Audio Features Workshop 2024

Jiameng Ma(s4255445)

October 2024

Assignment1:

Track	Estimated Tempo (BPM)	Manual Listening BPM
Kevin MacLeod Vibe Ace	117	128
Track01	61.52	53
Track02	129.20	108
Track03	95.70	99
Track04	143.55	134

Table 1: Beats Per Minute (BPM) Estimation

Assignment2:

a):

1. **Kevin MacLeod Vibe Ace:**

   In **Full Spectrogram**, High-intensity regions (in brighter colors) are visible at lower frequencies, which may represent both harmonic and percussive elements mixed together.This view provides a comprehensive look at all audio components but doesn't isolate the rhythmic or melodic aspects as effectively as the separated views.

   The **harmonic spectrogram** displays more stable and continuous frequencies, particularly in the lower ranges, which represent the tonal base of the track. In the context of Assignment 1, which calculated tempo, these harmonic components are less relevant since tempo is primarily driven by the percussive, rhythmic elements.

   This **percussive spectrogram** reveals the periodic pulses and sharp attacks associated with the track's rhythmic structure, which aligns with the tempo estimation from Assignment 1 (129.20 BPM). The separation allows a clearer view of the beats, which directly correlate with the BPM

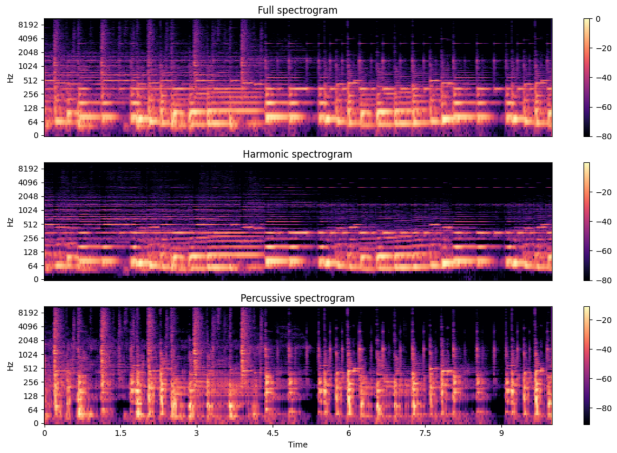

Figure 1: Spectrogram for Kevin MacLeod Vibe Ace

estimation, as the beats occur in regular intervals. This helps confirm that the track's estimated BPM (tempo) is supported by distinct rhythmic events observable in the percussive spectrogram.

2. **track01:**

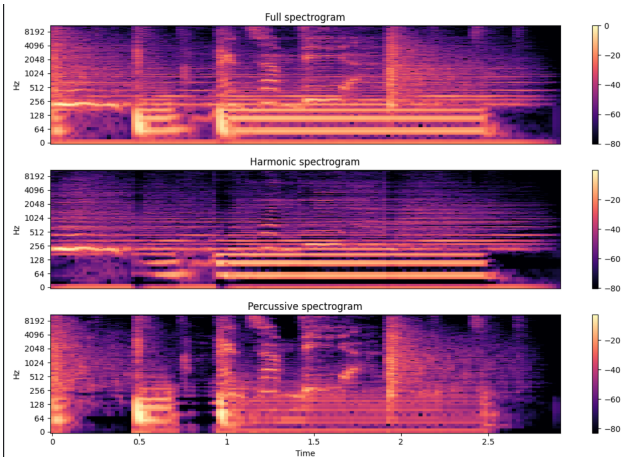

Figure 2: Spectrogram for track01

In the **Full Spectrogram** of track01, high-intensity regions (brighter colors) appear mostly at lower frequencies. These regions likely represent a blend of harmonic and percussive elements, providing an overall view of the track's audio components without isolating specific rhythmic or melodic features.

The **Harmonic Spectrogram** shows stable and continuous frequencies, especially in the lower ranges, which likely form the tonal foundation of the track. Compared to the percussive spectrogram, these harmonic components contribute less to tempo, as tempo is more closely associated with rhythmic elements.

The **Percussive Spectrogram** reveals periodic pulses and sharp attacks corresponding to the track's rhythm. These patterns align with the slower tempo estimation from Assignment 1 (61.52 BPM), as the bursts occur at intervals matching this tempo. The separation clarifies the track's beat structure, supporting the tempo calculated in Assignment 1 by highlighting consistent rhythmic events.

