Table of Contents

Project Overview

This project aims to generate a word level dataset to train models for ITN. The dataset obtained is in a similar format to the Google Text Normalisation dataset. The pipeline makes use of youtube video and first gets youtube captions which act as the written form text. The same youtube videos are then put into an ASR system to get the transcripts of these videos which is the text in spoken form. The two texts in written and spoken form are then broken down into word level and matched to each other, giving the final output.

Step 1: Data Crawling and Cleaning

Note: make sure to install the necessary dependencies as stated in the requirements file

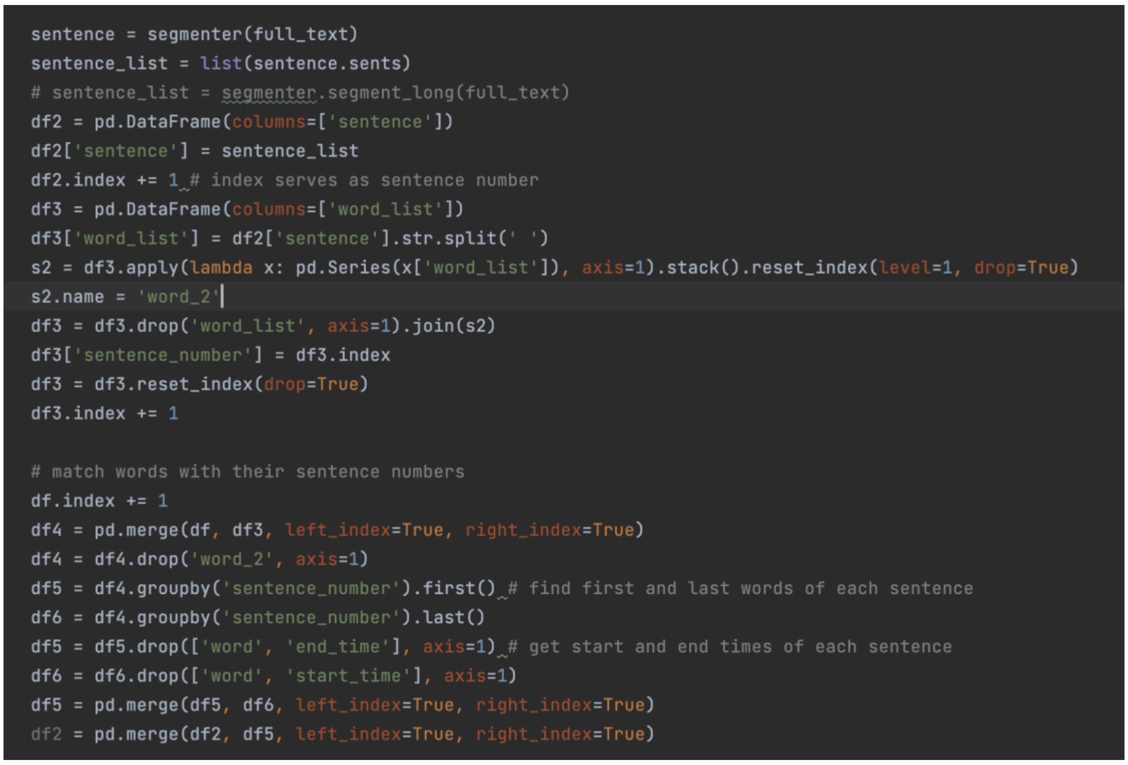
Script to run: ./crawl_youtube/youtube_crawler.sh 7 "<out_base_dir>" "<youtuber name>"

"<out_base_dir>": the directory to store the project on
"<youtuber name>": youtuber channel whose videos will be used. In this case, it is “Chris @HoneyMoneySG”

This step also does sentence segmentation on the youtube captions text file using this python file: ./get_vtt_and_clean/sentence_segmentation_with_times.py. However, the original sentence segmentor used, Deepsegment, is no longer supported and hence a different segmentor needs to be used. In this case, the segmentor used was “spacy”. These sentences are the **written form text**.

