Electrical and Computer Engineering Department

Data Set Crawler for Multilingual ASR Engine

Mo Shi

Zekun Zhang

Ziqi Wang

Chaoji Zuo

Duc Le

Team Project Number:

S19-38

Advisor:

Dr. Shahab Jalavand (from Interactions)

May 1, 2019

Electrical and Computer Engineering Department

Rutgers University, Piscataway, NJ 08854

1. Introduction

Our project focuses on constructing training data set for existing ASR engines. Currently, the ASR engine can be used to recognize the speech contents of English, Chinese, and other languages that have a large enough population of speakers. The input data for the ASR engine training process contains two parts: the speech content and the related text content. With a specific technology called "forced alignment," the engine will be able to match those two contents together. Then the audio and corresponding text will be in alignment and can be used for subsequent training. But one major problem faced by the current ASR engine is that it's hard to find resources that contain abundant data for languages used by the minority in another word they are the languages which are "not so popular".

Our project topic is provided by Interactions LLC, which focuses on finding some multilingual resources and finishing the web crawling to construct a normalized training dataset. Besides, we are going to do the force alignment work with the Montreal Forced Aligner and display our result with Praat, finally use the results to analyze the quality.

The first step of our project is to collect data using the technique named "web crawling" or "web scraping". The web crawler can be able to recognize the structure of resource websites and find the detail URL-links that contain both the audio documents and the corresponding texts. After finding appropriate resources, the crawler will download those contents automatically and put the data into the database. In this way, we've built up the training dataset for the ASR engine.

The second step of this project is the evaluation and training data. The evaluation will be done through 1) the efficiency of the program (the amount of data we can collect in a specific length of time) and 2) the rate of speech-text alignment (the number of words in the text that can be matched with the audio), and 3) how many small languages resources contains in the database. After the evaluations, if the datasets are good enough, we are going to train these data with the Montreal Forced Aligner. Then we can test the training result and if it's good enough we would go on our next step--displaying.

Our project in the future can work like text-speech translation tools such as video caption real-time generator. In addition, samples with several people talking at the same time and noisy samples with low signal-to-noise ratio need to be filtered, which could be implemented as neural network-based modules.

2. Methods / Approach

2.1. Methods

2.1.1. Description

In order to achieve our task, we need to design a crawler system to craw the audio and text resources from different websites. Our crawler should take a link of a website as input, then the crawler can locate the audio file's URL and its relevance text in the website, and download a list of audios and their relevance text. And the efficiency of the system is very import because we need to download a lot of audios. We use BeautifulSoup to parse HTML files and use Selenium to emulate the browser's process in Python.

Another important part of the crawler system is the selection of the website. Cause it is impossible to create one crawler which can crawl all the web on the Internet, we need to choose several websites which can provide us enough and high-quality sources. Thus, target resources websites should satisfy the following conditions: a sufficient amount of videos or audios, support multilingual resources, have some subtitles or text of those videos or audios.

2.1.2. Resource Finding

Difficulties:

Since the existing ASR system downloaded videos on YouTube and BBC news combined with the description texts as the training set. At the very beginning, we tried to analysis video websites such as BBC, TED, and Republic. However, we were faced with some problems: 1) The combined description text is in another language such as English. 2)The quality of the videos is so unstable, the description text can’t match the video, or the video do not contain human voice. 3) The videos are very big, while we need enough time videos to train a single language, but our computers don’t have such a large space for these data. 4) Generalization is difficult. Because these websites use different structures, so if we want to train another language, we need to generate another app to solve this problem, which is not effective.

Requirements:

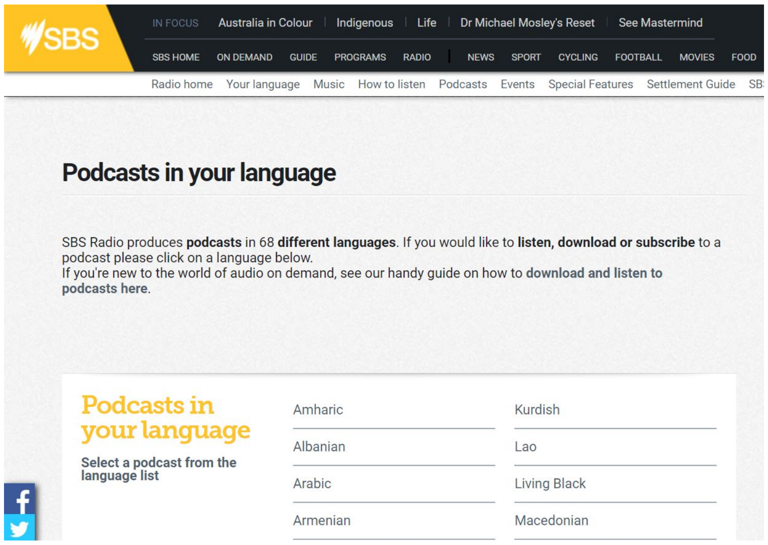
As discussed above, we have specified requirement for videos: 1) The description and video should in the same language. 2) The match rate between video and combined text should be high enough. 3) The scale of data should be small, but the quality of data needs to be high. 4) If possible, we want to find a website that has all kinds of languages so that we can achieve generalization.

We start to find the audio news websites, we got SBS news, but it still has some internal problems, although the data is in small scale and it contains multiple languages, it still didn't solve the match rate problem.

