Exercise2_Mert_Erol_20_915_245

October 20, 2024

1 Excercise 2 (Speech Technology course, HS 2024)

1.1 Instructions

- Make sure you have uploaded the audio files to Google Drive. Even though we won't use all of them in this exercise, it will be required to use later.
- Please read the markdown sections, and code comments carefully before answering.
- You are required to treat `...` as incomplete code, which you are required to complete.
- Each incomplete region marked by `...` can be completed with a maximum of 2 statements (2 lines of code in Python).
- You may refer to the slides and reference material, but may not use AI code completion.
- Run all cells in the notebook even if it does not require any answer from your part.
- The point for each section or sub-section is given in square brackets. E.g [15 pt] means 15 points.
- Pay clear attention to Q. & A. questions. The markdown-python cell separation is not always obvious.
- **ATTENTION**: There are many places where the path of the audio file needs to be fixed by you.

[2]:
```
# import pytorch and os
import numpy as np
import torch
import os
from torch import nn

torch.manual_seed(42)
```

[2]:
```
<torch._C.Generator at 0x7fce9c7b07b0>
```

[3]:
```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

2 Import data from Google drive

[4]:
```
os.listdir('/content/drive/MyDrive/')
```

```
[4]: ['Testologie.gslides',
 'Biologie.gdoc',
 '_',
 'DeutschUfsatz.gdoc',
 'Présentation.mov',
 '+.zip',
 'vitamin-k-h5osxzrb0lak.zip',
 'Alexander und Johannes.gdoc',
 'Instrumentenkunde.zip',
 'Firefox_Setup_54.0.1__1_.exe',
 'Debatte.gdoc',
 'FRZ-Koche_RAW.mp4',
 'Recht_02_Einführung_Privatrecht_Aufgaben_Schülerversion_RO_20180116.docx',
 'Music Own Song.gdoc',
 'gerechter krieg.pdf',
 'Philo',
 'English Slam.gslides',
 'Streaming Budget.gsheet',
 'Unbenannte Tabelle.gsheet',
 'Kopie von Nathan der weisi backup.gdoc',
 'Nathan der weise.gdoc',
 '3cW_BWL_C2 Strategietypen und Businessplan_Aufgaben_1 (1).docx',
 '3cW_Einführung Strategie_Aufgaben.docx',
 '3cW_C1 Einführung Strategie_Lehrerversion_SuS.pptx',
 '3cW_BWL_C2 Strategietypen und Businessplan_Aufgaben_1 (1).gdoc',
 '3cW_C1 Einführung Strategie_Lehrerversion_SuS.gslides',
 '3cW_Einführung Strategie_Aufgaben.gdoc',
 '3cW_Erbrecht_Aufgaben.docx',
 '3cW_Erbrecht_SuS.pdf',
 '3cW_Erbrecht_Aufgaben.gdoc',
 'Unbenanntes Dokument (3).gdoc',
 '.minecraft.zip (Unzipped Files)',
 'Notability (2)',
 'Notability (1)',
 'Impfdashboard POC.gdoc',
 'Global Epidemics POC.gdoc',
 'Aufgaben_Geld.docx',
 'Geldbeträge legen… Lösungen.gdoc',
 'Multiplizieren.gdoc',
 'Szene ish Sylwester.gslides',
 'Längen&Flächen.gdoc',
 'Subtrahieren & Addieren.gdoc',
 'Booking.com Barca.gdoc',
 'Unbenanntes Dokument (2).gdoc',
 'mods',
 'Barca.gdoc',
 'Brief.gdoc',
```

```
 'Lebenslauf.gdoc',
 'Drop the ship',
 'Dani Brief 3.0.gdoc',
 'Dani Rechtschutz 2.0.gdoc',
 'Notability',
 'Unbenanntes Dokument (1).gdoc',
 'Capsim Decisions.gdoc',
 'Capsim Group 3.gdoc',
 'Uni Grades and shit.gsheet',
 'Unbenanntes Dokument.gdoc',
 'ECL1 Ex01.gdoc',
 'LargeLanguageMiners MC&Discord server.gdoc',
 'Kopie von Zapier: Outlook signature templates\n.gdoc',
 'vcard.vcf',
 'pixel_mert.png',
 'librispeech',
 'Colab Notebooks',
 'Eseltritt HS24']
```

[ ]:
```
# running this breaks the rest
#from google.colab import drive
#drive.mount('/content/drive/MyDrive/librispeech')      # <-- add the path to␣
 ↪your drive
```

[5]:
```
# Print the list of audio files in the librispeech folder that will be used in␣
 ↪this exercise.
# Make sure the necessary files are available.
! ls '/content/drive/MyDrive/librispeech'    # change the path to your correct␣
 ↪path
```