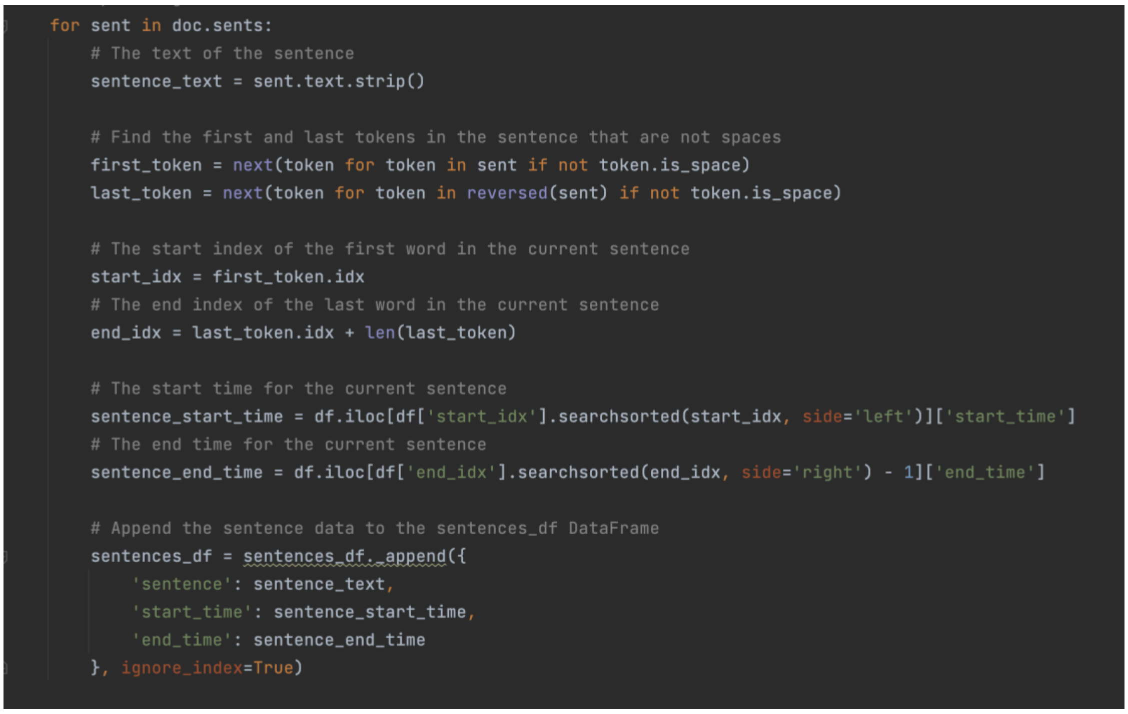
Additionally, in this same file “./get_vtt_and_clean/sentence_segmentation_with_times.py”, the original code under the sentence_segmentation function did not give the correct timestamps for each of the segmented sentences. Code was modified so that the sentence segmentation with times file now reflect the correct time stamps. This is important as the next step requires the time stamps to split the wav files accurately.

Old code:



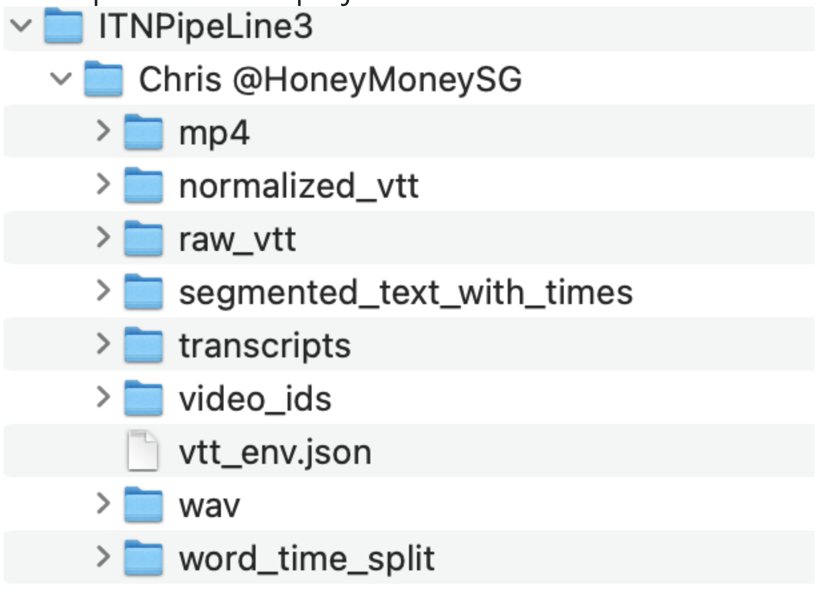
```
sentence = segmenter(full_text)
sentence_list = list(sentence.sents)
# sentence_list = segmenter.segment_long(full_text)
df2 = pd.DataFrame(columns=['sentence'])
df2['sentence'] = sentence_list
df2.index += 1 # index serves as sentence number
df3 = pd.DataFrame(columns=['word_list'])
df3['word_list'] = df2['sentence'].str.split(' ')
s2 = df3.apply(lambda x: pd.Series(x['word_list']), axis=1).stack().reset_index(level=1, drop=True)
s2.name = 'word_2'
df3 = df3.drop('word_list', axis=1).join(s2)
df3['sentence_number'] = df3.index
df3 = df3.reset_index(drop=True)
df3.index += 1

# match words with their sentence numbers
df.index += 1
df4 = pd.merge(df, df3, left_index=True, right_index=True)
df4 = df4.drop('word_2', axis=1)
df5 = df4.groupby('sentence_number').first() # find first and last words of each sentence
df6 = df4.groupby('sentence_number').last()
df5 = df5.drop(['word', 'end_time'], axis=1) # get start and end times of each sentence
df6 = df6.drop(['word', 'start_time'], axis=1)
df5 = pd.merge(df5, df6, left_index=True, right_index=True)
df2 = pd.merge(df2, df5, left_index=True, right_index=True)
```

