend to be more natural and avoid command-and-control style of previous generation. For that reason most interface designers prefer natural language recognition with statistical language model than old-fashioned VXML grammars.

Adapting or building acoustic model

There are, however, applications where the current models won't work. For example, handwriting recognition or dictation support for another language. In these cases, you will need to train your own model and this tutorial will show you how to do that for the CMUSphinx speech recognition engine. Before starting with training make sure you are familar with concepts, prepared the language model and you indeed need to train the model and have resources to do that.

Data Preparation

The trainer learns the parameters of the models of the sound units using a set of sample speech signals. This is called a training database. A choice of already trained databases will also be provided to you. The database contains information required to extract statistics from the speech in form of the acoustic model. Database should have enough speakers recording, variety of recording conditions, enough acoustic variations and all possible linguistic sentences. The size of the database depends on the complexity of the task you want to handle as mentioned above. A Database should have the two parts mentioned above - training part and test part. Usually test part is about 1/10th of the full data size, but we don't recommend you to have test data more than 4 hours of recordings. You have to design database prompts and postprocess the results to ensure that audio actually corresponds to prompts. The file structure for the database is: etc

- your_db.dic - Phonetic dictionary
- your_db.phone - Phoneset file
- your_db.lm.DMP - Language model
- your_db.filler - List of fillers
- your_db_train.fileids - List of files for training
- your_db_train.transcription - Transcription for training
- your_db_test.fileids - List of files for testing
- your_db_test.transcription - Transcription for testing
- wav
- speaker_1
- file_1.wav - Recording of speech utterance
- speaker_2
- file_2.wav

It's critical to have audio files in a specific format. Sphinxtrain does support some variety of sample rates but by default it's configured to train from 16khz 16bit mono files in MS **WAV** format. YOU NEED TO MAKE SURE THAT YOU RECORDINGS ARE AT A SAMPLING RATE OF 16 KHZ (or 8 kHz if you train a telephone model) IN MONO WITH SINGLE CHANNEL.

Adapting the default acoustic model

The adaptation process takes transcribed data and improves the model you already have. It's more robust than training and could lead to a good results even if your adaptation data is small. For example, it's enough to have 5 minutes of speech to significantly improve the dictation accuracy by adaptation to the particular speaker.

Required Files

The actual set of sentences you use is somewhat arbitrary, but ideally it should have good coverage of the most frequently used words or phonemes in the set of sentences or the type of text you want to recognize. For example, if you want to recognize isolated commands, you need tor record them. If you want to recognize dictation, you need to record full sentences. For simple voice adaptation we have had good results simply using sentences from the CMU ARCTIC text-to-speech databases. To that effect, here are the first 20 sentences from ARCTIC, a fileids file, and a transcription file

- arctic20.fileids
- arctic20.transcription

Adapting the acoustic model

First we will copy the default acoustic model from PocketSphinx into the current directory in order to work on it. Assuming that you installed PocketSphinx under /usr/local, the acoustic model directory is /usr/local/share/pocketsphinx/model/en-us/en-us. Copy this directory to your working directory: cp -a /usr/local/share/pocketsphinx/model/en-us/en-us.

Generating acoustic feature files

In order to run the adaptation tools, you must generate a set of acoustic model feature files from these WAV audio recordings. This can be done with the sphinx_fe tool from SphinxBase. It is imperative that you make sure you are using the same acoustic parameters to extract these features as were used to train the standard acoustic model. Since PocketSphinx 0.4, these are stored in a file called feat.params in the acoustic model directory. You can simply add it to the command line for sphinx_fe, like this:

```
sphinx_fe -argfile en-us/feat.params \
      -samprate 16000 -c arctic20.fileids \
     -di . -do . -ei wav -eo mfc -mswav yes
```

Tuning the performance

Speech recognition accuracy is not always great.

The first thing you need to understand if your accuracy just lower than expected or very low. If it's very low most likely you misconfigured the decoder. If it's lower than expected, you can apply various ways to improve it. The first thing you should do is to collect a database of test samples and measure the recognition accuracy. You need to dump utterances into wav files, write reference text and use decoder to decode it. Then calculate WER using the word_align.pl tool from Sphinxtrain. Test database size depends on the accuracy but usually it's enough to have 30 minutes of transcribed audio to test recognizer accuracy reliably.

Test Database setup

To test the recognition you need to configure the decoding with the required paramters, in particular, you need to have a language model <your.lm>. Create fileids file test.fileids. Create transcription file test.transcription. Put the audio files in wav folder. Make sure those files have proper format and sample rate.

Now run the decoder:

```
pocketsphinx_batch \
 -adcin yes \
 -cepdir wav \
 -cepext .wav \
 -ctl test.fileids \
 -lm <your.lm, for example en-us.lm.dmp from pocketsphinx> \
 -dict <your.dic, for example cmudict-en-us.dict from pocketsphinx> \
 -hmm <your_hmm, for example en-us> \
 -hyp test.hyp

word_align.pl test.transcription test.hyp
```

word_align.pl script is a part of sphinxtrain distribution

Make sure to add -samprate 8000 to the above command if you are decoding 8kHz files! The script word-align.pl from Sphinxtrain will report you the exact error rate which you can use to decide if adaptation worked.

Mathematical Model

Let us consider a set S = { I, O, Fn, Su, F }
Where,
I => Input
I = {Audio or Speech}
O => Output
O = {Text which is extracted from the audio signal}
Su => Success
Su = {Proper word or sentence which is spoken is obtained}
F => Failure
F = {Unable to extract the patterns from the input audio signal and thus failure in giving the correct word as output in the form of text}
Fn => Functions
Fn = {speechAcquistion(), formFrames() }

speechAcquistion()
{
This function takes the audio signal as the input and using Pulse Codde Modulation can encode the input audio into 16-bit encoded data.
}
formFrames()
{
This function is used to form the input encoded audio signals into frames of size 960.
short pcm_[960];
}

Preprocessing()
{
It is used to extract the feature vector in the form of ceptral coeffecients
}

HiddenMarcov()
{
It is used to compare the words by using probability distribution and give the desired text as the output.
}

Conclusions

Therefore we conclude that Hidden Marcov Model (HMM) can be used to efficiently extract the text from the input audio. It is used to recognize the speech and thus give an output in textual format. The older techniques, were not able to efficiently extract chemical names. But HMM can efficiently extract the chemical names. And give the desired text as an output.

REFERENCES

Raghavendhar Reddy, B. and E. Mahender, Speech to Text Conversion using Android Platform. January -February 2013, Parvathapur, Uppal, Hyderabad, India.

MJF Gales. Semi-tied full-covariance matrices for hidden Markov models. 1997.

Hermansky, H. 1990. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*, 87(4):1738{1752.

Harb, B., Chelba, C., Dean, J. and G. Ghemawhat. 2009. Back-o_ Language Model Compression.

Maryam Kamvar and Shumeet Baluja. A large scale study of wireless search behavior: Google mobile search. In CHI, pages 701{709, 2006.

Wikipedia

****