```
1089-134686-0000_4k.wav  1089-134686-0000.wav  1089-134686-0004.wav
1089-134691-0003.wav
1089-134686-0000.txt     1089-134686-0001.wav  1089-134691-0000.wav
1089-134691-0004.wav
1089-134686-0000_v2.wav  1089-134686-0002.wav  1089-134691-0001.wav
1089-134686-0000_v3.wav  1089-134686-0003.wav  1089-134691-0002.wav
```

---

3 Section 1: Audio data analysis

In this section, we will load the audio files, get the sampling rate, and calculate the number of frames for an audio.

3.0.1 Test Audio manipulation with pydub [5pt]

[6]:
```
# Install pydub here
!pip install pydub
```

```
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub
Successfully installed pydub-0.25.1
```

[7]:
```
# Import AudioSegment from pydub
from pydub import AudioSegment
```

[8]:
```
# load the 1089-134686-0000.wav from the librispeech folder. Change the path
# according to where the files are located in your drive.
filename = '/content/drive/MyDrive/librispeech/1089-134686-0000.wav'    # <---␣
↪path to the audio file
```

[9]:
```
# load the audio file pointed by the variable 'filename' here with AudioSegment␣
↪from pydub
audio = AudioSegment.from_file(filename, 'wav')
```

[10]:
```
# Complete the next two statements so that we
# can print the duration and number of samples
duration_seconds = audio.duration_seconds
sampling_rate = audio.frame_rate

print("Total duration: {}".format(duration_seconds))
print("Sampling rate is {}".format(sampling_rate))
```

```
Total duration: 10.435
Sampling rate is 16000
```

3.0.2 Q & A

Q. Write the formula (in this markdown cell) to obtain the number of samples in an audio file given the sampling rate and duration in seconds. Verify the formula with the above 3 values.

A. Number of Samples=Sampling Rate*Duration (seconds)

[11]:
```
# use the above formula to compute the number of samples for the above audio␣
↪file
# from the sampling_rate and duration
num_samples_computed: int = 166960

# 16000*10.435
```

[12]:
```
assert isinstance(num_samples_computed, int), "Make sure num_samples_computed␣
↪is an integer"
```

[13]:
```
# import Audio and display from ipython display and play the audio here
from IPython.display import Audio, display
display(Audio(filename))
```

```
<IPython.lib.display.Audio object>
```

3.1 Verify and resample if necessary [10 pt]

3.1.1 Q & A

Q. I have a speech processing system (e.g. ASR) that is designed to process audio at 16 kHz. I now have two audio files (1) file1.wav sampled at 32 kHz, and (2) file2.wav sampled at 8 kHz. What pre-processing step is necessary before I can use the system with these two files?

A. Enter your answer here. Downsample file1.wav to 16 kHz and upsample file2.wav to 16 kHz

---

Consider the three files in your folder:

1. 1089-134686-0000.wav
2. 1089-134686-0000_v2.wav
3. 1089-134686-0000_v3.wav

Make sure that all the three files are in the same sampling rate at 16 kHz (i.e. 16'000). If not, convert them to 16 kHz. You can use Pydub as shown in class, or you can use any of your favorite library.

[14]:
```
# complete the paths of the file below
file1 = '/content/drive/MyDrive/librispeech/1089-134686-0000.wav'
file2 = '/content/drive/MyDrive/librispeech/1089-134686-0000_v2.wav'
file3 = '/content/drive/MyDrive/librispeech/1089-134686-0000_v3.wav'
```

[15]:
```
for f in [file1, file2, file3]:
  # load the audio with AudioSegment
  audio = AudioSegment.from_file(f, "wav")
  sample_rate = audio.frame_rate
  if sample_rate != 16_000:
    # resample the audio and reassign the variable to 'audio'
    audio = audio.set_frame_rate(16000)
  # complete the test below to prove that the sampling rate has changed
  assert audio.frame_rate == 16_000, "Sampling rate has not changed!!"
```

3.2 Antialiasing example [5 pt]

Downsample file1 (`1089-134686-0000.wav`) to 4 kHz. Play the audio and note the difference in a maximum of two sentences.

[16]:
```
# you may copy the code from the previous cell to load the audio
audio = AudioSegment.from_file("/content/drive/MyDrive/librispeech/
↪1089-134686-0000.wav", "wav")  # load the audio with AudioSegment
sample_rate = audio.frame_rate  # get the sample_rate
desired_sample_rate = 4000

