ESE_2240_Spring_2025_Lab_6_Group3

March 18, 2025

Laura Gao, Mohamed Elsheshtawy, Celia Tung Group 3
https://colab.research.google.com/drive/1mx_YRmUOiDSBaWGUOoBfki-uzZfwG07V?usp=sharing

[ ]:
```
!pip -q install pydub
```

[ ]:
```
import numpy as np
from IPython.display import Javascript, Audio
from google.colab import output
from base64 import b64decode
from io import BytesIO
from pydub import AudioSegment
```

1 0. Provided Functions

[ ]:
```
def compute_fft(x, fs: int, center_frequencies=True):
    """
    Compute the DFT of x.

    Args:
        x: Signal of length N.
        fs: Sampling frequency.
        center_frequencies: If true then returns frequencies on [-f/2,f/2]. If␣
 ↪false then returns frequencies between [0,f].
    Returns:
        X: DFT of x.
        f: Frequencies
    """
    N = len(x)
    X = np.fft.fft(x, norm="ortho")
    f = np.fft.fftfreq(N, 1/fs)
    if center_frequencies:
        X = np.fft.fftshift(X)
        f = np.fft.fftshift(f)
    return X, f

def inner_product(x, y):
```

```
    """
    Computes the inner product between two numpy arrays x and y.

    Args:
        x: numpy array.
        y: numpy array.
    """
    return np.abs(x.T) @ np.abs(y)
```

[ ]:
```
RECORD = """
const sleep  = time => new Promise(resolve => setTimeout(resolve, time))
const b2text = blob => new Promise(resolve => {
    const reader = new FileReader()
    reader.onloadend = e => resolve(e.srcElement.result)
    reader.readAsDataURL(blob)
})
var text = document.createTextNode("TEST")
document.querySelector("#output-area").appendChild(text)
var record = time => new Promise(async resolve => {
    chunks = []
    stream = await navigator.mediaDevices.getUserMedia({ audio: true })
    recorder = new MediaRecorder(stream)
    recorder.ondataavailable = e => chunks.push(e.data)
    recorder.onstop = async ()=>{
        blob = new Blob(chunks)
        text = await b2text(blob)
        resolve(text)
    }
    recorder.start()
    text.data = "Start talking"
    await sleep(time)
    text.data = "Stop talking"
    recorder.stop()
})
"""

def record(fs=20_000, T=2):
    """
    Record audio from microphone in colab.
    Args:
        T: Duration in seconds.
        fs: Sampling rate.
    Returns:
        A numpy array of length approximately T*fs.
    """
    display(Javascript(RECORD))
    s = output.eval_js(f"record({T*1000})")
```

```
    b = b64decode(s.split(',')[1])
    samples = (
        sum(AudioSegment.from_file(BytesIO(b)).split_to_mono())
        .set_frame_rate(fs)
        .get_array_of_samples()
    )
    fp_arr = np.array(samples).T.astype(np.float32)
    fp_arr /= np.iinfo(samples.typecode).max
    return fp_arr
```

2 Q1. Acquire and Process Training Sets

We leave this to you

[ ]:
```
# Useful plotting function
def plot_signal_and_dft(x, t, X, f, title):
  plt.figure(figsize=(10,4))
  plt.subplot(1, 2, 1)
  plt.plot(t, x)
  plt.title('Signal')
  plt.xlabel('Time [s]')

  plt.subplot(1, 2, 2)
  plt.plot(f, np.abs(X))
  plt.title('Magnitude of DFT')
  plt.xlabel('Frequency [Hz]')
  plt.xlim([0, fs/2])

  plt.suptitle(title)
  plt.tight_layout()
  plt.show()
```

[ ]:
```
def normalize_unit_energy(signal):
  norm = np.sum(np.abs(signal**2))
  return signal / np.sqrt(norm)

T = 2.0
fs = 20_000
num_recs = 10

# TODO: Acquire and process training set for "one"
ones = []
for i in range(num_recs):
  next = input()
  x = record(fs)
  X, f = compute_fft(x, fs)
```

```
  X_normalized = normalize_unit_energy(X)
  ones.append(X_normalized)


for i in range(num_recs):
  np.save(f'one_{i}.npy', ones[i])
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[ ]:
```
# TODO: Acquire and process training set for "two"
twos = []
for i in range(num_recs):
  next = input()
  x = record(fs)
  X, f = compute_fft(x, fs)
  X_normalized = normalize_unit_energy(X)
  twos.append(X_normalized)
```

```
for i in range(num_recs):
  np.save(f'two_{i}.npy', twos[i])
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[ ]:
```
# TODO: Create test sets for "one" and "two" - Remember to use numpy.save() to␣
  ↪save your recordings
test = []
for i in range(num_recs):
  next = input()
  x = record(fs)
  X, f = compute_fft(x, fs)
  X_normalized = normalize_unit_energy(X)
  test.append(X_normalized)


for i in range(num_recs):
  np.save(f'test_{i}.npy', test[i])
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>
```

3 Q2. Comparison with Average Spectrum

We leave this to you

[ ]:
```
ones = [np.load(f'one_{i}.npy') for i in range(num_recs)]
twos = [np.load(f'two_{i}.npy') for i in range(num_recs)]
test = [np.load(f'test_{i}.npy') for i in range(num_recs)]

ones_average = np.mean(ones, axis=0)
twos_average = np.mean(twos, axis=0)

for element in test:
  inner_product_with_one = inner_product(element, ones_average)
  inner_product_with_two = inner_product(element, twos_average)
  if inner_product_with_one > inner_product_with_two:
    print("one")
  else:
```

```
    print("two")
```

```
one
two
two
two
one
two
one
two
one
two
```

4 Comments:

This accuracy is 9/10. Our test set has alternated recordings of one and two and everything was classified correctly except for the third one.

We use the absolute value to obtain the magnitudes for the inner products because there could be parts of the spectra that is complex, so we need to put it through abs.

5 Q3. Nearest Neighbor Comparison

We leave this to you

[ ]:
```
for element in test:
  max_inner_products_with_ones = max([inner_product(element, one) for one in␣
 ↪ones])
  max_inner_products_with_twos = max([inner_product(element, two) for two in␣
 ↪twos])

  if max_inner_products_with_ones > max_inner_products_with_twos:
    print("one")
  else:
    print("two")
```

```
one
two
two
two
one
two
one
two
one
two
```

6 Comments:

The classification accuracy is still 90% with the same results being assigned to each test case. Maybe test recording #3 just had a lot of noise in the background or was otherwise wrong.