3. **track02:**

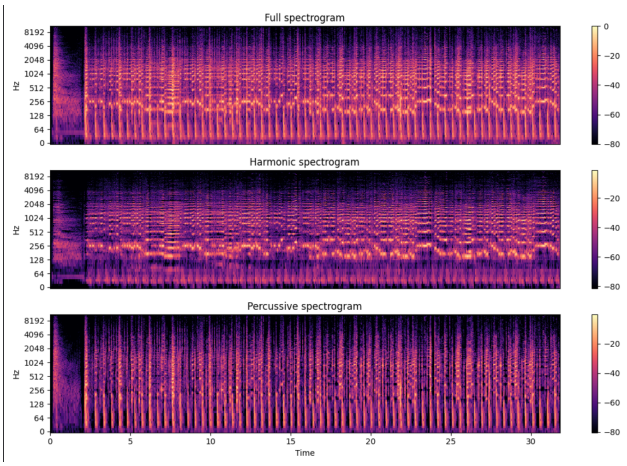

Figure 3: Spectrogram for track02

In **Full Spectrogram**, high-intensity regions (in brighter colors) are visible across various frequency bands, indicating a blend of harmonic and percussive elements. This view offers a comprehensive overview of all audio components, though it does not distinctly separate rhythmic and melodic aspects, making it less effective for isolating the beat.

The **harmonic spectrogram** shows stable, continuous frequencies, particularly in the lower ranges, representing the track's tonal foundation. While these harmonic elements contribute to the track's overall tonal quality, they are less relevant to tempo estimation, as tempo is primarily influenced by percussive, rhythmic components.

The **percussive spectrogram** highlights sharp, periodic pulses, emphasizing the track's rhythmic structure. These regular intervals directly correlate with the tempo estimated in Assignment 1 (around 129.20 BPM). The separation of beats in this spectrogram visually confirms the BPM

calculation, showing distinct rhythmic events that align with the track's estimated tempo.

4. **track03:**

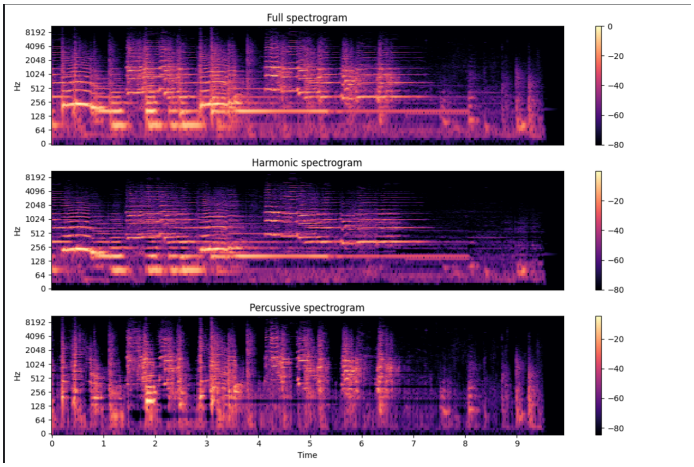

Figure 4: Spectrogram for track03

In the **Full Spectrogram** for track03, we see bright regions concentrated in the lower frequencies, likely representing both harmonic and percussive elements together. This mixed view gives a broad perspective on the entire audio but lacks the clarity needed to separate rhythmic pulses from melodic content.

The **harmonic spectrogram** shows smoother, sustained patterns across time, primarily in the lower frequencies. These indicate the tonal structure of the track, providing a base that remains steady. Harmonic elements are less impactful on tempo estimation as they represent the underlying musical tones rather than the beats driving the rhythm.

In the **percussive spectrogram**, sharp, periodic bursts are more evident, especially between 4 and 8 seconds. These bursts correlate with the rhythmic pattern of the track and would align with tempo or beat estimations, as observed in Assignment 1. The clearer separation of beats here allows for a better understanding of the tempo structure, with the distinct attacks supporting a consistent rhythmic pace.

5. **track04:**

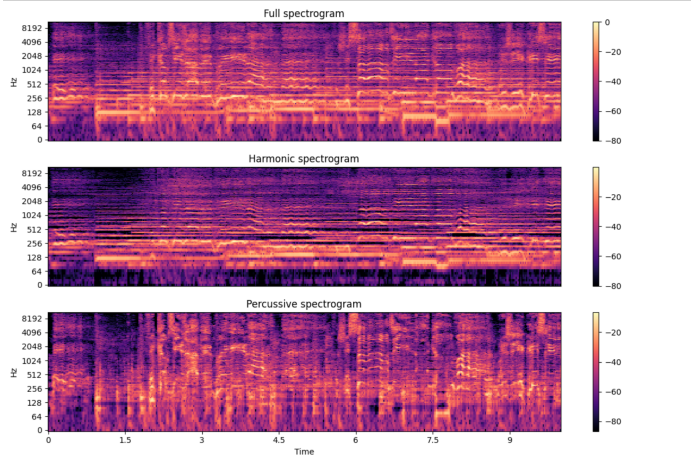

Figure 5: Spectrogram for track04

In the **Full Spectrogram**, we observe high-energy regions across both low and mid frequencies, which blend harmonic and percussive components. This view provides an overall picture of the track's audio structure but doesn't clearly isolate the rhythmic beats from the tonal background.