# downsample the audio to 4000 and save it as 1089-134686-0000_4k.wav in your␣
↪drive
audio = audio.set_frame_rate(4000)  # downsample and reassign it to the audio␣
↪variable

# save it to Google drive (see notebook from Lecture 2)
audio.export("/content/drive/MyDrive/librispeech/1089-134686-0000_4k.wav",␣
↪format='wav')
```

[16]:
```
<_io.BufferedRandom
name='/content/drive/MyDrive/librispeech/1089-134686-0000_4k.wav'>
```

[17]:
```
# display and listen to the downsampled audio
filename = "/content/drive/MyDrive/librispeech/1089-134686-0000_4k.wav"
display(Audio(filename))
```

```
<IPython.lib.display.Audio object>
```

Q. What do you think about the audio when downsampled to 4 kHz?

A. The voice seems a bit deeper while also being kinda distorted and muffled

---

3.3 Load all audio files in google drive [10 pt]

[18]:
```
# Now, let's load the all the audio files in the librispeech folder
audio_folder_path = '/content/drive/MyDrive/librispeech'   # <--- path to a␣
↪folder containing audio files
```

[19]:
```
def load_audio_files(folder_path):
  """Loads audio files from a given folder using pydub.

  Args:
    folder_path: The path to the folder containing audio files.

  Returns:
    A list of AudioSegment objects, one for each audio file in the folder.
  """
```

```
    audio_files = []
    for filename in os.listdir(folder_path):
      if filename.endswith("wav"):  # Add the extension of the filenames and add␣
↪the conditions for other extensions if necessary
        file_path = os.path.join(folder_path, filename)   # get the whole path of␣
↪the filename
        audio = AudioSegment.from_file(file_path, "wav")  # load the audio file␣
↪here
        audio_files.append(audio)  # add the audio to the list
    return audio_files
```

[20]:
```
# load the audio files
audio_files = load_audio_files(audio_folder_path)

