Lab 6 - Solution

November 13, 2019

1 Lab 6 - Solution

In this lab we will, as a class, create the grading script for the final project.

```
In [1]: import librosa
        import numpy as np
        import sys
        import pickle
        import time
        import IPython.display as ipd
        from codec import encode, decode
```

1.1 Import Audio File

Load in an audio file to test with codec.

```
In [2]: def load_cd_quality_audio(filename):
            audio, sr = librosa.load(filename, sr = 44100, dtype='float_')
            max_int_value = 2**15 - 1
            audio *= max_int_value
            audio = audio.astype('int16')
            return audio

        x = load_cd_quality_audio("taxman.wav")
```

1.2 Runtime

Encode and decode the audio. Time the processes.

```
In [3]: encodeStartTime = time.time()
        x_encoded = encode(x)
        ert = time.time() - encodeStartTime

        decodeStartTime = time.time()
        x_decoded = decode(x_encoded)
        drt = time.time() - decodeStartTime
```

1.3 Format Check

Check to make sure the decoded audio is mono 16 bit.

```
In [4]: def check_decoded_output(d):
            if type(d) != np.ndarray:
                print('ERROR: Your decoded signal is not a numpy array!')
            elif d.dtype != 'int16':
                print('ERROR: Your decoded signal does not contain 16 bit integers!')
            elif len(d.shape) != 1:
                print('ERROR: Your signal is not a 1-dimensional vector!')
            else:
                print('Your decoded signal passes the format check.')

        check_decoded_output(x_decoded)
```

```
Your decoded signal passes the format check.
```

1.4 Compression Ratio

Compare the sizes of the original and encoded structures.

```
In [5]: def compressionRatio(original, encoded):
            o_str = pickle.dumps(original)
            e_str = pickle.dumps(encoded)
            return sys.getsizeof(o_str)/sys.getsizeof(e_str)

        cr = compressionRatio(x, x_encoded)
```

1.5 SNR

Compare the original signal content to the decoded version

```
In [6]: def signalToNoise(original, decoded):

            original = original.astype('float_')
            decoded = decoded.astype('float_')

            # force the signals to be the same length
            diff = len(original) - len(decoded)
            if diff < 0:
                decoded = decoded[:diff]
```

```
            elif diff > 0:
                decoded = np.append(decoded, np.zeros( (diff,1) ) )

            # compute snr
            signal = np.power(original,2)
            noise = np.power(original - decoded,2)

            signal = np.where(signal == 0, np.finfo(np.float32).eps, signal)
            noise = np.where(noise == 0, np.finfo(np.float32).eps, noise)

            return np.mean(10 * np.log10(signal/noise))

        snr = signalToNoise(x, x_decoded)
```

1.6 Evaluate Codec

Print out evalutation of codec. Listen to the results

```
In [7]: print("Compression Ratio: ", str(round(cr,4)))
        print()
        print("Total Runtime: ", str(round(ert + drt,4)))
        print("\tEncode Runtime: ", str(round(ert,4)))
        print("\tDecode Runtime: ", str(round(drt,4)))
        print()
        print("SNR: ", str(round(snr,4)))

Compression Ratio:  1.0

Total Runtime:  0.0001
        Encode Runtime:  0.0001
        Decode Runtime:  0.0

SNR:  135.3601


In [8]: # Orignal
        ipd.Audio(x, rate = 44100)

Out[8]: <IPython.lib.display.Audio object>

In [9]: # Decoded
        ipd.Audio(x_decoded, rate = 44100)

Out[9]: <IPython.lib.display.Audio object>
```