Non-Negative Matrix Factorization for Audio Source Separation

Prashant Bharti 202251102 **Utkarsh Rana 202251148**
Vinod Tembhurne 202251157 **Vivek Sonkar 202251158**

Based on Paper: *Non-negative matrix factorization for speech/music separation using source dependent decomposition rank, temporal continuity term and filtering.*

1 Objective

Separating sources from an audio mixture is a fundamental problem in signal processing, commonly known as source separation. A typical scenario is speech/music separation, where one aims to isolate the human speech from background music in a recording.

Traditional methods (like ICA or PCA) fail when sources are not statistically independent or when signals are highly correlated. A more powerful approach is Non-negative Matrix Factorization (NMF), which leverages the non-negativity of magnitude spectrograms.

This report is based on the paper: "Non-negative matrix factorization for speech/-music separation using source dependent decomposition rank, temporal continuity term and filtering[1]."

Ideas mentioned in the paper

NMF is a recently well-known method for separating speech from music signal as a single channel source separation problem. Spectrogram of each source signal is factorized as a multiplication of two matrices known as basis and weight matrices (iteratively updating with respect to cost function).

Contributions of the paper

- **Drawback 1:** In standard NMF, each frame of signal is considered as an independent observation.

  Solution 1: For overcoming, a regularization term is added to the cost function to consider spectral temporal continuity.

- **Drawback 2:** Same decomposition rank is usually used for different sources.

  Solution 2: It is proposed to use different decomposition ranks for speech and music signals as different sources.

- **Solution 3:** It is proposed to apply a filter to the signals estimated by NMF. The filter is constructed by signals estimated by our regularized NMF method.

In our Work:

We try to, in general, separate a mixed audio of two different sources (one drum and other piano instrument) using NMF. We explain the theoretical foundation, see the role of **Wiener filtering[1]**, and connect these with the Python implementation. Also we plot the spectrogram of the resulting audio tracks.

2 Theoretical Background

2.1 Spectrogram Representation

An audio signal $x(t)$ is transformed into a time-frequency representation using the Short-Time Fourier Transform (STFT):

$$X(f,t) = \sum_n x(n)w(t-n)e^{-j2\pi fn}$$

where w is a window function. The magnitude spectrogram $V = |X(f,t)|$ is then used for separation.

2.2 Non-negative Matrix Factorization (NMF)

Given a non-negative spectrogram $V \in \mathbb{R}_+^{F\times T}$ (frequency bins F, time frames T), NMF approximates it as:

$$V \approx WH$$

where:

- $W \in \mathbb{R}_+^{F\times K}$ is the basis matrix (spectral patterns).
- $H \in \mathbb{R}_+^{K\times T}$ is the activation matrix (temporal activations).
- K is the rank of the decomposition, which controls how detailed the representation is.

This decomposition works well for audio because:
- Speech and music are additive in spectrogram space.
- Both are naturally non-negative.

2.3 Source Separation with NMF

If the mixture consists of speech and music, we assume:

$$V \approx V^s + V^m = W_s H_s + W_m H_m$$

where:

- W_s, H_s represent the speech components.
- W_m, H_m represent the music components.