# check how many files were loaded
print(len(audio_files))
```

```
13
```

3.4 Section 2: Pytorch practice [15 pt]

- Create a random one dimensional tensor with 100 elements

Perform the following operations on the tensor: - Square each element - Add all elements to get the sum all elements. Use only operation on the tensor

- Combine the above two operations i.e. square all elements, then add all values, then take square root to get energy value given a tensor
- Create a random two dimensional 100x160 tensor
  - Perform the same operations on the columns to get 100 energy values.
  - Take log on the 100 values to get log energy values

[21]:
```
# create a random one-dimensional tensor with 100 elements
a = random_tensor = torch.rand(100)
a
```

[21]:
```
tensor([0.8823, 0.9150, 0.3829, 0.9593, 0.3904, 0.6009, 0.2566, 0.7936, 0.9408,
        0.1332, 0.9346, 0.5936, 0.8694, 0.5677, 0.7411, 0.4294, 0.8854, 0.5739,
        0.2666, 0.6274, 0.2696, 0.4414, 0.2969, 0.8317, 0.1053, 0.2695, 0.3588,
        0.1994, 0.5472, 0.0062, 0.9516, 0.0753, 0.8860, 0.5832, 0.3376, 0.8090,
        0.5779, 0.9040, 0.5547, 0.3423, 0.6343, 0.3644, 0.7104, 0.9464, 0.7890,
        0.2814, 0.7886, 0.5895, 0.7539, 0.1952, 0.0050, 0.3068, 0.1165, 0.9103,
        0.6440, 0.7071, 0.6581, 0.4913, 0.8913, 0.1447, 0.5315, 0.1587, 0.6542,
        0.3278, 0.6532, 0.3958, 0.9147, 0.2036, 0.2018, 0.2018, 0.9497, 0.6666,
        0.9811, 0.0874, 0.0041, 0.1088, 0.1637, 0.7025, 0.6790, 0.9155, 0.2418,
        0.1591, 0.7653, 0.2979, 0.8035, 0.3813, 0.7860, 0.1115, 0.2477, 0.6524,
        0.6057, 0.3725, 0.7980, 0.8399, 0.1374, 0.2331, 0.9578, 0.3313, 0.3227,
        0.0162])
```

[22]:
```python
# square each element of that tensor
a_sq = a*a
a_sq
```

[22]:
```
tensor([7.7840e-01, 8.3723e-01, 1.4658e-01, 9.2027e-01, 1.5245e-01, 3.6108e-01,
        6.5829e-02, 6.2987e-01, 8.8505e-01, 1.7738e-02, 8.7347e-01, 3.5234e-01,
        7.5586e-01, 3.2230e-01, 5.4922e-01, 1.8439e-01, 7.8401e-01, 3.2937e-01,
        7.1065e-02, 3.9369e-01, 7.2701e-02, 1.9480e-01, 8.8162e-02, 6.9170e-01,
        1.1091e-02, 7.2627e-02, 1.2875e-01, 3.9746e-02, 2.9942e-01, 3.7951e-05,
        9.0546e-01, 5.6650e-03, 7.8502e-01, 3.4013e-01, 1.1401e-01, 6.5444e-01,
        3.3400e-01, 8.1718e-01, 3.0765e-01, 1.1718e-01, 4.0239e-01, 1.3279e-01,
        5.0471e-01, 8.9569e-01, 6.2257e-01, 7.9194e-02, 6.2194e-01, 3.4747e-01,
        5.6839e-01, 3.8122e-02, 2.5460e-05, 9.4138e-02, 1.3570e-02, 8.2859e-01,
        4.1476e-01, 5.0000e-01, 4.3314e-01, 2.4138e-01, 7.9442e-01, 2.0951e-02,
        2.8247e-01, 2.5195e-02, 4.2795e-01, 1.0746e-01, 4.2668e-01, 1.5668e-01,
        8.3667e-01, 4.1473e-02, 4.0724e-02, 4.0716e-02, 9.0197e-01, 4.4439e-01,
        9.6261e-01, 7.6321e-03, 1.6499e-05, 1.1841e-02, 2.6783e-02, 4.9353e-01,
        4.6109e-01, 8.3807e-01, 5.8461e-02, 2.5327e-02, 5.8567e-01, 8.8743e-02,
        6.4555e-01, 1.4543e-01, 6.1783e-01, 1.2436e-02, 6.1343e-02, 4.2568e-01,
        3.6688e-01, 1.3877e-01, 6.3686e-01, 7.0544e-01, 1.8882e-02, 5.4320e-02,
        9.1744e-01, 1.0975e-01, 1.0416e-01, 2.6253e-04])
```

[23]:
```python
# sum all the elements of the tensor
a_sum = torch.sum(a)
a_sum
```

[23]:
```
tensor(51.5830)
```

[24]:
```python
# square each element and then take the sum of all elements
a_sq_sum = torch.sum(a_sq)
a_sq_sum
```

[24]:
```
tensor(35.1974)
```

[25]:
```python
# verify if the sum of squared computed above is correct using the below␣
↪commands
sum = 0
for ele in a:
  sum += ele**2