New code:



```
for sent in doc.sents:
    # The text of the sentence
    sentence_text = sent.text.strip()

    # Find the first and last tokens in the sentence that are not spaces
    first_token = next(token for token in sent if not token.is_space)
    last_token = next(token for token in reversed(sent) if not token.is_space)

    # The start index of the first word in the current sentence
    start_idx = first_token.idx
    # The end index of the last word in the current sentence
    end_idx = last_token.idx + len(last_token)

    # The start time for the current sentence
    sentence_start_time = df.iloc[df['start_idx'].searchsorted(start_idx, side='left')]['start_time']
    # The end time for the current sentence
    sentence_end_time = df.iloc[df['end_idx'].searchsorted(end_idx, side='right') - 1]['end_time']

    # Append the sentence data to the sentences_df DataFrame
    sentences_df = sentences_df._append({
        'sentence': sentence_text,
        'start_time': sentence_start_time,
        'end_time': sentence_end_time
    }, ignore_index=True)
```

The following files should be in the project folder after running the entire step 1:
*ITNPipeLine3 is the project folder



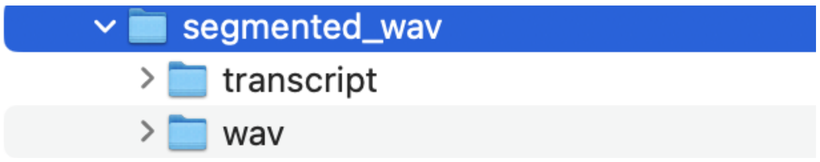
```
ITNPipeLine3
    Chris @HoneyMoneySG
        mp4
        normalized_vtt
        raw_vtt
        segmented_text_with_times
        transcripts
        video_ids
        vtt_env.json
        wav
        word_time_split
```

Step 2: Generate Spoken Form Transcript

First script to run: ./get_wenet_output/generate_separated_wav_transcript.sh "<out_base_dir>" "<youtuber name>"

"<out_base_dir>": Same as step 1
"<youtuber name>": Same as step 1

This script divides the wav files obtained in step 1 into sentence level according to the timestamps of each sentence recorded in the segmented_text_with_times folder generated from the previous step. The output will then be the segmented wav files and the transcripts for each of these segmented wav files. The final output folders are shown in the screenshot below:



Second script to run: ./get_wenet_output/gen_transcript_analyse_save.sh "<ASR_input>" " 8016 0 "<ASR_output>"
*note that the ASR system needs to be set up first before this script can be run

"<ASR_input>": folder containing the folder of segmented wav files
"<ASR_output>": output folder to hold the ASR transcripts

The transcript output from the ASR system will be the **spoken form text**.

Step 3: Train Model and Inference

This step focuses on matching each word for every sentence in spoken form to the corresponding word in the written form text. This is done through the use of a word alignment tool called Cross Align which is another project.

Before the script can be run, the env.json file needs to be updated with the correct files. This will be further explained later.

The google text normalisation dataset needs to first be downloaded as shown in the readme file.

First script to run: ./model_train_inference/generate_train_test_file.py --json ./model_train_inference/env.json

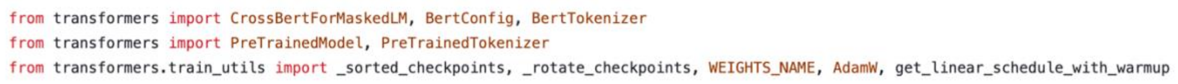
The first step in this script is to use the google text normalisation dataset and reformat it to fit the input format for Cross Align. The format is like this : (written form text) ||| (spoken form text)

The data from the google text normalisation dataset will be used to train the model for Cross Align.

