CS724: SENSING COMMUNICATIONS AND NETWORKING FOR SMART WIRELESS DEVICES

Assignment 2

Krishna Kumar Bais

Roll NO : 241110038

Report on Audio Analysis and Spectrogram Generation

Introduction

This report details the process of analyzing an audio file using Python and the librosa library. The analysis includes loading an audio file, visualizing its waveform, and generating various spectrogram representations, including the magnitude spectrogram, Mel spectrogram, and Constant-Q Transform (CQT). These visualizations provide insights into the frequency content and characteristics of the audio signal over time.

Audio File Loading

The audio file used for this analysis is named kk.mp3. The first step involved loading this audio file using the librosa library:

```
audio_signal, sample_rate = librosa.load(audio_file_path)
```

Upon loading, the audio signal is stored in the variable audio_signal, and the sample rate is stored in sample_rate. This allows for further analysis of the audio content.

Waveform Visualization

The waveform of the audio signal was plotted to visualize the amplitude variations over time. This is done using the following code:

```
plt.figure(figsize=(12, 4))
librosa.display.waveshow(audio_signal, sr=sample_rate)
plt.title("Audio Waveform")
plt.xlabel("Time (seconds)")
plt.ylabel("Amplitude")
plt.show()
```

Observation: The waveform provides a visual representation of the audio signal, showing how the amplitude changes over time. Peaks in the waveform indicate louder sections of the audio, while valleys represent quieter sections.

Short-Time Fourier Transform (STFT)

To analyze the frequency content of the audio signal, the Short-Time Fourier Transform (STFT) was computed:

```
 stft_output = librosa.stft(audio_signal, n_fft=fft_window_size,
hop_length=hop_length)
```

The STFT allows us to observe how the frequency spectrum of the audio signal evolves over time.

Magnitude Spectrogram Visualization

The magnitude spectrogram was calculated from the STFT output and visualized using a heatmap:

```
 magnitude_spectrogram = np.abs(stft_output) ** 2
 plt.figure(figsize=(12, 4))
 librosa.display.specshow(librosa.power_to_db(magnitude_spectrogram),
sr=sample_rate, hop_length=hop_length, x_axis="time", y_axis="linear",
cmap='hot')
 plt.colorbar(format="%+2.0f dB")
 plt.title("Magnitude Spectrogram Heatmap")
 plt.xlabel("Time (seconds)")
 plt.ylabel("Frequency (Hz)")
 plt.show()
```

Observation: The magnitude spectrogram illustrates the power of different frequency components over time. The color intensity represents the amplitude of the frequencies, with brighter colors indicating higher energy levels. This visualization helps identify prominent frequencies and their changes throughout the audio.

Mel Spectrogram Calculation

The Mel spectrogram was computed to represent the audio signal in a way that is more aligned with human auditory perception:

```
 mel_spectrogram = librosa.feature.melspectrogram(y=audio_signal,
sr=sample_rate, n_fft=fft_window_size, hop_length=hop_length)
 mel_spectrogram_db = librosa.power_to_db(mel_spectrogram)
```

The Mel spectrogram was then visualized:

```
 plt.figure(figsize=(12, 4))
 librosa.display.specshow(mel_spectrogram_db, sr=sample_rate,
hop_length=hop_length, x_axis="time", y_axis="mel", cmap='viridis')
 plt.colorbar(format="%+2.0f dB")
 plt.title("Mel Spectrogram")
 plt.xlabel("Time (seconds)")
 plt.ylabel("Mel Frequency (Hz)")
 plt.show()
```

Observation: The Mel spectrogram provides a perceptually relevant representation of the audio signal, emphasizing frequencies that are more important for human hearing. This visualization is particularly useful in applications such as speech recognition and music analysis.

