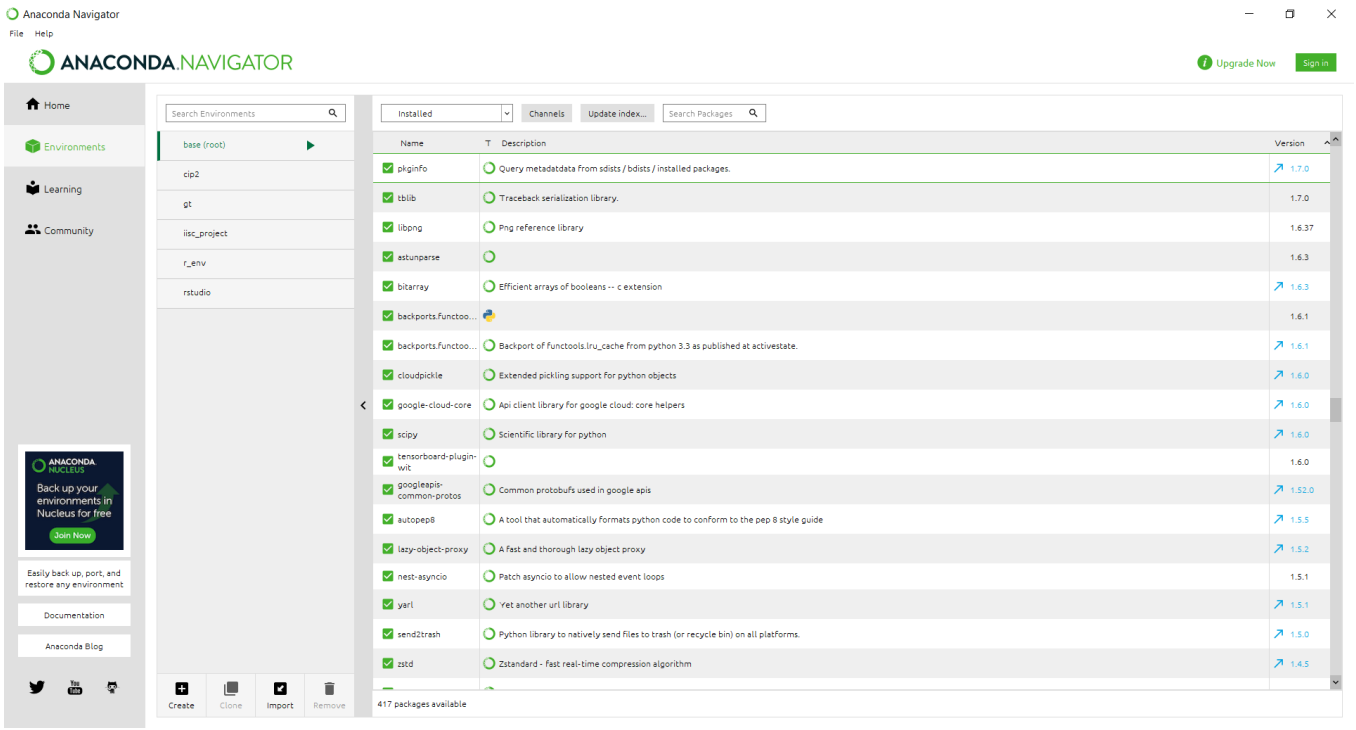
NLP Midsemester

V.S.Suryaa

2018103610

P-Batch

Environment used:



Step 1: (Q1)

Using the downloaded video, we generate “converted.wav” that has only the audio without video. Using the extracted audio, we convert it into text file named “recog.txt”.

In [1]:
```
import speech_recognition as sr
import moviepy.editor as mp
```

In [2]:
```
clip = mp.VideoFileClip("video1.mp4")

clip.audio.write_audiofile("converted.wav")
```

```
chunk:   0%|
MoviePy - Writing audio in converted.wav

MoviePy - Done.
```

In [3]:
```
r = sr.Recognizer()
audio = sr.AudioFile("converted.wav")
```

In [4]:
```
with audio as source:
    audio_file = r.record(source)
result = r.recognize_google(audio_file)
```