So we need to come up with an idea that the match rate is great enough, the free audiobook is the best. And Mo Shi came up with the idea that maybe we can find Bible websites and it is the widest spread published books all over the world, so it must contain multiple languages, and we can solve the match rate problem as the audio is based on the book. Then we made the decision of our resources.

2.1.3. Language Selection

With the requirement of multiple languages supporting, both resource websites we found contains a web page that allows users to achieve language selection. The link for each language will derive the user to a page where all resources in this specific language are located, so we chose the page of this language list as the main page where our program starts to work.



Podcasts in your language

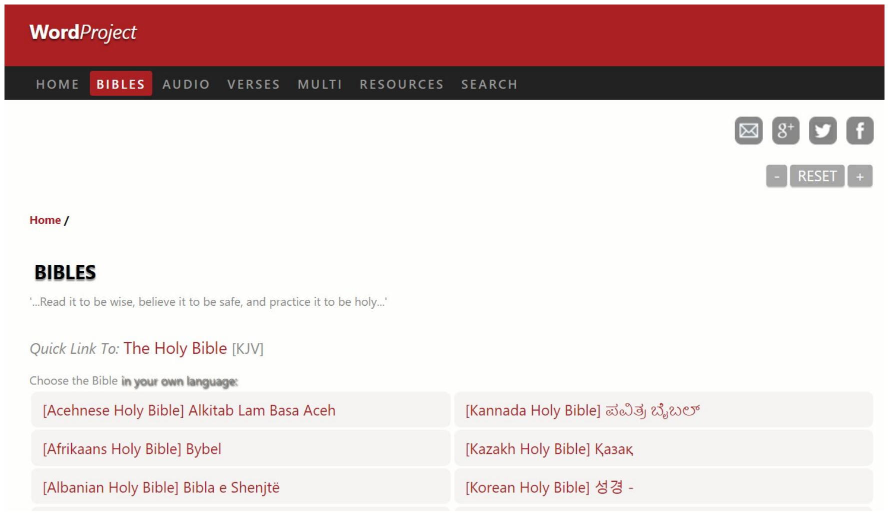
SBS Radio produces **podcasts** in 68 **different languages**. If you would like to **listen, download or subscribe** to a podcast please click on a language below.
If you're new to the world of audio on demand, see our handy guide on how to **download and listen to podcasts here**.

Podcasts in your language

Select a podcast from the language list

Amharic
Albanian
Arabic
Armenian
Kurdish
Lao
Living Black
Macedonian

2.1.1-1 language list of SBS Radio



BIBLES

'...Read it to be wise, believe it to be safe, and practice it to be holy...'

Quick Link To: The Holy Bible [KJV]

Choose the Bible in your own language:

[Acehnese Holy Bible] Alkitab Lam Basa Aceh	[Kannada Holy Bible] ಪವಿತ್ರ ಬೈಬಲ್
[Afrikaans Holy Bible] Bybel	[Kazakh Holy Bible] Қазақ
[Albanian Holy Bible] Bibla e Shenjtë	[Korean Holy Bible] 성경 -

2.1.1-2 language list of WordProject

One thing needs to take notice of is that not all languages provide the website that fits our requirement. For instance, in SBS Radio the language list contains "Lao", but when we open its page it turns out that there's no podcast, which means the web page is currently incomplete so that Lao is actually not supported. Similarly, in the list of WordProject, though the text version of Bible in different languages can always be found, the audio version in some languages are not provided. Moreover, for some of the languages the web page has not been implemented, so the link will actually derive the user to some other bible websites.

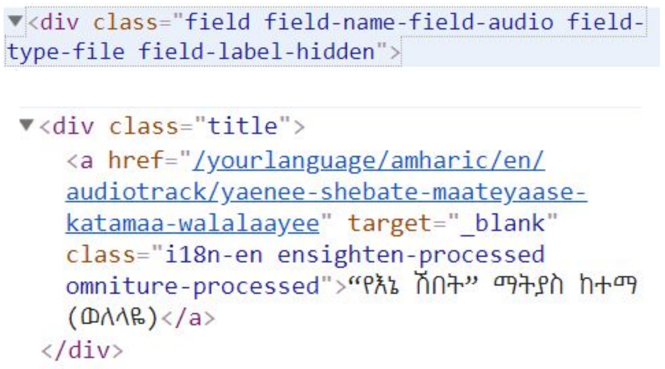
To guarantee that our program always works correctly, we need to make sure only when the language input by the user is supported does the program start to search for all resources. In order to achieve this, first we did some manual test for each language to see if it provides the resources we wanted (this could be achieved using some extra codes but since there's not too much work we just chose a more straightforward way). After this manual filtering part, we got a list of all available language choices successfully. The next step we took was building a Python dictionary of key-value pairs, with key being the language itself and value being its hyperlink. Once the user input the target language, the program will find the hyperlink in that dictionary and look for corresponding resources. If user's input is not a key of the dictionary, i.e. the language is currently not supported, the program will return a notice to let the user input again.

2.1.4. Text Crawling

Once the language has been selected, the program will jump to the page where all resources in this specific language are located. The next step is to find the audio and the corresponding text.

For SBS Radio, the page contains a column of rows, with each row being the section of one piece of news that contains a hyperlink. When clicking the link, a new page of that specific news will open, in which we could find its audio document and description. Besides the description, some news also has extra detailed contents below the description, so the text we want for our program contains two parts: the description and the detailed contents.