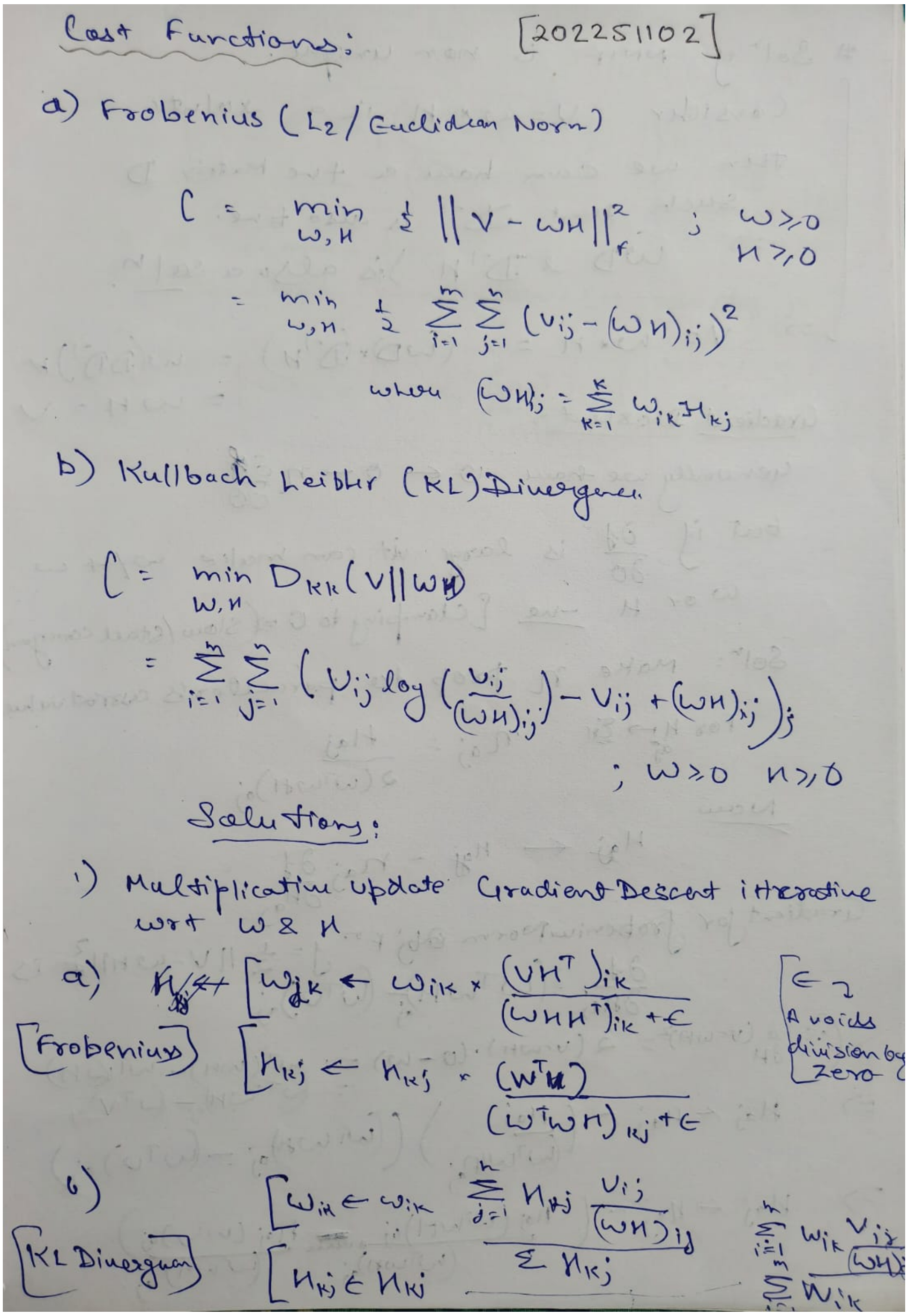
By clustering or pre-training dictionaries for speech and music, we can assign subsets of W and H to the respective sources.

2.4 Cost Function

NMF optimization is performed by minimizing the **Kullback-Leibler (KL) divergence** between V and WH:

$$D_{KL}(V||WH) = \sum_{f,t} \left(V_{ft} \log \frac{V_{ft}}{(WH)_{ft}} - V_{ft} + (WH)_{ft} \right)$$

Multiplicative update rules are then applied to update W and H iteratively. Following is some detail regarding the cost functions used:



Cost Functions: [20225I102]

a) Frobenius (L2 / Euclidean Norm)

$$C = \min_{W,H} \frac{1}{2} \|V - WH\|_F^2 \; ; \; W \geq 0, \; H \geq 0$$

$$= \min_{W,H} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} (V_{ij} - (WH)_{ij})^2$$

where $(WH)_{ij} = \sum_{k=1}^{K} W_{ik} H_{kj}$

b) Kullback Leibler (KL) Divergence

$$C = \min_{W,H} D_{KL}(V||WH)$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{n} \left(V_{ij} \log \left(\frac{V_{ij}}{(WH)_{ij}} \right) - V_{ij} + (WH)_{ij} \right) \; ; \; W \geq 0 \;\; H \geq 0$$

Solutions:

1) Multiplicative update Gradient Descent itterative wrt W & H

a) [Frobenius]