sum
```

[25]:
```
tensor(35.1974)
```

[26]:
```python
# take the square root of the sum of the squared elements to compute the energy
energy = torch.sqrt(a_sq_sum)
energy
```

[26]: tensor(5.9327)

[27]:
```
# run all the above commands on a random 2 dimensional tensor of dim 100x160 to␣
↪get the energy
x = torch.rand(100, 160)   # create a random 2 dimensional tensor
x_sq = torch.square(x)   # compute the square of the tensor
x_sq_sum = torch.sum(x_sq, dim=0)  # get the sum of the tensor across the␣
↪columns
x_sq_sum_sqr = torch.sqrt(x_sq_sum)   # compute the square root
x_sq_sum_sqr
```

[27]:
```
tensor([5.7111, 5.8186, 5.9421, 5.9185, 5.3794, 5.8461, 5.6229, 5.8315, 5.4879,
        5.5983, 5.9932, 6.0213, 5.7431, 5.6395, 6.0477, 5.9583, 5.9936, 5.7420,
        6.0014, 5.7516, 6.2283, 5.3087, 5.5614, 5.3031, 5.6396, 5.2963, 5.6375,
        6.1682, 5.9941, 6.2563, 5.7537, 5.6696, 5.1721, 5.6082, 5.5811, 5.9393,
        5.4079, 5.7711, 5.5181, 5.7472, 5.5417, 5.3338, 5.6708, 5.6668, 5.8337,
        6.1032, 5.8952, 5.2922, 5.3657, 5.3868, 5.8377, 5.8779, 5.9503, 5.8923,
        5.6297, 5.5330, 5.6147, 5.9041, 5.8634, 5.5893, 5.3415, 5.3992, 5.4710,
        5.5661, 5.3117, 5.9385, 5.9472, 5.9040, 6.2034, 5.4951, 5.5479, 6.0110,
        5.6174, 5.1581, 6.0778, 5.8624, 5.7439, 5.8113, 5.9218, 6.0855, 5.9342,
        5.4862, 6.1392, 5.7821, 5.6319, 5.2634, 5.5642, 5.3245, 5.7531, 5.7563,
        5.7462, 5.9608, 5.9194, 6.0537, 5.8206, 5.7429, 6.2792, 6.2621, 5.9535,
        5.5456, 5.5483, 5.8813, 5.8033, 5.8016, 5.5024, 5.7089, 5.5061, 5.7961,
        5.7272, 5.8849, 5.9361, 5.4185, 5.6078, 5.7693, 5.9793, 6.0663, 6.3255,
        5.8741, 5.8809, 5.7564, 6.0524, 5.9536, 5.8537, 6.0805, 6.0308, 6.3134,
        5.8053, 5.5621, 5.7748, 5.6830, 5.7685, 5.8171, 5.6492, 5.8973, 5.0220,
        5.8436, 6.0081, 5.7857, 6.1102, 6.1102, 6.1383, 5.9180, 5.8889, 5.9978,
        5.4743, 5.2190, 5.8211, 5.1508, 6.0242, 5.6624, 5.8134, 5.6078, 5.5932,
        5.7522, 6.0316, 5.6543, 6.0803, 5.2294, 5.9035, 5.7169])
```

4 Section 3: VAD

This section focuses on Energy-based VAD

4.0.1 Complete the code below [20 pt]

The code below runs computes frame-level log energy.

[28]:
```
def energy_passes_threshold(energy_value: float, threshold: float) -> bool:
  # Write code here. The function must return true if energy_value
  # is greater than or equal to the threshold. Otherwise return False.
  return True if threshold <= energy_value else False
```

[29]:
```
assert energy_passes_threshold(10, 5) == True, "Test failed"
assert energy_passes_threshold(10, 15) == False, "Test failed"
print("All tests passed!")
```

```
All tests passed!
```

[30]:
```python
from pathlib import Path
from typing import Union, Sequence

# Assume the window size to be 25 milliseconds. Enter the integer value
# of the size in terms of number of samples assuming we will process all
# audio in 16 kHz
# 16,000samples/s * 0.025s=400samples
WINDOW_SIZE_IN_SAMPLES: int = 400

# Similarly, ff the window shift is 10 milliseconds, what should be the
# value in terms of number of samples?
# 16,000samples/s * 0.010s = 160samples
FRAME_SHIFT_IN_SAMPLES: int = 160
```

[31]:
```python
assert isinstance(WINDOW_SIZE_IN_SAMPLES, int), "WINDOW_SIZE_IN_SAMPLES should␣
↪be int (an integer)"
assert isinstance(FRAME_SHIFT_IN_SAMPLES, int), "FRAME_SHIFT_IN_SAMPLES should␣
↪be int (an integer)"
print("All tests passed!")
```

```
All tests passed!
```

[32]:
```python
def normalize_tensor(value: torch.Tensor):
  m = value.mean()
  s = value.std()
  return (value-m)/s
```

[33]:
```python
def compute_energy_of_one_window(windowed_signal: torch.Tensor):
  # assume the input to be a one dimensional tensor
  energy = torch.sum(torch.square(windowed_signal))
  return energy
```

[34]:
```python
def energy_based_vad(audio_file: Union[Path, str], threshold: float=0.0) ->␣
↪Sequence[bool]:
  """Computes Voice Activity Detection (VAD) using energy thresholding.