In [5]:
```
# exporting the result
with open('recog.txt',mode ='w') as file:
    file.write("\n")
    file.write(result)
    print("ready!")
```

```
ready!
```

recog.txt:

recog - Notepad

File Edit Format View Help

excuse me do you mind everywhere else is

full not at all

I'm wrong by the way Ron Weasley I'm

Harry Harry Potter so so it's true I

mean do you really have the the the what

wicked

anything off the trolley dears no thanks

I'm all set

we'll take the lot

whoa

first of all searches take the beans

they mean every flavor there's chocolate

and peppermint and there's also spinach

liver and tripe George sweat he got a

bogey flavored one month

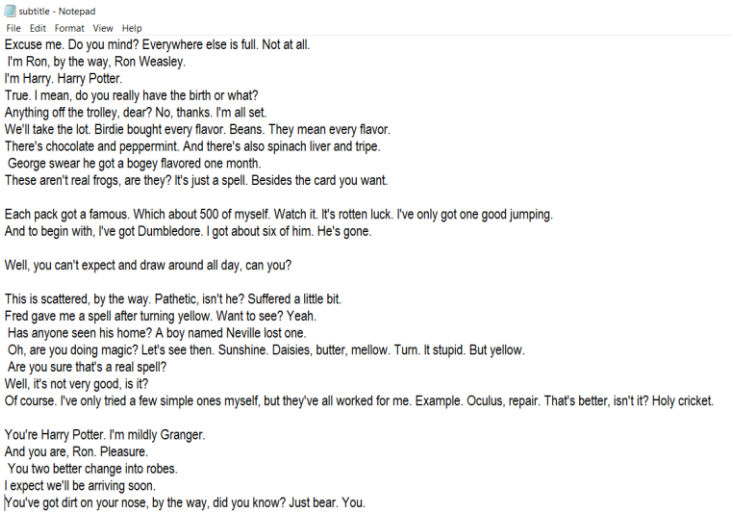
these aren't real frogs are they it's

just a spell besides it's the cards you

want each packs got a famous what was it

video1.mp4:

subtitle.txt:


subtitle - Notepad
File Edit Format View Help
Excuse me. Do you mind? Everywhere else is full. Not at all.
I'm Ron, by the way, Ron Weasley.
I'm Harry. Harry Potter.
True. I mean, do you really have the birth or what?
Anything off the trolley, dear? No, thanks. I'm all set.
We'll take the lot. Birdie bought every flavor. Beans. They mean every flavor.
There's chocolate and peppermint. And there's also spinach liver and tripe.
George swear he got a bogey flavored one month.
These aren't real frogs, are they? It's just a spell. Besides the card you want.

Each pack got a famous. Which about 500 of myself. Watch it. It's rotten luck. I've only got one good jumping.
And to begin with, I've got Dumbledore. I got about six of him. He's gone.

Well, you can't expect and draw around all day, can you?

This is scattered, by the way. Pathetic, isn't he? Suffered a little bit.
Fred gave me a spell after turning yellow. Want to see? Yeah.
Has anyone seen his home? A boy named Neville lost one.
Oh, are you doing magic? Let's see then. Sunshine. Daisies, butter, mellow. Turn. It stupid. But yellow.
Are you sure that's a real spell?
Well, it's not very good, is it?
Of course. I've only tried a few simple ones myself, but they've all worked for me. Example. Oculus, repair. That's better, isn't it? Holy cricket.

You're Harry Potter. I'm mildly Granger.
And you are, Ron. Pleasure.
You two better change into robes.
I expect we'll be arriving soon.
You've got dirt on your nose, by the way, did you know? Just bear. You.

Step2: (Q2 and Q3)

Calculating cosine similarity between subtitle.txt and recog.txt.

Preprocessing of text is done before using to calculate cosine similarity.

In [1]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Suryaa\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[1]:
```
True
```

In [2]:
```
sub_file = open("subtitle.txt", "r")
sub = sub_file.read().lower()
sub_file.close()
print(sub)
```

```
excuse me. do you mind? everywhere else is full. not at all. i'm ron, by th
ey, dear? no, thanks. i'm all set. we'll take the lot. birdie bought every
ge swear he got a bogey flavored one month. these aren't real frogs, are th

each pack got a famous. which about 500 of myself. watch it. it's rotten lu

well, you can't expect and draw around all day, can you?

this is scattered, by the way. pathetic, isn't he? suffered a little bit. f
h, are you doing magic? let's see then. sunshine. daisies, butter, mellow.
ied a few simple ones myself, but they've all worked for me. example. oculu

you're harry potter. i'm mildly granger. and you are, ron. pleasure. you tw
st bear. you.
```

