Assignment Documentation - Joshua Fernandes

1) Count the number of pauses

To count the number of pauses, I split the audio file into chunks of 0.2 seconds and used a 0.2 second low noise sample as reference. For each chunk, I computed the cosine similarity between itself and the reference file to classify it as a silent/non-silent chunk. I only classified a silent chunk to be a pause if the chunk before it wasn't since multiple silent chunks in a row would mean that they're part of the same pause.

Algorithm:

1. Initialize the counter and set the chunk size to 0.2 seconds.
2. Load audio to memory
3. Load utterance sample.
4. Split audio into chunks of 0.2 seconds
5. Initialize a variable 'prev' with value 1 to keep track of the previous chunk's state
6. Iterate through chunks:
    a. Convert the waveform to a spectrogram
    b. Flatten the spectrogram
    c. Compute the cosine similarity between the chunk and the sample:
        i. If the similarity > 0.95 AND prev == 1 :
            1. increment counter by 1
            2. Set prev = 0
        ii. else:
            1. Set prev = 1
7. Return count

Code:

```
def no_of_pauses(audio):
    f = open('silence.pickle', 'rb')
    silence = pickle.load(f)
    f.close()

    wav, sr = librosa.load(audio)
    if sr != 22050:
        wav = librosa.resample(wav, sr, 22050)

    chunk_size = 4410
    prev, pauses = 1, 0
    for i in range(0, len(wav), chunk_size):
        chunk = wav[i:i+chunk_size]
        if len(chunk) == 4410:
            X = librosa.stft(chunk)
            Xdb = librosa.amplitude_to_db(abs(X))
            Xdb = Xdb.reshape(Xdb.shape[0]*Xdb.shape[1], -1)
            if cosine_similarity(silence.reshape(1,-1), Xdb.reshape(1,-1)) > 0.95:
                if prev == 1:
                    pauses += 1
                prev = 0
            else:
                prev = 1
        else:
            pass
    return pauses
```

2) Count the repetition of words

Algorithm:

1. Convert speech to text and tokenize.
2. Get a list of all the unique words.
3. Create a dictionary where each unique word is a key.
4. Iterate through the list of all unique words:
    a. Use the ‘count()’ function to check the number of occurrences of a given word within the tokenized text
    b. By using the word as a key, update it’s value within the dictionary.
5. Create a new list
6. Iterate through the dictionary and append a word to the new list if it’s occurrence is more than 1.
7. Return the length of the new list

Code:

```
def repetition_of_words(text):
    words_unique = set(text)
    occurrences = [0 for x in range(len(words_unique))]
    words_dict = dict(zip(words_unique, occurrences))
    for x in words_unique:
        words_dict[x] = text.count(x)
    repeated_words = [x for x in words_unique if words_dict[x] > 1]
    return len(repeated_words)
```

3) Count the number of different words used

Algorithm:

1. Convert speech to text and tokenize.
2. Convert the tokenized text into a set
3. Return the length of the set.

Code:

```
def unique_words(text):
    return len(set(text))
```

4) Number of words spoken per minute

Algorithm:

1. Convert speech to text and tokenize.
2. Get the duration of the audio file in minutes.
3. Divide the number of words by the duration in minutes
4. Round up the result to two decimal places.

Code:

```
def words_per_minute(audio, text):
    f = audioread.audio_open(audio)
    mins = (f.duration)/60
    wpm = len(text)/mins # no of words/duration in mins
    return round(wpm, 2) # round up to 2 decimal places
```

5) Count number of "aaaa"

'aaa' is an example of a speech interjection. For this task, I used an audio sample of such an utterance. I split the audio file into chunks of 0.5 seconds and computed the cosine similarity of each chunk with the sample utterance.

Algorithm:

1. Initialize the counter and set the chunk size to 0.5 seconds.
2. Load audio to memory
3. Load utterance sample.
4. Split audio into chunks of 0.5 seconds
5. Iterate through chunks:
    a. Convert the waveform to a spectrogram
    b. Flatten spectrogram from (1025, 22) to (22550,)
    c. Compute the cosine similarity between the chunk and the sample:
        i. If the similarity > 0.9 : increment counter by 1
        ii. else: pass
6. Return count

Code:

```
def count_interjections(audio): # aaa
    f = open('sample.pickle', 'rb')
    sample = pickle.load(f)
    f.close()
    wav, sr = librosa.load(audio)
    if sr != 22050:
        wav = librosa.resample(wav, sr, 22050)
    # split audio into chunks of 0.5 seconds
    count, chunk_size = 0, 11025
    for i in range(0, len(wav), chunk_size):
        chunk = wav[i:i+chunk_size]
        if len(chunk) == 11025:
            X = librosa.stft(chunk)
            Xdb = librosa.amplitude_to_db(abs(X))
            Xdb = Xdb.reshape(Xdb.shape[0]*Xdb.shape[1], -1)
            if cosine_similarity(sample.reshape(1,-1), Xdb.reshape(1,-1)) > 0.9:
                count += 1
        else:
            pass

    return count
```

Output:

```
Enter Filename: voice.flac

Converting speech to text...

No. of Pauses: 68
Unique words: 129
Words Per Minute: 76.56
Repeated Words: 49
Interjections (aaa): 8
```