The first thing to do is to find all news we want and build a list of their hyperlinks, which means we need to traverse all sections in all pages. In the html document of each page, we found that all pieces of news are stored in the same structure of a "div" tag with the same class name. Moreover, under this tag we could find another "div" tag with an attribute of *"class='title'"* that contains the hyperlink we wanted. So in order to find all news, it actually means we need to find all tags of this kind.



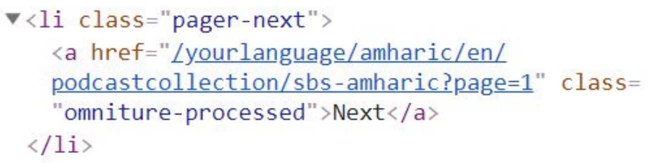
```
▼<div class="field field-name-field-audio field-
type-file field-label-hidden">

▼<div class="title">
    <a href="/yourlanguage/amharic/en/
    audiotrack/yaenee-shebate-maateyaase-
    katamaa-walalaayee" target="_blank"
    class="i18n-en ensighten-processed
    omniture-processed">"የእኔ ሸበት" ማትያስ ከተማ
    (ወላላዬ)</a>
  </div>
```

2.1.4-1 "div" tag where the hyperlink to the news is stored

To achieve this, we imported a Python library called "BeautifulSoup", which is used to construct the html document to a tree, and each node of the tree is a tag object corresponding to the html document. In this way, we could easily find any particular tag and all its information which is achieved by the "find all" method of BeautifulSoup.

After getting all resources of one page, we need our program to jump to the next page and get the resources recursively until all pages have been visited. This is achieved by finding the hyperlink to the next page, which is stored in the "next-page button" of each page. This button can also be found easily using the same find-all method above. The hyperlink is stored in a "li" tag.



```
▼<li class="pager-next">
   <a href="/yourlanguage/amharic/en/
   podcastcollection/sbs-amharic?page=1" class=
   "omniture-processed">Next</a>
 </li>
```

2.1.4-2 "li" tag that contains the hyperlink to next page

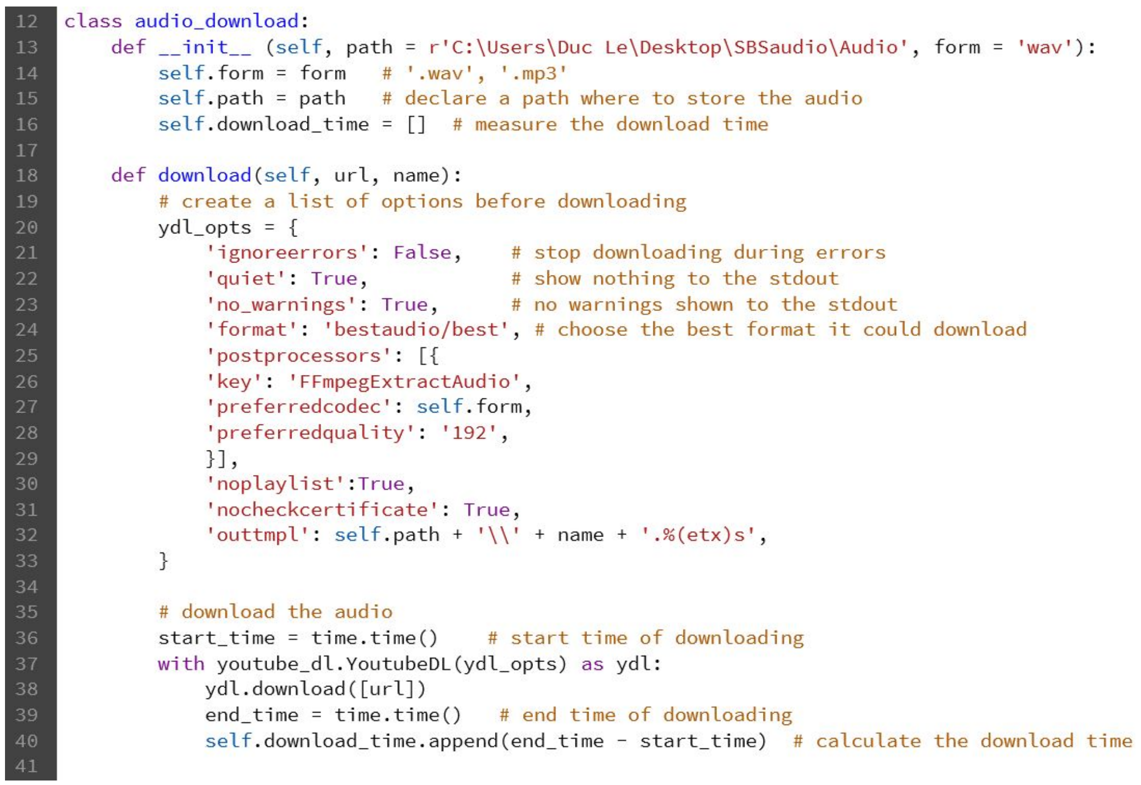
In SBS Radio website, the maximum number of pages is 10. When reaching the last page, there won't be any next-page button found, so the program will know the searching process is over and stop.

Now we're able to jump to the hyperlink of each resource, the last step is to locate the description and contents and extract them into a .txt document as the final text we need. Usually the text contents in a html document is stored in a specific type of tag named "p", which also fits to our situation. Though there are many p-tags in one page and some texts are not what we are looking for, we found that all useful texts are stored in the same "div" tag. Then we could just find all paragraphs under this specific tag, combine them in a .txt document with a simple file I/O operation and get our final text successfully.