In [3]:
```
recog_file = open("recog.txt", "r")
recog = recog_file.read().lower()
recog_file.close()
print(recog)
```

```
excuse me do you mind everywhere else is

full not at all
```

In [4]:

```
# tokenization
X_list = word_tokenize(sub)
Y_list = word_tokenize(recog)

# sw contains the list of stopwords
sw = stopwords.words('english')
l1 =[];l2 =[]

# remove stop words from the string
X_set = {w for w in X_list if not w in sw}
Y_set = {w for w in Y_list if not w in sw}

# form a set containing keywords of both strings
rvector = X_set.union(Y_set)
for w in rvector:
    if w in X_set: l1.append(1) # create a vector
    else: l1.append(0)
    if w in Y_set: l2.append(1)
    else: l2.append(0)
c = 0

# cosine formula
for i in range(len(rvector)):
        c+= l1[i]*l2[i]
cosine = c / float((sum(l1)*sum(l2))**0.5)
print("similarity: ", cosine)
```

```
similarity:  0.7872057314294038
```

Cosine similarity score obtained: 0.7872

Step 3: (Q4)

Training the audio text to generate new text.

In [1]:

```
import numpy
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense, Dropout, GRU
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
import keras.utils as ku

import pandas as pd
import numpy as np
import string, os

import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)
filename = "subtitle.txt"
raw_text = open(filename, 'r').read()
```

```
Using TensorFlow backend.
```

In [2]:

```
def clean_text(txt):
    txt = "".join(v for v in txt if v not in string.punctuation).lower()
    txt = txt.encode("utf8").decode("ascii",'ignore')
    return txt
raw_text = clean_text(raw_text)
```

In [3]:

```
tokenizer = Tokenizer()
def get_sequence_of_tokens(corpus):
    tokenizer.fit_on_texts(corpus)
    total_words = len(tokenizer.word_index) + 1

    ## convert data to sequence of tokens
    input_sequences = []
    for line in corpus:
        token_list = tokenizer.texts_to_sequences([line])[0]
        for i in range(1, len(token_list)):
            n_gram_sequence = token_list[:i+1]
            input_sequences.append(n_gram_sequence)
```

```
            n_gram_sequence = token_list[:i+1]
            input_sequences.append(n_gram_sequence)
    return input_sequences, total_words
inp_sequences, total_words = get_sequence_of_tokens(raw_text.split('\n\n'))
```

In [4]:

```
def generate_padded_sequences(input_sequences):
    max_sequence_len = max([len(x) for x in input_sequences])
    input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre

    predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
    label = ku.to_categorical(label, num_classes=total_words)
    return predictors, label, max_sequence_len

predictors, label, max_sequence_len = generate_padded_sequences(inp_sequences)
```

In [5]:

```
def create_model(max_sequence_len, total_words):
    input_len = max_sequence_len - 1
    model = Sequential()

    # Add Input Embedding Layer
    model.add(Embedding(total_words, 10, input_length=input_len))

    # Add Hidden Layer 1 - LSTM Layer
    model.add(GRU(128))
    model.add(Dropout(0.1))
    model.add(Dense(256, activation='relu'))
    # Add Output Layer
    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Model summary:

```
model = create_model(max_sequence_len, total_words)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 91, 10)            1740
_________________________________________________________________
gru_1 (GRU)                  (None, 128)               53376
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_2 (Dense)              (None, 174)               44718
=================================================================
Total params: 132,858
Trainable params: 132,858
Non-trainable params: 0
_________________________________________________________________
```

Training the model:

In [6]:
```
model.fit(predictors, label, epochs=50, verbose=1)
```