Constant-Q Transform (CQT)

The Constant-Q Transform (CQT) was computed to analyze the audio signal with a logarithmic frequency scale, which is more suited for musical applications:

```
 cqt_output = librosa.cqt(audio_signal, sr=sample_rate,
hop_length=hop_length)
 cqt_db = librosa.amplitude_to_db(np.abs(cqt_output))
```

The CQT was then visualized:

```
 plt.figure(figsize=(12, 4))
 librosa.display.specshow(cqt_db, sr=sample_rate, hop_length=hop_length,
x_axis="time", y_axis="cqt_hz", cmap='inferno')
 plt.colorbar(format="%+2.0f dB")
 plt.title("Constant-Q Transform (CQT)")
 plt.xlabel("Time (seconds)")
 plt.ylabel("Frequency (Hz)")
 plt.show()
```

Observation: The CQT visualization highlights the harmonic content of the audio signal, making it particularly effective for analyzing musical signals. The logarithmic frequency scale allows for better representation of musical notes and their relationships.

Conclusion

The analysis of the audio file kk.mp3 through various visualizations provides valuable insights into the characteristics of the audio signal. The waveform, magnitude spectrogram, Mel spectrogram, and Constant-Q Transform all serve different purposes in understanding the frequency content and temporal evolution of the audio. These tools are essential for applications in audio processing, music analysis, and speech recognition, enabling researchers and practitioners to extract meaningful information from audio signals.

Report on Accelerometer Data Visualization for Different Body Positions

Introduction

This report outlines the process of visualizing accelerometer data collected from a device placed in three different body positions: sitting, standing, and lying down. The accelerometer measures acceleration forces in three dimensions (X, Y, and Z), which can be analyzed to understand the movement and stability of the body in various postures. The goal of this analysis is to plot the acceleration data over time for each position, allowing for visual comparison of the readings.

Data Collection

The accelerometer data was collected by placing a device (such as a smartphone) in three distinct positions:

1. **Sitting**: The device was placed on the body while sitting.
2. **Standing**: The device was placed on the body while standing.
3. **Lying Down**: The device was placed on the body while lying down.

Each position's data was saved in separate CSV files (sitting_file.csv, standing_file.csv, and sleeping_file.csv). Each file contains columns for time and acceleration values along the X, Y, and Z axes:

- time: Time in seconds
- gFx: Acceleration along the X-axis (m/s^2)
- gFy: Acceleration along the Y-axis (m/s^2)
- gFz: Acceleration along the Z-axis (m/s^2)

Data Visualization

The visualization of the accelerometer data was accomplished using Python's pandas and matplotlib libraries. The following function was defined to plot the acceleration data for each body position:

```
 def plot_acceleration_data(dataframe, position_name, color_map, title,
filter_range=None):
     time_values = dataframe['time'].values
     x_acceleration = dataframe['gFx'].values
     y_acceleration = dataframe['gFy'].values
     z_acceleration = dataframe['gFz'].values

     if filter_range:
         start_idx, end_idx = filter_range
         time_values = time_values[start_idx:end_idx]
         x_acceleration = x_acceleration[start_idx:end_idx]
         y_acceleration = y_acceleration[start_idx:end_idx]
         z_acceleration = z_acceleration[start_idx:end_idx]
```

```
     plt.figure(figsize=(12, 6))
     plt.plot(time_values, x_acceleration, color=color_map['x'], label='X-
axis')
     plt.plot(time_values, y_acceleration, color=color_map['y'], label='Y-
axis')
     plt.plot(time_values, z_acceleration, color=color_map['z'], label='Z-
axis')
     plt.xlabel('Time (seconds)')
     plt.ylabel('Acceleration (m/s²)')
     plt.title(f'{position_name}: Time vs Acceleration' + (' (Filtered)' if
filter_range else ''))
     plt.legend()
     plt.grid(True)
     plt.show()
```

Visualization for Each Position

1. **Sitting Position**:

   - The data for the sitting position was loaded from sitting_file.csv and plotted using the defined function. The X, Y, and Z accelerations were represented in blue, green, and orange, respectively.

```
sitting_data = pd.read_csv('sitting_file.csv')
plot_acceleration_data(
    sitting_data,
    position_name='Sitting',
    color_map={'x': 'blue', 'y': 'green', 'z': 'orange'},
    title='Sitting: Time vs Acceleration'
)
```

2. **Standing Position**:

    - The data for the standing position was loaded from standing_file.csv. The plot was filtered to display a specific range of samples (from index 1500 to 6000) to focus on a particular segment of the data. The X, Y, and Z accelerations were represented in purple, green, and magenta, respectively.

```
standing_data = pd.read_csv('standing_file.csv')
plot_acceleration_data(
    standing_data,
    position_name='Standing',
    color_map={'x': 'purple', 'y': 'green', 'z': 'magenta'},
    filter_range=(1500, 6000),
    title='Standing: Time vs Acceleration'
)
```

3. **Lying Down Position**:

    - The data for the lying down position was loaded from sleeping_file.csv and plotted. The X, Y, and Z accelerations were represented in red, yellow, and green, respectively.

```
lying_data = pd.read_csv('sleeping_file.csv')
plot_acceleration_data(
    lying_data,
    position_name='Lying Down',
    color_map={'x': 'red', 'y': 'yellow', 'z': 'green'},
    title='Lying Down: Time vs Acceleration'
)
```

Observations and Insights

1. **Sitting Position**: The plot for the sitting position likely shows relatively stable acceleration values, with minor fluctuations along the X and Z axes, indicating minimal movement.

2. **Standing Position**: The standing position plot may exhibit more variability in the acceleration readings, particularly in the X and Z axes, reflecting the natural movements and adjustments made while standing.

3. **Lying Down Position**: The lying down position plot should resemble the sitting position but may show different patterns based on the orientation of the device.

Conclusion

The visualization of accelerometer data for different body positions provides valuable insights into the movement and stability associated with each posture. By plotting the acceleration data over time, it becomes easier to compare the characteristics of sitting, standing, and lying down positions. This analysis can be further extended to develop algorithms for posture classification and monitoring applications in health and fitness.

Report on Accelerometer Data Classification and Visualization

Introduction

This report details the process of classifying body postures based on accelerometer data collected from a smartphone or similar device. The accelerometer measures acceleration forces in three dimensions (X, Y, and Z), which can be analyzed to determine whether a person is sitting, standing, or lying down. The analysis includes loading the accelerometer data from JSON files, calculating the minimum and maximum values for each posture, and visualizing the results.

Data Collection

Accelerometer data was collected by placing the device in three different positions: lying down, standing, and sitting. The data for each posture was recorded and saved in separate text files in JSON format. Each line in the files contains the accelerometer readings for the X, Y, and Z axes, structured as follows:

Json

```json
{"x": value_x, "y": value_y, "z": value_z}
```

Data Loading and Preprocessing

The first step in the analysis was to load the accelerometer data from the JSON files using a Python script. The following function was implemented to read the data:

```python
def load_accelerometer_data(file_name):
    x_values, y_values, z_values = [], [], []

    with open(file_name, 'r') as file:
        for line in file:
            data = json.loads(line.strip())
            x_values.append(float(data['x']))
            y_values.append(float(data['y']))
            z_values.append(float(data['z']))

    return x_values, y_values, z_values
```

Using this function, the accelerometer data was loaded for each posture:

```python
sleep_x, sleep_y, sleep_z = load_accelerometer_data("KrishnaSleeping.txt")
stand_x, stand_y, stand_z = load_accelerometer_data("KrishnaStanding.txt")
sit_x, sit_y, sit_z = load_accelerometer_data("KrishnaSitting.txt")
```

Min-Max Calculation

To classify the postures effectively, the minimum and maximum values for each axis of the accelerometer data were calculated for each posture. This was done using the following function:

```python
def find_min_max(data):
    return min(data), max(data)
```