For WorldProject, the crawling procedures are actually simpler, because all texts of different chapters in bible are already divided, and for each chapter all contents are put together in the same place in the page. The only difference is that now we need to traverse all chapters instead of all sections of news.

2.1.5. Audio Crawling / Format Transformation

Audio crawling is one of the most difficult parts of the project. There are some free tools online that could help with downloading video, not audio and that is the problem. If video is downloaded, the audio must be extracted and that will take some time. Even when the audio extraction is implemented with ease, the storage for video is very unnecessary. Downloading the video or audio is also troublesome since the tool might get confused when there is more than 2 videos are playing in the same webpage or it just downloads a part of the video. To tackle this problem, a lot of tools had been tested and some tactics had been used to see the effectiveness of each tool. The first tactic is to download the video and then extract the audio. This takes long time due to downloading the best quality video (to have best audio quality) and sometimes, it downloads a part of the video. To extract the audio, FFmpeg tool is used. The second tactic is to find a tool that could do the converting after downloading and only stores the audio, which is more memory-efficient. The tool found is youtube_dl, which gives quite a lot of options for the downloading and converting. Belows is the class created to download audio using youtube_dl:



```python
class audio_download:
    def __init__ (self, path = r'C:\Users\Duc Le\Desktop\SBSaudio\Audio', form = 'wav'):
        self.form = form   # '.wav', '.mp3'
        self.path = path   # declare a path where to store the audio
        self.download_time = []  # measure the download time

    def download(self, url, name):
        # create a list of options before downloading
        ydl_opts = {
            'ignoreerrors': False,    # stop downloading during errors
            'quiet': True,            # show nothing to the stdout
            'no_warnings': True,      # no warnings shown to the stdout
            'format': 'bestaudio/best', # choose the best format it could download
            'postprocessors': [{
            'key': 'FFmpegExtractAudio',
            'preferredcodec': self.form,
            'preferredquality': '192',
            }],
            'noplaylist':True,
            'nocheckcertificate': True,
            'outtmpl': self.path + '\\' + name + '.%(etx)s',
        }

        # download the audio
        start_time = time.time()    # start time of downloading
        with youtube_dl.YoutubeDL(ydl_opts) as ydl:
            ydl.download([url])
            end_time = time.time()   # end time of downloading
            self.download_time.append(end_time - start_time)  # calculate the download time
```

2.1.5-1 using youtube_dl to download/transform audios

As can be seen, youtube_dl use FFmpeg to do video conversion.

2.1.6. Forced Alignment

Before putting all the resources we crawled into the ASR engine, there's one last step to let our data be in the good shape for training -- we need to match the contents in the text with the corresponding part in the audio, which requires the implementation of forced alignment.

Forced alignment is a technique that for each word appears in the text, it will find its position in the audio described with the starting time and ending time, and generate a new TextGrid file in which all texts are aligned with the corresponding audio contents. To achieve this, we used a tool called Montreal Forced Aligner, which implemented the forced alignment function with the underlying technology of Kaldi ASR Toolkit. There are other current tools that achieved this function, but we chose MFA since it supports multiple languages and provides pre-trained models for each of them.

Two types of models are used in this procedure, which are Grapheme-to-Phoneme (G2P) model and acoustic model. G2P model is used to generate a “pronunciation dictionary” for all resources under each language, which looks like the following:

(2.1.6-1 pronunciation dictionary)

The first column lists all words appeared in all text resources, and the second column lists a series of phonemes the tells how each word is pronounced in that language. With this pronunciation dictionary being the third input besides the audio/text pair, MFA will be able to align the audio and texts based on the other model called acoustic model. Though MFA has a series of acoustic models for each language, it also allows the user to train their own models based on any data set provided -- this is actually a general use case for forced alignment that train and align the same data (i.e. the training and testing data will be exactly the same). The final output for each audio/text pair aftering training and aligning will be a TextGrid file that looks like this:

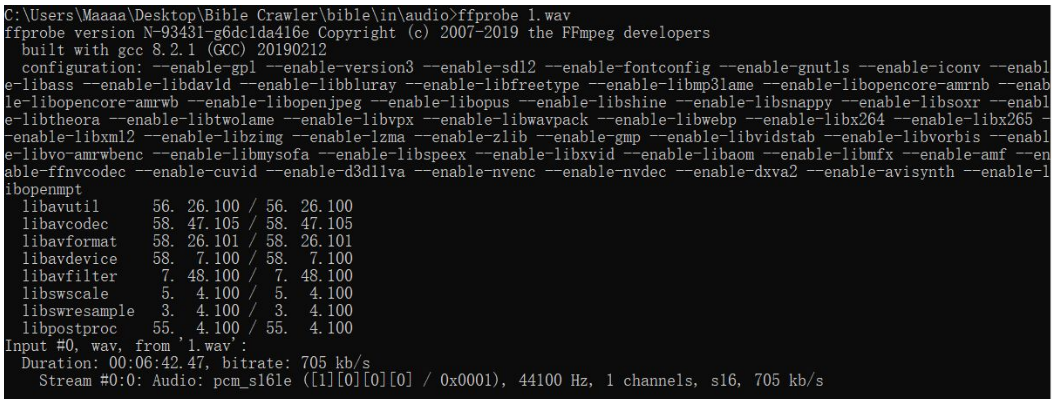
(2.1.6-2 TextGrid file opened by Praat)

Above is a sample output opened by a software called Praat. In this picture we can see that each word are aligned with the corresponding audio in the TextGrid file, which also suggests it’s now in the good shape for ASR training.