The next step in this script is to format the written and spoken form texts from the youtube videos to match the Cross Align format too.

Next, follow the instructions from Cross Align to train the model to do word alignment for inverse text normalisation. Cross align was originally used to do word alignment for language translation but can be retrained for other tasks. It produces outputs in the widely-used i-j "Pharaoh format," where a pair i-j indicates that the i-th word (zero-indexed) of the source language is aligned to the j-th word of the target sentence. Use the formatted google text normalisation dataset to first train the model then run inference on the youtube dataset.

When using cross align, I discovered that there were some dependencies that were no longer supported. Particularly in the run_train.py file, the following did not work:



```
from transformers import CrossBertForMaskedLM, BertConfig, BertTokenizer
from transformers import PreTrainedModel, PreTrainedTokenizer
from transformers.train_utils import _sorted_checkpoints, _rotate_checkpoints, WEIGHTS_NAME, AdamW, get_linear_schedule_with_warmup
```

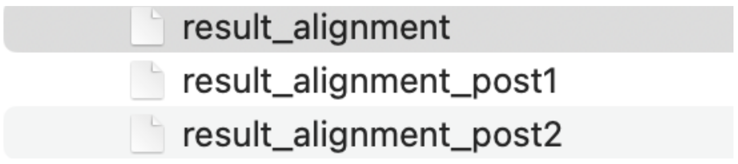
I then replaced them with the awesome_align package as shown below:

```
from awesome_align.train_utils import _sorted_checkpoints,
_rotate_checkpoints, WEIGHTS_NAME, AdamW, get_linear_schedule_with_warmup

from awesome_align import modeling
from awesome_align.configuration_bert import BertConfig
from awesome_align.modeling import BertForMaskedLM
from awesome_align.tokenization_bert import BertTokenizer
from awesome_align.tokenization_utils import PreTrainedTokenizer
from awesome_align.modeling_utils import PreTrainedModel
```

Note that training the Cross Align model requires the use of a GPU and cannot just be done on a CPU. As I only had my CPU to work with, I was not able to train the Cross Align model and used a different word alignment tool. I instead used BERT to do the word alignment instead. However, one issue with using BERT was that each word in the written form sentence can only be matched to one other word in the spoken form sentence. However, text like "2023" needs to be matched to a phrase "twenty twenty three" and hence using BERT lead to some inaccuracies in the data.

After obtaining the word alignment for each sentence, run the final script in the pipeline: ./model_train_inference/generate_dataset.py --json ./model_train_inference/env.json

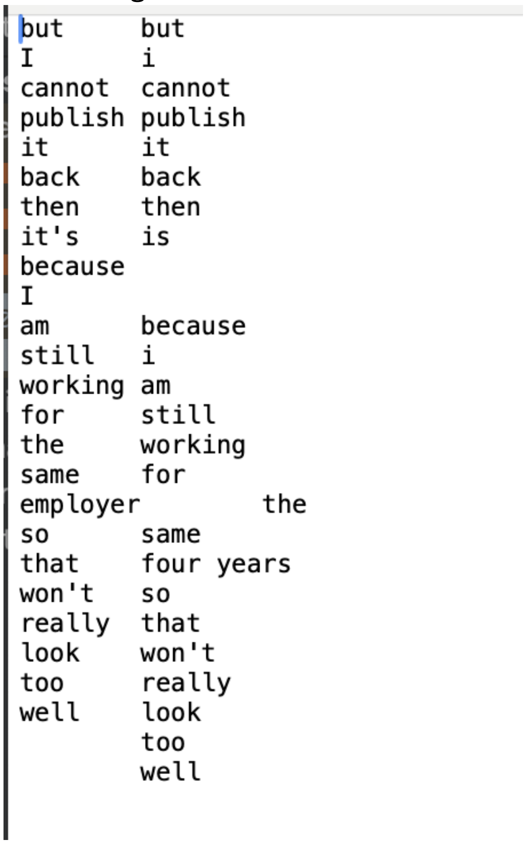
This script will reformat the outputs and split the sentences into word level similar to the format of the google text normalisation datset. The following 3 files should be obtained:



result_alignment
result_alignment_post1
result_alignment_post2

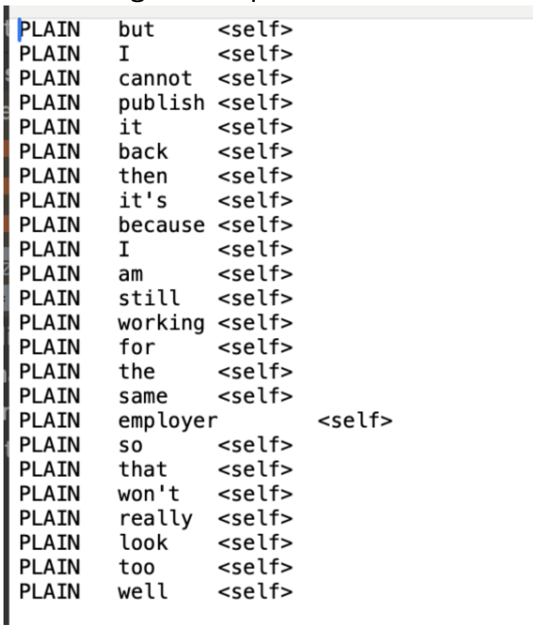
Examples of the contents of the three files are also shown below.

result_alignment:



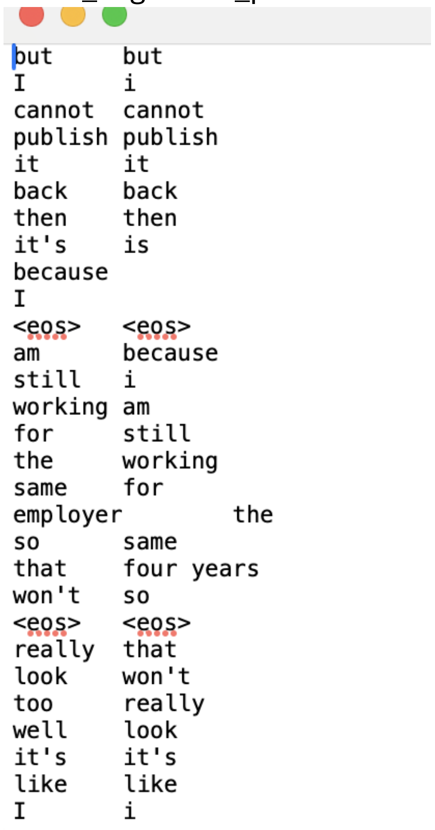
```
but      but
I        i
cannot   cannot
publish  publish
it       it
back     back
then     then
it's     is
because
I
am       because
still    i
working  am
for      still
the      working
same     for
employer         the
so       same
that     four years
won't    so
really   that
look     won't
too      really
well     look
         too
         well
```

result_alignment_post1:



```
PLAIN   but     <self>
PLAIN   I       <self>
PLAIN   cannot  <self>
PLAIN   publish <self>
PLAIN   it      <self>
PLAIN   back    <self>
PLAIN   then    <self>
PLAIN   it's    <self>
PLAIN   because <self>
PLAIN   I       <self>
PLAIN   am      <self>
PLAIN   still   <self>
PLAIN   working <self>
PLAIN   for     <self>
PLAIN   the     <self>
PLAIN   same    <self>
PLAIN   employer        <self>
PLAIN   so      <self>
PLAIN   that    <self>
PLAIN   won't   <self>
PLAIN   really  <self>
PLAIN   look    <self>
PLAIN   too     <self>
PLAIN   well    <self>
```

result_alignment_post2:



```
but     but
I       i
cannot  cannot
publish publish
it      it
back    back
then    then
it's    is
because
I
<eos>   <eos>
am      because
still   i
working am
for     still
the     working
same    for
employer        the
so      same
that    four years
won't   so
<eos>   <eos>
really  that
look    won't
too     really
well    look
it's    it's
like    like
I       i
```

As mentioned previously, the env.json needs to be updated with the correct files before running any of the scripts from this step. The files that need to be updated are shown below:

```
"generate_train_test_file": {
   "infile_dir_path": "file path to the downloaded google text
normalisation dataset particularly the en_with_types file",
   "outfile_dir_path": "output folder for the reformatted google text
normalisation dataset that will be used to train Cross Align",
   "wenet_dir": “output from the ASR system",
   "youtube_dir": “folder with the segmented wav transcripts and wav
files",
   "out_dir": “output folder for the formatted written and spoken form
sentences from the youtube videos that Cross Align will run inference on”
},
"generate_dataset": {
   "data_dir_path": “folder containing the file with the input to the Cross
Align model",
   "test_data": “data used to inference the cross align model",
   "out_alignment": “file containing the word alignment output from the
Cross Align model",
   "output_name": "result_alignment (name of the outputs)"
},
```