```
Epoch 1/50
264/264 [==============================] - 1s 3ms/step - loss: 5.1601 - accuracy: 0.0114
Epoch 2/50
264/264 [==============================] - 1s 2ms/step - loss: 5.1514 - accuracy: 0.0379
Epoch 3/50
264/264 [==============================] - 0s 2ms/step - loss: 5.1397 - accuracy: 0.0417
Epoch 4/50
264/264 [==============================] - 0s 2ms/step - loss: 5.0918 - accuracy: 0.0417
Epoch 5/50
264/264 [==============================] - 0s 2ms/step - loss: 5.0239 - accuracy: 0.0417
Epoch 6/50
264/264 [==============================] - 0s 2ms/step - loss: 5.0146 - accuracy: 0.0417
Epoch 7/50
264/264 [==============================] - 0s 2ms/step - loss: 4.9725 - accuracy: 0.0417
Epoch 8/50
264/264 [==============================] - 0s 2ms/step - loss: 4.9240 - accuracy: 0.0417
Epoch 9/50
264/264 [==============================] - 0s 2ms/step - loss: 4.8959 - accuracy: 0.0417
Epoch 10/50
264/264 [==============================] - 0s 2ms/step - loss: 4.8829 - accuracy: 0.0417
Epoch 11/50
264/264 [==============================] - 0s 2ms/step - loss: 4.8654 - accuracy: 0.0417
Epoch 12/50
264/264 [==============================] - 0s 2ms/step - loss: 4.8417 - accuracy: 0.0417
Epoch 13/50
264/264 [==============================] - 1s 2ms/step - loss: 4.8164 - accuracy: 0.0417
Epoch 14/50
264/264 [==============================] - 0s 2ms/step - loss: 4.7828 - accuracy: 0.0417
Epoch 15/50
264/264 [==============================] - 0s 2ms/step - loss: 4.7443 - accuracy: 0.0455
Epoch 16/50
264/264 [==============================] - 0s 2ms/step - loss: 4.6960 - accuracy: 0.0417
Epoch 17/50
264/264 [==============================] - 0s 2ms/step - loss: 4.6356 - accuracy: 0.0455
Epoch 18/50
264/264 [==============================] - 0s 2ms/step - loss: 4.5720 - accuracy: 0.0530
Epoch 19/50
264/264 [==============================] - 0s 2ms/step - loss: 4.4900 - accuracy: 0.0758
Epoch 28/50
264/264 [==============================] - 0s 2ms/step - loss: 2.9113 - accuracy: 0.2614
Epoch 29/50
264/264 [==============================] - 0s 2ms/step - loss: 2.7658 - accuracy: 0.2311
Epoch 30/50
264/264 [==============================] - 0s 2ms/step - loss: 2.6271 - accuracy: 0.3106
Epoch 31/50
264/264 [==============================] - 0s 2ms/step - loss: 2.4267 - accuracy: 0.3030
Epoch 32/50
264/264 [==============================] - 1s 2ms/step - loss: 2.2354 - accuracy: 0.4583
Epoch 33/50
264/264 [==============================] - 0s 2ms/step - loss: 2.0022 - accuracy: 0.4735
Epoch 34/50
264/264 [==============================] - 0s 2ms/step - loss: 1.8149 - accuracy: 0.5189
Epoch 35/50
264/264 [==============================] - 0s 2ms/step - loss: 1.6734 - accuracy: 0.5644
Epoch 36/50
264/264 [==============================] - 1s 2ms/step - loss: 1.5930 - accuracy: 0.5985
Epoch 37/50
264/264 [==============================] - 0s 2ms/step - loss: 1.4305 - accuracy: 0.6439
Epoch 38/50
264/264 [==============================] - 0s 2ms/step - loss: 1.3126 - accuracy: 0.6780
Epoch 39/50
264/264 [==============================] - 0s 2ms/step - loss: 1.1578 - accuracy: 0.7462
Epoch 40/50
264/264 [==============================] - 1s 2ms/step - loss: 1.0610 - accuracy: 0.7614
Epoch 41/50
264/264 [==============================] - 0s 2ms/step - loss: 0.9906 - accuracy: 0.7727
Epoch 42/50
264/264 [==============================] - 0s 2ms/step - loss: 0.8736 - accuracy: 0.8258
Epoch 43/50
264/264 [==============================] - 0s 2ms/step - loss: 0.7905 - accuracy: 0.8333
Epoch 44/50
264/264 [==============================] - 0s 2ms/step - loss: 0.7401 - accuracy: 0.8636
Epoch 45/50
264/264 [==============================] - 0s 2ms/step - loss: 0.6554 - accuracy: 0.8636
Epoch 46/50
264/264 [==============================] - 0s 2ms/step - loss: 0.6322 - accuracy: 0.8712
Epoch 47/50
264/264 [==============================] - 0s 2ms/step - loss: 0.5816 - accuracy: 0.9167
Epoch 48/50
264/264 [==============================] - 0s 2ms/step - loss: 0.5107 - accuracy: 0.9129
Epoch 49/50
264/264 [==============================] - 0s 2ms/step - loss: 0.4569 - accuracy: 0.9280
Epoch 50/50
264/264 [==============================] - 0s 2ms/step - loss: 0.4524 - accuracy: 0.9394
```

]: <keras.callbacks.callbacks.History at 0x28355a27148>

Predicting new text using trained model:

In [7]:
```
def generate_text(seed_text, next_words, model, max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
        predicted = model.predict_classes(token_list, verbose=0)

        output_word = ""
        for word,index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " "+output_word
    return seed_text.title()
```

In [11]:
```
print (generate_text("This is", 5, model, max_sequence_len))
print (generate_text("Potter", 5, model, max_sequence_len))
print (generate_text("can", 5, model, max_sequence_len))
```