2.2. Experiment / Product Results

We tested our programs on a Windows 10 / Mac operating system. The languages we chose for testing are Hindi and Cantonese, which are supported both by SBS Radio and Bible websites (for other unsupported languages the programs were able to recognize and raise an error to let the user input again). Our current estimation concerns mainly two aspects: the correctness and time cost.

For the Bible crawler, the program is able to download and transform audios and texts in around 53 minutes for nearly all 1,189 chapters of the Bible – for some of the chapters (for our test always less than 5) there might be some problem while requesting for the web page, and if that occurs we just simply abandon that resource since it does not have a big influence. We tested the format of the audio we’ve got after transformation using the “ffprobe” command, and the results are as following:



```
C:\Users\Maaaa\Desktop\Bible Crawler\bible\in\audio>ffprobe 1.wav
ffprobe version N-93431-g6dc1da416e Copyright (c) 2007-2019 the FFmpeg developers
  built with gcc 8.2.1 (GCC) 20190212
  configuration: --enable-gpl --enable-version3 --enable-sdl2 --enable-fontconfig --enable-gnutls --enable-iconv --enabl
e-libass --enable-libdav1d --enable-libbluray --enable-libfreetype --enable-libmp3lame --enable-libopencore-amrnb --enab
le-libopencore-amrwb --enable-libopenjpeg --enable-libopus --enable-libshine --enable-libsnappy --enable-libsoxr --enabl
e-libtheora --enable-libtwolame --enable-libvpx --enable-libwavpack --enable-libwebp --enable-libx264 --enable-libx265 -
-enable-libxml2 --enable-libzimg --enable-lzma --enable-zlib --enable-gmp --enable-libvidstab --enable-libvorbis --enabl
e-libvo-amrwbenc --enable-libmysofa --enable-libspeex --enable-libxvid --enable-libaom --enable-libmfx --enable-amf --en
able-ffnvcodec --enable-cuvid --enable-d3d11va --enable-nvenc --enable-nvdec --enable-dxva2 --enable-avisynth --enable-l
ibopenmpt
  libavutil      56. 26.100 / 56. 26.100
  libavcodec     58. 47.105 / 58. 47.105
  libavformat    58. 26.101 / 58. 26.101
  libavdevice    58.  7.100 / 58.  7.100
  libavfilter     7. 48.100 /  7. 48.100
  libswscale      5.  4.100 /  5.  4.100
  libswresample   3.  4.100 /  3.  4.100
  libpostproc    55.  4.100 / 55.  4.100
Input #0, wav, from '1.wav':
  Duration: 00:06:42.47, bitrate: 705 kb/s
    Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 44100 Hz, 1 channels, s16, 705 kb/s
```

2.2-1 format of audio sample

It can be seen that the audio fits our requirements of sample rate being 44100Hz and channels being 1 or 2, which shows the correctness of our program.

For the SBS crawler we did similar test, and our program was able to gain all 100 current resources in the particular language in 10 minutes. We also tried to apply some filtering strategies to the resources we obtained. For example, we chose the maximum duration to be 8 minutes and the minimum text length to be 100 characters to guarantee the quality of our resources to some extent. Unfortunately, in this case only around 30 resources out of 100 were able to pass the filter, which suggests the resources we got may not be of good enough quality.

A sample output of the final TextGrid file after alignment is already shown in 2.1.6. To test the quality of our resources, we need to compute the matching rate (i.e. the successfully aligned time divided by the total time) based on the information in all TextGrid files, which will be illustrated in the upcoming section.

3. Estimation

3.1. Description

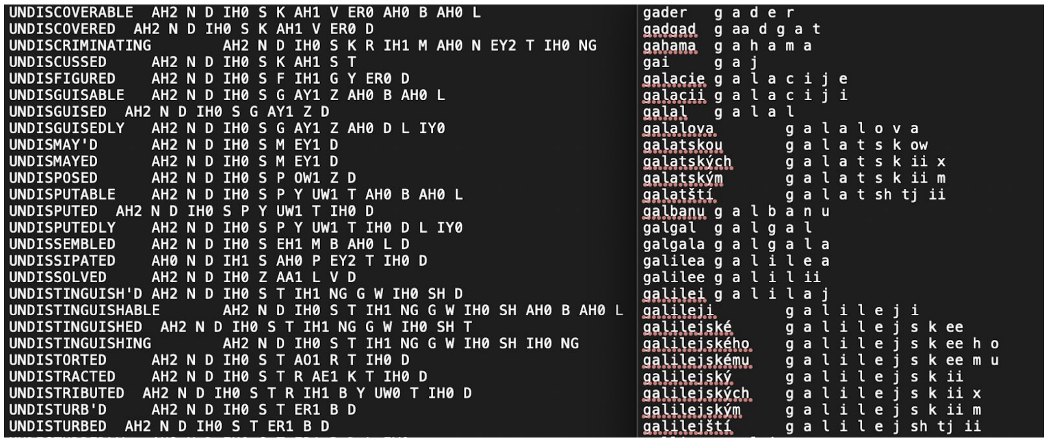
In this section we'll illustrate our methods of estimation. More specifically, we'll list the factors we found that influence the performance, which include 1) the property of the language itself and the generated pronunciation dictionary, 2) the quality of the acoustic model and the provided training data set, 3) the beam value for training and aligning, and 4) other findings. For the performance measurement, we evaluated the performance through different aspects. The major aspect is the match rate. Besides, we also looked at the number of documents being successfully aligned and the total time cost for alignment.