  Args:
    audio_file: Path to the audio_file
    threshold: The energy threshold for VAD.

  Returns:
    A list of booleans, indicating voice activity for each audio file.
  """
  audio = AudioSegment.from_file(audio_file)  # load the audio file with␣
↪AudioSegment
  assert audio.frame_rate == 16000, "Make sure the sampling rate of the audio␣
↪is 16000"
```

```
    samples = torch.Tensor(audio.get_array_of_samples())  # load the samples from␣
↪the audio file into a Tensor (see example from class)
    assert isinstance(samples, torch.Tensor), "ERROR: samples variable should be␣
↪a Tensor"

    window = torch.hann_window(WINDOW_SIZE_IN_SAMPLES)
    import torch.nn.functional as F
    padded = F.pad(samples, [0, WINDOW_SIZE_IN_SAMPLES])

    # NOTE: Bonus points if you could vectorize (assuming you do not use AI tools)
    energy_per_frame = []
    for i in range(0, samples.shape[0], FRAME_SHIFT_IN_SAMPLES):
      windowed_signal = window * padded[i:i+WINDOW_SIZE_IN_SAMPLES]
      energy_value = compute_energy_of_one_window(windowed_signal)
      energy_per_frame.append(energy_value)

    energy_per_frame = torch.Tensor(energy_per_frame)
    # take log of energy
    logenergy_per_frame = energy_per_frame.log()

    normalized_energy_per_frame = normalize_tensor(energy_per_frame)

    # the following line of code is equivalent to checking each value in the
    vad_results_per_frame = normalized_energy_per_frame > threshold

    return vad_results_per_frame
```

[35]:
```
# call the above function for one audio file
# audio_file = '/content/drive/MyDrive/work/uzh/teaching/2024-speech-technology/
↪audio_files_ex2/librispeech/1089-134686-0000.wav'
audio_file = "/content/drive/MyDrive/librispeech/1089-134686-0000.wav"  # fix␣
↪the audio filename path in the above command here
vad_results_energy_based = energy_based_vad(audio_file)  # call the function␣
↪with the default threshold of 0.0

vad_results_energy_based
```

[35]:
```
tensor([False, False, False,  …, False, False, False])
```

[36]:
```
def number_of_speech_frames(vad_results_per_frame: torch.Tensor) -> int:
    """This function returns the number of values that are True in the input␣
↪tensor"""
    return vad_results_per_frame.sum()

vad_results_per_frame = number_of_speech_frames(vad_results_energy_based)
vad_results_per_frame
```

[36]: tensor(233)

[37]:
```
# In this cell you only need to make sure that the variable 'txt_filename'␣
↪points
# to the correct file.

def load_groundtruth_from_labfile():
  """This function loads a manually labelled file

  It is used as input to compute frame-level accuracy of the VAD algorithm.
  """
  # txt_filename = '/content/drive/MyDrive/work/uzh/teaching/
↪2024-speech-technology/audio_files_ex2/librispeech/1089-134686-0000.txt'
  txt_filename = "/content/drive/MyDrive/librispeech/1089-134686-0000.txt"  #␣
↪see example in the above line
  boundaries = []
  with open(txt_filename) as ipf:
    for ln in ipf:
      lns = ln.strip().split()
      if lns[-1] == 'speech':  # if the 3rd column is speech
        boundaries.append(list(map(float, lns[0:2])))
  return boundaries
```

[38]:
```
# Nothing to do here. Simply run the cell.
def create_frame_level_groundtruth(boundaries, file_duration,␣
↪n_frames_per_second=100):
  import math
  total_frames = int(math.ceil(file_duration*n_frames_per_second))
  groundtruth = torch.zeros(total_frames, dtype=torch.uint32)
  for start, end in boundaries:
    start_frame_id = int(math.floor(start*n_frames_per_second))
    end_frame_id = int(math.ceil(end*n_frames_per_second))
    start_frame_id = min(start_frame_id, total_frames)
    end_frame_id = max(end_frame_id, total_frames)
    groundtruth[start_frame_id:end_frame_id] = 1
  return groundtruth