$$W_{ik} \leftarrow W_{ik} \times \frac{(VH^T)_{ik}}{(WHH^T)_{ik} + \epsilon}$$

$$H_{kj} \leftarrow H_{kj} \times \frac{(W^T V)}{(W^T W H)_{kj} + \epsilon}$$

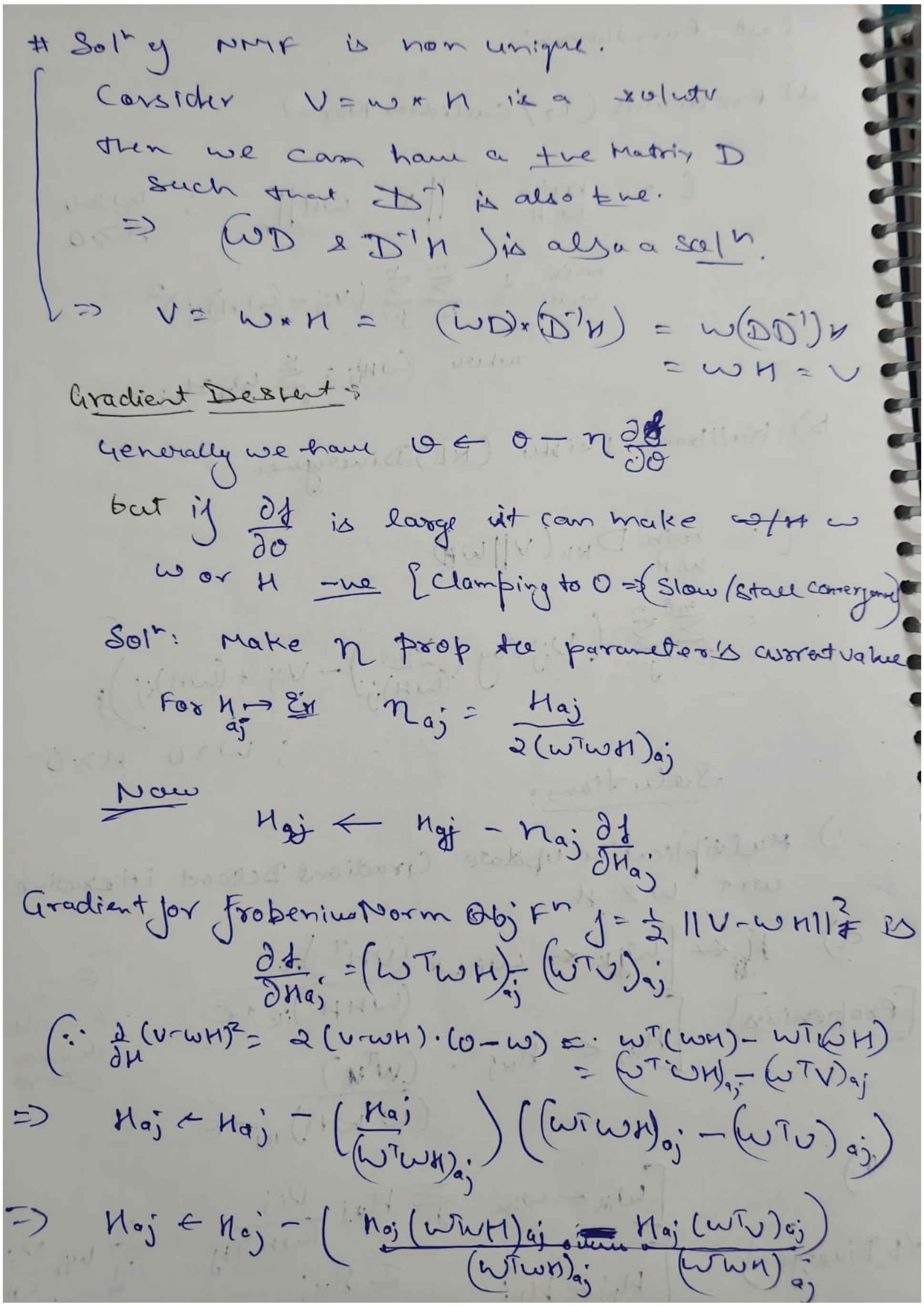
[ϵ → Avoids division by zero]

b) [KL Divergence]

$$W_{ik} \leftarrow W_{ik} \frac{\sum_{j=1}^{n} H_{kj} \frac{V_{ij}}{(WH)_{ij}}}{\sum H_{kj}}$$

$$H_{kj} \leftarrow H_{kj} \frac{\sum_{i=1}^{m} W_{ik} \frac{V_{ij}}{(WH)_{ij}}}{\sum_{i=1}^{m} W_{ik}}$$

Figure 1: Cost function.


Sol^n of NMF is non unique.

Consider $V = W * H$ is a solution

then we can have a +ve Matrix D such that D^{-1} is also +ve.

$\Rightarrow$ (WD & $D^{-1}H$) is also a sol^n.