3.2. Language / Pronunciation Dictionary

After we've obtained all the normalized audio and text resources, for any single language, the most very first thing is to generate a pronunciation dictionary. The dictionary is generated by G2P model. The G2P model is a pretrained model by Montreal Forced Aligner for some languages and it is trying to combine the audio information (phonemes) and text information (graphemes) together, which means to change grapheme to phoneme. So in another word, the pronunciation dictionary is a .txt file which contains words and corresponding pronunciations. Just like the Figure() shows, the first column is a word and second column is its pronunciation.

It looks like it's a easy job to finish the force alignment work. However, in our training process, we found that due to the quality of G2P model, the result of some languages would be really bad and almost impossible for the future training work. As it shown in Figure(), the left part is a dictionary for Chinese character, but if you know something about Chinese, you would immediately find that this is not a dictionary for Chinese because it can't be used in other cases. Chinese is kind of a character language which means every character in Chinese is a word, but in the dictionary we produced in our project, the 'word' contains not only one character, in another word, it's just like a sentence in other languages and a sentence is almost impossible for training use in our project.

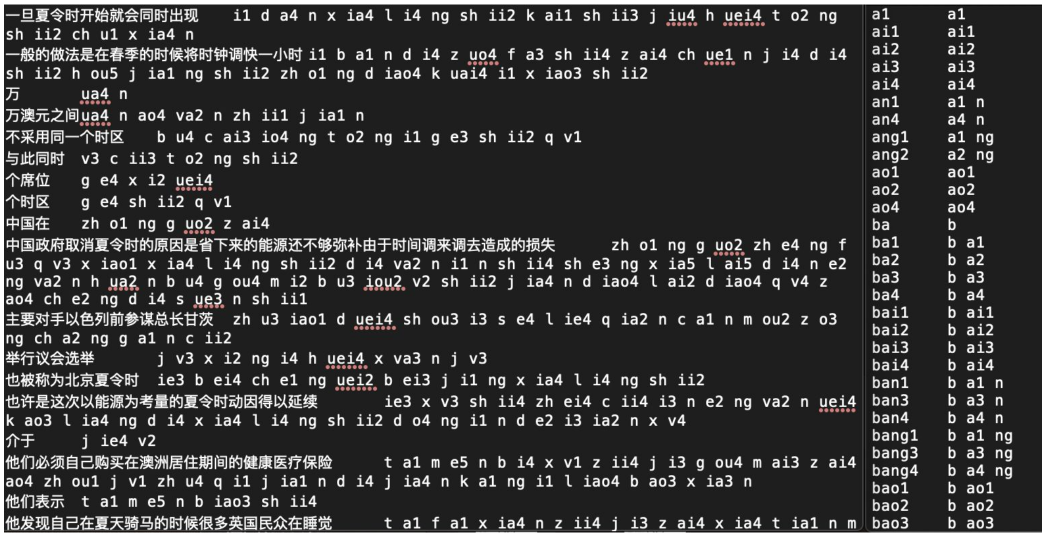
In a conclusion that, first the quality of G2P models determine the quality of the dictionaries trained by these models. But in our project, we won't train our own G2P models. So, the only way to improve the quality of our train data is trying to use more data to create a better dictionary. And what's more, due to the different language system, some Latin-based language would get a better performance than the languages like Chinese, Japanese and Korean.



```
UNDISCOVERABLE  AH2 N D IH0 S K AH1 V ER0 AH0 B AH0 L
UNDISCOVERED  AH2 N D IH0 S K AH1 V ER0 D
UNDISCRIMINATING        AH2 N D IH0 S K R IH1 M AH0 N EY2 T IH0 NG
UNDISCUSSED     AH2 N D IH0 S K AH1 S T
UNDISFIGURED    AH2 N D IH0 S F IH1 G Y ER0 D
UNDISGUISABLE   AH2 N D IH0 S G AY1 Z AH0 B AH0 L
UNDISGUISED  AH2 N D IH0 S G AY1 Z D
UNDISGUISEDLY   AH2 N D IH0 S G AY1 Z AH0 D L IY0
UNDISMAY'D      AH2 N D IH0 S M EY1 D
UNDISMAYED      AH2 N D IH0 S M EY1 D
UNDISPOSED      AH2 N D IH0 S P OW1 Z D
UNDISPUTABLE    AH2 N D IH0 S P Y UW1 T AH0 B AH0 L
UNDISPUTED  AH2 N D IH0 S P Y UW1 T IH0 D
UNDISPUTEDLY    AH2 N D IH0 S P Y UW1 T IH0 D L IY0
UNDISSEMBLED    AH2 N D IH0 S EH1 M B AH0 L D
UNDISSIPATED    AH0 N D IH1 S AH0 P EY2 T IH0 D
UNDISSOLVED     AH2 N D IH0 Z AA1 L V D
UNDISTINGUISH'D AH2 N D IH0 S T IH1 NG G W IH0 SH D
UNDISTINGUISHABLE       AH2 N D IH0 S T IH1 NG G W IH0 SH AH0 B AH0 L
UNDISTINGUISHED  AH2 N D IH0 S T IH1 NG G W IH0 SH T
UNDISTINGUISHING        AH2 N D IH0 S T IH1 NG G W IH0 SH IH0 NG
UNDISTORTED     AH2 N D IH0 S T AO1 R T IH0 D
UNDISTRACTED    AH2 N D IH0 S T R AE1 K T IH0 D
UNDISTRIBUTED  AH2 N D IH0 S T R IH1 B Y UW0 T IH0 D
UNDISTURB'D     AH2 N D IH0 S T ER1 B D
UNDISTURBED  AH2 N D IH0 S T ER1 B D
```

```
gader   g a d e r
gadgad  g aa d g a t
gahama  g a h a m a
gai     g a j
galacie g a l a c i j e
galacii g a l a c i j i
galal   g a l a l
galalova        g a l a l o v a
galatskou       g a l a t s k ow
galatských      g a l a t s k ii x
galatským       g a l a t s k ii m
galatští        g a l a t sh tj ii
galbanu g a l b a n u
galgal  g a l g a l
galgala g a l g a l a
galilea g a l i l e a
galilee g a l i l ii
galilei g a l i l a j
galileji        g a l i l e j i
galilejské      g a l i l e j s k ee
galilejského    g a l i l e j s k ee h o
galilejskému    g a l i l e j s k ee m u
galilejský      g a l i l e j s k ii
galilejských    g a l i l e j s k ii x
galilejským     g a l i l e j s k ii m
galilejští      g a l i l e j sh tj ii
```