The **harmonic spectrogram** reveals continuous frequency bands, particularly in the lower frequency range, representing the track's melodic base. These stable patterns are indicative of tonal elements that form the harmonic foundation but do not directly influence the tempo, which is more rhythm-dependent.

In the **percussive spectrogram**, distinct bursts appear intermittently, corresponding to rhythmic events. These sharper, periodic structures align with the track's beats, which would be relevant for tempo analysis, as seen in Assignment 1. The isolated view of these percussive elements provides a clearer perspective on the rhythmic structure, supporting BPM estimation through the presence of regular beat patterns.

b):

Key Improvements:

1. High-Pass Filtering to Remove Low-Frequency Noise

```
# Function to apply a high-pass filter
def highpass_filter(data, cutoff=300, fs=22050, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
```

```
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return lfilter(b, a, data)

# Apply high-pass filter
y_filtered = highpass_filter(y, cutoff=300, fs=sr)
```

Here, a high-pass filter with a cutoff of 300 Hz removes low-frequency noise, which often interferes with both harmonic and percussive elements. By eliminating frequencies below 300 Hz, we reduce low-end noise and retain only the essential harmonic and percussive elements, making the HPSS process more effective in separating these components.

2. Normalization of Audio Signal

```
# Normalize the audio signal
y = y / np.max(np.abs(y))
```

Normalizing the audio ensures a consistent amplitude range for all signals, which improves the stability of subsequent processing steps, including HPSS. This can help in balancing the energy distribution in the spectrogram, giving HPSS more reliable data to work with.

3. Adjusting the margin Parameter in HPSS

```
# Separate with different margin values
margins = [1, 2, 4, 8, 16]
D_harmonic_list = []
D_percussive_list = []

for margin in margins:
    D_harmonic_temp, D_percussive_temp = librosa.decompose.hpss(D, margin=margin)
    D_harmonic_list.append(D_harmonic_temp)
    D_percussive_list.append(D_percussive_temp)
```

Experimenting with different margin values allows us to control the separation more precisely. Lower margins allow more mixed elements, while higher margins strictly separate the harmonic and percussive components. By testing multiple values, we can determine which margin best suits our audio, balancing the separation quality and preserving key components.

Assignment 3:

a):

Improvements:

1. Applying Non-Local Filtering for Repetitive Instrumental Removal

```
S_filter = librosa.decompose.nn_filter(S_full,
                                       aggregate=np.median,
                                       metric='cosine',
                                       width=int(librosa.time_to_frames(2, sr=sr)))
S_filter = np.minimum(S_full, S_filter)
```

   This step reduces the background instrumental noise by filtering out repetitive sounds, typically associated with instruments. The "nn_filter" function aggregates similar frames, minimizing the instrumental component while retaining unique vocal characteristics.

2. Creating Soft Masks for Separation

```
margin_i, margin_v = 2, 10
power = 2
mask_i = librosa.util.softmask(S_filter, margin_i * (S_full - S_filter), power=power)
mask_v = librosa.util.softmask(S_full - S_filter, margin_v * S_filter, power=power)
```

   The masks, mask_i (for instruments) and mask_v (for vocals), are created using soft-masking. Adjusting margin_i and margin_v determines the strength of the separation. In this code, margin_v is set higher to allow more of the vocal components to pass through, while margin_i suppresses the instrumental part.

3. Vocal Isolation and Saving In the first code, after separating the spectrogram components, the vocal component (S_foreground) is transformed back to a time-domain audio signal and saved:

```
y_vocals = librosa.istft(S_foreground * phase)
sf.write('track04_vocals_only.wav', y_vocals, sr)
```

   This section is not present in the second code. The librosa.istft function reconstructs the vocals-only audio from the spectrogram. By saving this reconstruction to wav, the first code produces an output file with isolated vocals, which is essential for creating a usable audio file.