The min-max values were calculated for each posture as follows:

```python
 sleep_min_max = [find_min_max(sleep_x), find_min_max(sleep_y),
find_min_max(sleep_z)]
 stand_min_max = [find_min_max(stand_x), find_min_max(stand_y),
find_min_max(stand_z)]
 sit_min_max = [find_min_max(sit_x), find_min_max(sit_y),
find_min_max(sit_z)]
```

Posture Classification

The classification of real-time accelerometer data was performed using a function that checks if the current readings fall within the defined min-max ranges for each posture. The classification results were stored in a list and printed to the console:

```python
 def determine_posture(ax, ay, az, posture_limits):
     return all(min_val <= val <= max_val for (min_val, max_val), val in
zip(posture_limits, [ax, ay, az]))
```

The classification process was implemented as follows:

```python
for i in range(len(test_x)):
    ax, ay, az = test_x[i], test_y[i], test_z[i]

    if determine_posture(ax, ay, az, stand_min_max):
        classification_results.append(("Standing", ax, ay, az))
    elif determine_posture(ax, ay, az, sit_min_max):
        classification_results.append(("Sitting", ax, ay, az))
    elif determine_posture(ax, ay, az, sleep_min_max):
        classification_results.append(("Lying Down", ax, ay, az))
    else:
        classification_results.append(("Pattern Not Recognized", ax, ay,
az))
```

Data Visualization

The accelerometer data for each posture was visualized using matplotlib. The following function was used to plot the data:

```python
def plot_accelerometer_data(x_data, y_data, z_data, posture_name):
    plt.figure(figsize=(10, 6))
    plt.plot(x_data, label='X-axis', color='r', marker='o')
    plt.plot(y_data, label='Y-axis', color='g', marker='o')
    plt.plot(z_data, label='Z-axis', color='b', marker='o')
    plt.title(f"Accelerometer Data for {posture_name}")
    plt.xlabel("Sample Index")
    plt.ylabel("Acceleration")
    plt.legend()
    plt.grid(True)
    plt.show()
```

The data was plotted for each posture, allowing for visual comparison:

```
plot_accelerometer_data(sleep_x, sleep_y, sleep_z, "Lying Down")
plot_accelerometer_data(stand_x, stand_y, stand_z, "Standing")
plot_accelerometer_data(sit_x, sit_y, sit_z, "Sitting")
```

Real-Time Test Data Visualization

Finally, the real-time test data was plotted along with the classification results. Each axis was represented using different colors, and the classification labels were annotated on the plot:

```
 plt.figure(figsize=(10, 6))
 plt.plot(test_x_vals, label='X-axis', color='r', marker='x')
 plt.plot(test_y_vals, label='Y-axis', color='g', marker='x')
 plt.plot(test_z_vals, label='Z-axis', color='b', marker='x')

 for i, label in enumerate(test_labels):
     plt.text(i, test_x_vals[i], label, fontsize=9, color='black',
rotation=30)

 plt.title("Real-Time Test Data Classification")
 plt.xlabel("Sample Index")
 plt.ylabel("Acceleration")
 plt.legend()
 plt.grid(True)
 plt.show()
```

Observations and Insights

1. **Data Patterns**: The plots for each posture show distinct patterns in the accelerometer readings. For instance, the standing position typically exhibits more variability in the X and Z axes due to slight movements, while the sitting and lying down positions show more stability.

2. **Classification Accuracy**: The classification results printed in the console provide insights into how well the algorithm distinguishes between different postures based on the accelerometer data.

3. **Real-Time Testing**: The final plot illustrates the real-time classification of the test data, allowing for a quick visual assessment of the posture classification performance.

Conclusion

The analysis and visualization of accelerometer data demonstrate the effectiveness of using sensor readings to classify body postures. By calculating the min-max values for each posture and visualizing the data, it is possible to distinguish between sitting, standing, and lying down positions. This methodology can be further refined and expanded for applications in health monitoring, fitness tracking, and human-computer interaction.