```
This Is Scattered By The Way Pathetic
Potter Im Ron By The Way
Can You Cant Ron Pleasure You
```

Step 4: (Q5)

The predicted text is used to generate audio file.

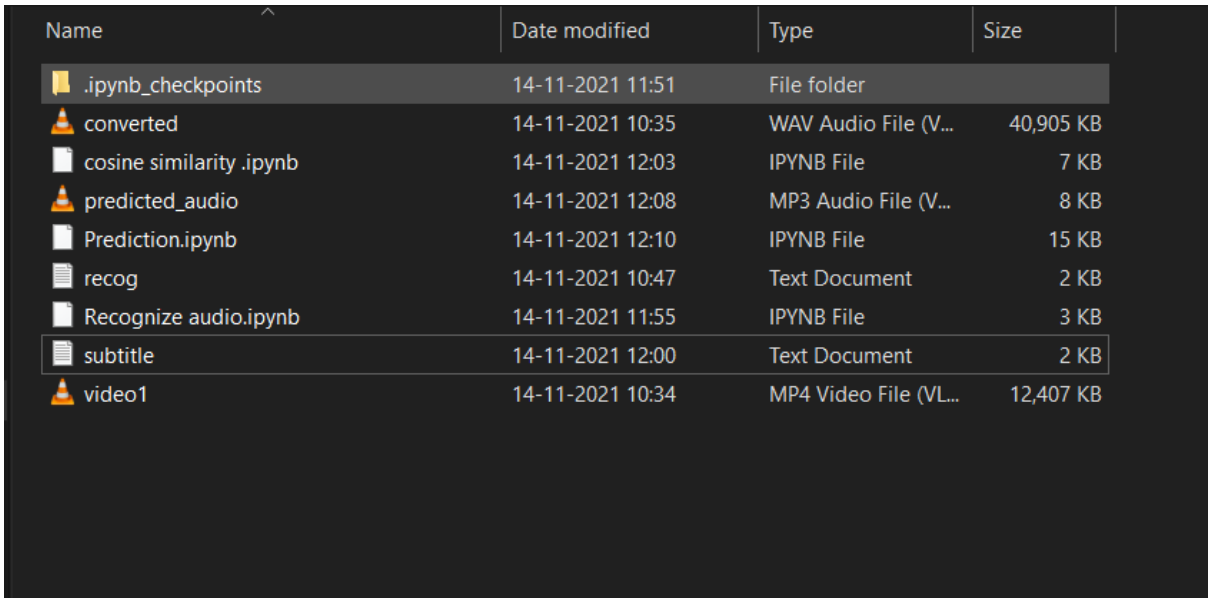
In [12]:
```
!pip install gTTS
from gtts import gTTS
import os

mytext = generate_text("Potter", 3, model, max_sequence_len)
language = 'en'

myobj = gTTS(text=mytext, lang=language, slow=False)

myobj.save("predicted_audio.mp3")
```

```
Requirement already satisfied: gTTS in c:\users\suryaa\anaconda3\lib\site-packages (2.2.3)
Requirement already satisfied: click in c:\users\suryaa\anaconda3\lib\site-packages (from gTTS) (7.1.2)
Requirement already satisfied: requests in c:\users\suryaa\anaconda3\lib\site-packages (from gTTS) (2.25.1)
Requirement already satisfied: six in c:\users\suryaa\anaconda3\lib\site-packages (from gTTS) (1.15.0)
Requirement already satisfied: idna<3,>=2.5 in c:\users\suryaa\anaconda3\lib\site-packages (from requests->gTTS) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\suryaa\anaconda3\lib\site-packages (from requests->gTTS) (1.26.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\suryaa\anaconda3\lib\site-packages (from requests->gTTS) (2021.10.8)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\users\suryaa\anaconda3\lib\site-packages (from requests->gTTS) (4.0.0)
```



The predicted audio is available in predicted_audio.mp3