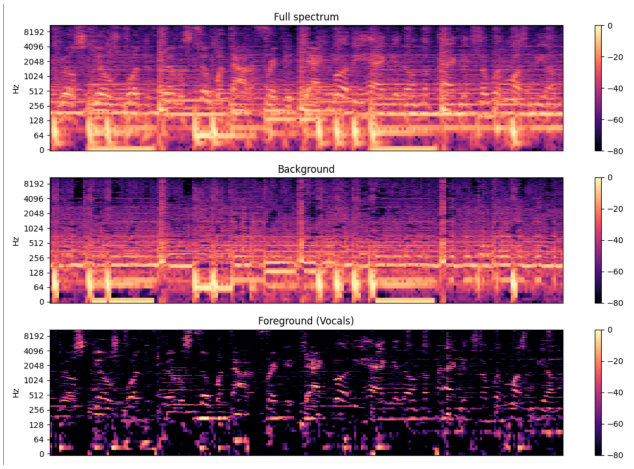

Figure 6: Spectrogram of cheese N pot-C(vocal only)

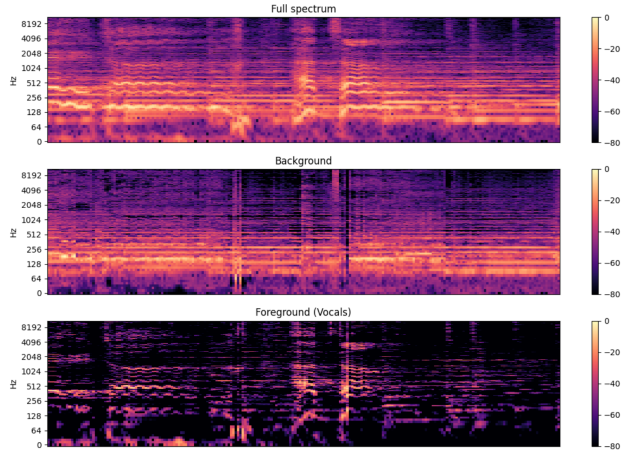

Figure 7: Spectrogram of track04(vocal only)

b):

Improvements:

1. High-Pass Filtering to Reduce Instrumental Noise

```
def highpass_filter(data, cutoff=200, fs=22050, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return lfilter(b, a, data)
```

```
# Apply high-pass filter before processing
y_filtered = highpass_filter(y, cutoff=200, fs=sr)
```

This function applies a high-pass filter to remove frequencies below 200 Hz, which typically contain bass and other low-frequency instruments. This step reduces noise in the vocal signal and makes separation cleaner by minimizing the influence of non-vocal components in the lower spectrum.

2. Non-Local Filtering with Adaptive Frame Width

```
# Compute the spectrogram and apply non-local filtering
S_full, phase = librosa.magphase(librosa.stft(y_filtered))
frame_width = int(librosa.time_to_frames(2, sr=sr))
S_filter = librosa.decompose.nn_filter(S_full, aggregate=np.median, metric='cosine', wi
S_filter = np.minimum(S_full, S_filter)
```

Using a wider "frame width" of 2 seconds helps better identify and retain non-repetitive vocal elements, which often vary more than background instruments. The "nn filter" smooths the background by averaging similar frames, making it easier to distinguish between repetitive instrumental sections and variable vocal components.

3. Enhanced Soft-Masking for Clearer Separation

```
# Adjust the mask margins for better separation
margin_i, margin_v = 2, 15  # Higher margin for vocals
power = 2

mask_i = librosa.util.softmask(S_filter, margin_i * (S_full - S_filter), power=power)
mask_v = librosa.util.softmask(S_full - S_filter, margin_v * S_filter, power=power)

# Separate vocals and instrumentals
S_foreground = mask_v * S_full
S_background = mask_i * S_full
```

The "margin v" for the vocal mask is set higher (15) to make the separation stronger for vocals. This suppresses instrumental components more effectively, allowing only the vocal signal to pass through. The "softmask" function uses this margin to create a "soft" filter that is more flexible than a binary mask, improving the naturalness of the separated vocals.

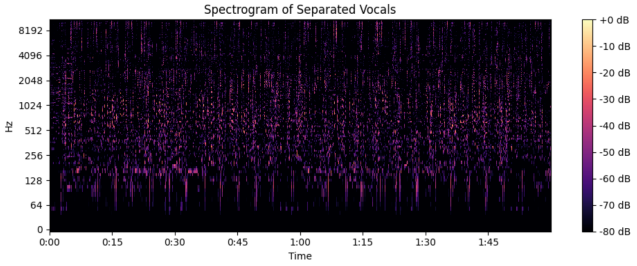

Figure 8: Spectrogram of cheeseN adapted

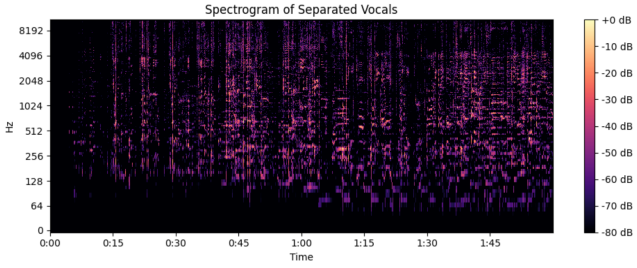

Figure 9: Spectrogram of track04 adapted