FIGURE: left is dictionary for English, right is dictionary for Czech



```
一旦夏令时开始就会同时出现    i1 d a4 n x ia4 l i4 ng sh ii2 k ai1 sh ii3 j iu4 h uei4 t o2 ng
sh ii2 ch u1 x ia4 n
一般的做法是在春季的时候将时钟调快一小时 i1 b a1 n d i4 z uo4 f a3 sh ii4 z ai4 ch ue1 n j i4 d i4
sh ii2 h ou5 j ia1 ng sh ii2 zh o1 ng d iao4 k uai4 i1 x iao3 sh ii2
万      ua4 n
万澳元之间ua4 n ao4 va2 n zh ii1 j ia1 n
不采用同一个时区    b u4 c ai3 io4 ng t o2 ng i1 g e3 sh ii2 q v1
与此同时  v3 c ii3 t o2 ng sh ii2
个席位    g e4 x i2 uei4
个时区    g e4 sh ii2 q v1
中国在    zh o1 ng g uo2 z ai4
中国政府取消夏令时的原因是省下来的能源还不够弥补由于时间调来调去造成的损失      zh o1 ng g uo2 zh e4 ng f
u3 q v3 x iao1 x ia4 l i4 ng sh ii2 d i4 va2 n i1 n sh ii4 sh e3 ng x ia5 l ai5 d i4 n e2
ng va2 n h ua2 n b u4 g ou4 m i2 b u3 iou2 v2 sh ii2 j ia4 n d iao4 l ai2 d iao4 q v4 z
ao4 ch e2 ng d i4 s ue3 n sh ii1
主要对手以色列前参谋总长甘茨  zh u3 iao1 d uei4 sh ou3 i3 s e4 l ie4 q ia2 n c a1 n m ou2 z o3
ng ch a2 ng g a1 n c ii2
举行议会选举       j v3 x i2 ng i4 h uei4 x va3 n j v3
也被称为北京夏令时  ie3 b ei4 ch e1 ng uei2 b ei3 j i1 ng x ia4 l i4 ng sh ii2
也许是这次以能源为考量的夏令时动因得以延续       ie3 x v3 sh ii4 zh ei4 c ii4 i3 n e2 ng va2 n uei4
k ao3 l ia4 ng d i4 x ia4 l i4 ng sh ii2 d o4 ng i1 n d e2 i3 ia2 n x v4
介于      j ie4 v2
他们必须自己购买在澳洲居住期间的健康医疗保险       t a1 m e5 n b i4 x v1 z ii4 j i3 g ou4 m ai3 z ai4
ao4 zh ou1 j v1 zh u4 q i1 j ia1 n d i4 j ia4 n k a1 ng i1 l iao4 b ao3 x ia3 n
他们表示  t a1 m e5 n b iao3 sh ii4
他发现自己在夏天骑马的时候很多英国民众在睡觉       t a1 f a1 x ia4 n z ii4 j i3 z ai4 x ia4 t ia1 n m
```

```
a1      a1
ai1     ai1
ai2     ai2
ai3     ai3
ai4     ai4
an1     a1 n
an4     a4 n
ang1    a1 ng
ang2    a2 ng
ao1     ao1
ao2     ao2
ao4     ao4
ba      b
ba1     b a1
ba2     b a2
ba3     b a3
ba4     b a4
bai1    b ai1
bai2    b ai2
bai3    b ai3
bai4    b ai4
ban1    b a1 n
ban3    b a3 n
ban4    b a4 n
bang1   b a1 ng
bang3   b a3 ng
bang4   b a4 ng
bao1    b ao1
bao2    b ao2
bao3    b ao3
```

FIGURE: left is dictionary for Chinese character, right is dictionary for Chinese Pinyin

3.3. Acoustic Model / Training Data Set

There are two kinds of acoustic model in our project: 1. pretrained model, this model is provided by MFA tool knit, and the developers used GlobePhone to train this model. 2.use our data set to train our own acoustic model. In this case, we use 15 different audio files together with their text files and dictionary to train our own model.