def get_file_duration(audio_file):
  audio = AudioSegment.from_file(audio_file, 'wav')
  return audio.duration_seconds

def get_frame_accuracy(reference, hypothesis):
  n_correct = (reference.long() == hypothesis.long()).sum()
  den = len(reference)
  print(den)
  return n_correct*100.0/den
```

4.0.2 Q & A

Q. What does the function `get_frame_accuracy` do exactly?

A.

4.1 The function computes the accuracy of predicted frames compared to actual frames. It counts how many predictions are correct, divides this count by the total number of frames in the reference, and returns the accuracy as a percentage. Additionally, it prints the number of reference frames for debugging purposes.

[39]:
```
boundaries = load_groundtruth_from_labfile()
duration = get_file_duration(audio_file)
groundtruth = create_frame_level_groundtruth(boundaries, duration)
```

[40]:
```
print("Frame accuracy is ", get_frame_accuracy(groundtruth,␣
↪vad_results_per_frame).item())
```

```
1044
Frame accuracy is  0.0
```

4.1.1 Manual observation [10 pt]

Q. Now we have a working solution to apply energy-based VAD on an audio file. In the above solution, we used a default threshold of 0.0. Given that roughly 1 second of speech produces 100 frames per second, do you think the algorithm is accurate? Open the audio file. You may use Audacity or any other visualization tool. Verify visually if this could be possible.

A. Enter your answer here WHen using the frame counter in audacity i only get 30 frames produced in one second

4.1.2 Find the best threshold [20 pt]

Next, try out the following values of threshold:

- 0.0 (the default one)
- 1.0
- -1.0
- -0.4

For each threshold value above, run the following:

- Compute `vad_results_energy_based` for a threshold
- print the number of frames with `number_of_speech_frames` function
- print the accuracy

[43]:
```
zero = energy_based_vad(audio_file)
one = energy_based_vad(audio_file, 1.0)
neg_one = energy_based_vad(audio_file, -1.0)
neg_four = energy_based_vad(audio_file, -0.4)
```

```
zero_nsf = number_of_speech_frames(zero)
one_nsf = number_of_speech_frames(one)
neg_one_nsf = number_of_speech_frames(neg_one)
neg_four_nsf = number_of_speech_frames(neg_four)

print("zero: ", zero_nsf)
print("one: ", one_nsf)
print("neg_one: ", neg_one_nsf)
print("neg_four: ", neg_four_nsf)

acc_zero = get_frame_accuracy(groundtruth, zero)
acc_one = get_frame_accuracy(groundtruth, one)
acc_neg_one = get_frame_accuracy(groundtruth, neg_one)
acc_neg_four = get_frame_accuracy(groundtruth, neg_four)

print(f"Accuracy zero: {acc_zero.item()}")
print(f"Accuracy one: {acc_one.item()}")
print(f"Accuracy neg_one: {acc_neg_one.item()}")
print(f"Accuracy neg_four: {acc_neg_four.item()}")
```

```
zero:  tensor(233)
one:  tensor(86)
neg_one:  tensor(1044)
neg_four:  tensor(644)
1044
1044
1044
1044
Accuracy zero: 27.681991577148438
Accuracy one: 13.793103218078613
Accuracy neg_one: 94.44444274902344
Accuracy neg_four: 66.85823822021484
```

Q. According to the results, which threshold works best?

A. The best threshold is -1 bc it gives us the highest accuracy

5 Feedback [5 pt]

Q. How did you find the difficulty level of this assignment

A.

(a) Easy
(b) Intermediate
(c) Difficult
(d) Extremely difficult

c

Q. How much time did you spend in finish this assignment

(a) 0 – 1 hour
(b) 1 – 2 hours
(c) 2 – 3 hours
(d) More than 3 hours

c