Since the data set is different, these two different acoustic models have different effects. When we use Bible or SBS to test the pretrained model we find that we need to set beam to a extremely large number such as 8000~10000 or it can't output correct Textgrid file. Because of the large beam, it usually takes much more time to get the output, on average, the align time is 450s/file. Furthermore, sometimes even the beam is large enough, the textgrid file sometimes still has some problems like there are too many spaces in the file. Meanwhile, if we use Bible to train our own acoustic model and test it using Bible, the beam will be small, the time cost will reduce a lot, and the match rate of the audio file and text file will increase, which means the space will reduce. The same situation happens to SBS news. Thus, we draw a conclusion that for different data set, we need to train different acoustic model to output textgrid file.

We infer that the reason why we need to train different models for different data set is that the features of these data sets are different. In Bible, there is always only one people talking. Thus, the talking frequency is always the same. The frequency of the talker is slow. And some vocabulary is exclusive, so the generalization might be a problem. In The SBS news, there might multiple speakers talking in the same time, the , there may be noisy in the background, and the text is only an abstract of the audio.

3.4. Beam

In our project, we used GMM-HMM model to achieve the forced alignment, GMM-HMM is a mainstream and traditional in Automatic Speech Recognition. And this model would build a state-time trellis, we need to decode this sequences to get the final result. And the MFA toolkit uses beam search as decoder. So the beam size became a key problem in balancing the search errors and decoding speed.

If the beam size is too small, the searching space would be too narrow, which may get a wrong alignment. What's more, In general, MFA is on the more conservative side of things when it comes to alignment quality, so if there's a deviation between the transcript and the audio, alignment shouldn't be outputted. Therefore, finding a suitable beam size is important.

We did a test on 100 SBS resources. When the beam size was 10 and the retry beam was 40, we got 30 unaligned files. When the beam size was 100 and the retry beam was 200, we got 5 unaligned files. When the beam size was 200 and the retry beam was 400, we got 4 unaligned files.

What's more, we found that if we used a big beam size in training, we don't need to use such beam size in aligning, and the performance would be still great.

3.5. Others

Duration of the audio files is a key problem in aligning speed and success rate. The MFA recommend the audio segments should be less than 30 seconds for best performance. But our resources would be much longer than 30 secondes. Our average audio duration is 4~5 minutes. This also leads to a bigger beam size in aligning and much longer aligning time.

In other to reduce the duration of each file, we tried to split the original corpus into several mini corpus. However a new problem arised, it's hard to split the audio and text properly. We need to make sure each mini audio has a right transcript, otherwise it would be unaligned.

We used silence detection to split the audio, once there has a 1500 ms duration under 30 dB, we would consider it as a pause and make a cut. Meanwhile, we would split corresponding transcript into sentences. We used this policy to do some test. And the performance were really great on some cases which had a perfect partition, they can be aligned successfully in a short time. But the wrong partition cases were unaligned and there had so many unsuccessfully partition data.

4. Conclusions / Summary

(Resource finding: summary; shortcomings
Web Crawling: summary; shortcomings

Data Processing: factors that influence the performance)

5. **Future Work**

The challenge for the quality of our data crawler is still in the resources. Even though we found two working resources for the crawler, both of them exhibit the very characteristic problems. First, the SBS has a variety of speakers with different accents and wider range of vocabulary, especially the commonly-used vocabulary, while the Bible has only 1 single speaker for each language, which makes our processed data less general, and not good enough to train the acoustic model. Furthermore, the Bible uses so many traditional or old-fashioned words, which are rarely used nowadays. Another problem posed by the Bible is that the audio is so well-filtered that it only contains the voice of the speaker, while in training the good model, we need many different scenarios like background noise or music. Second, even though the SBS does not have limitations above like the Bible, its match rate is not consistent and low, which is opposite to the Bible, whose match rate is perfect, every word has its corresponding audio part. One of the future work here is to find better resources which satisfy two most important conditions: stable match rate and generalized scenarios.

Even though we did finally create the TextGrid files, it is in the future work that what we will be doing with these files. Our ideas is to train the ASR to see how much it could improve the current ASR for low-resource languages. Second idea is to create a new model based on these files. The problems with Montreal Forced Aligner is its pretrained models are not good enough and lack some pronunciations. Relying on the tool completely would yield a poor processed data. We want to use the TextGrid files to train new models, which has more vocabularies, pronunciations and different scenarios like noise to make the models more generalized.

As mentioned above, the beam poses the limit in decoding speed. We have a suggestion that slicing the audio/text pairs would significantly improve the speed of processing. However, this might work for very good match rate resources like Bible but for resources like the SBS, adjusting the beam length still works best.

6. **Acknowledgments**

Our project could not have been successful without help from DR. Shahab Jalalvand from Interactions LLC. Throughout 3 months of hard work, many problems pop up and Dr. Shahab has shown us how to solve them. None of the members in the team had experience in working with Automatic Speech Recognition, Natural Language Processing or Data Crawling but we have learnt so much from this project and from Dr. Shahab. We would want to send our best regards to him for his instruction and inspiration. In addition, we also want to thank Dr. Hana Godrich and Department of Electrical and Computer Engineering for their support and guidance.

7. References

[1] Lakomkin, Egor, et al. "KT-Speech-Crawler: Automatic Dataset Construction for Speech Recognition from YouTube Videos." *arXiv preprint arXiv:1903.00216* (2019).

[2] McAuliffe, Michael, et al. "Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi." *Interspeech*. 2017.

[3] https://montreal-forced-aligner.readthedocs.io/en/latest/index.html