$\Rightarrow V = W * H = (WD) * (D^{-1}H) = W(DD^{-1})H = WH = V$

Gradient Descent:

Generally we have $\theta \leftarrow \theta - \eta \frac{\partial f}{\partial \theta}$

but if $\frac{\partial f}{\partial \theta}$ is large it can make W/H W or H −ve [Clamping to 0 ⇒ (Slow/stall convergence)]

Sol^n: Make η prop to parameter's current value

For $H_{aj} \rightarrow$ Eq. $\eta_{aj} = \frac{H_{aj}}{2(W^TWH)_{aj}}$

Now

$$H_{aj} \leftarrow H_{aj} - \eta_{aj} \frac{\partial f}{\partial H_{aj}}$$

Gradient for Frobenius Norm Obj F^n $f = \frac{1}{2}\|V - WH\|_F^2$ is

$$\frac{\partial f}{\partial H_{aj}} = (W^TWH)_{aj} - (W^TV)_{aj}$$

$$\left(\because \frac{\partial}{\partial H}(V - WH)^2 = 2(V - WH)\cdot(0 - W) = W^T(WH) - W^T(V) = (W^TWH)_{aj} - (W^TV)_{aj}\right.$$

$$\Rightarrow H_{aj} \leftarrow H_{aj} - \left(\frac{H_{aj}}{(W^TWH)_{aj}}\right)\left((W^TWH)_{aj} - (W^TV)_{aj}\right)$$

$$\Rightarrow H_{aj} \leftarrow H_{aj} - \left(\frac{H_{aj}(W^TWH)_{aj}}{(W^TWH)_{aj}} - \frac{H_{aj}(W^TV)_{aj}}{(W^TWH)_{aj}}\right)$$

Figure 2: Cost function derivation.

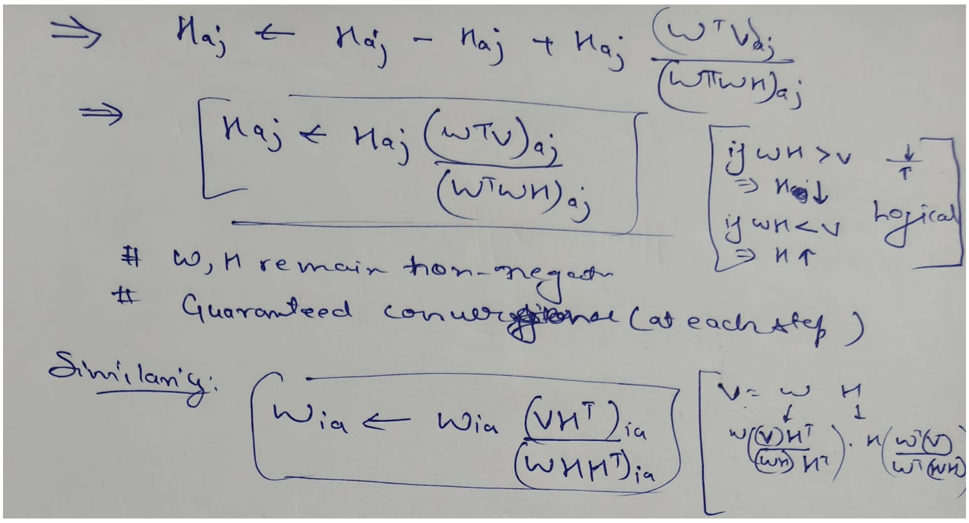


Figure 3: Cost function intuition.

2.5 Wiener Filtering

After NMF, the separated spectrograms $\hat{V}^s$ and $\hat{V}^m$ are estimated. To reconstruct time-domain signals, Wiener filtering [1] is applied:

$$\hat{S}(f,t) = \frac{\hat{V}^s(f,t)}{\hat{V}^s(f,t) + \hat{V}^m(f,t)} X(f,t)$$

$$\hat{M}(f,t) = \frac{\hat{V}^m(f,t)}{\hat{V}^s(f,t) + \hat{V}^m(f,t)} X(f,t)$$

This ensures that the separated signals add up to the original mixture. ALso it reduces interference between separated sources.

3 Python Implementation (`nmf4_wiener.py`)

3.1 Workflow

The Python implementation follows these steps:

1. Loading audio files (speech, music, mixture)
2. Compute STFT of mixture
3. Apply NMF to magnitude spectrogram
4. Split dictionary W and activations H between sources
5. Reconstruct speech and music spectrograms
6. Apply Wiener filtering
7. Perform inverse STFT to recover audio signals
8. Save separated audio

3.2 Dataset Preparation

To ensure the Python script can correctly access the audio files, dataset is organized into the following directory structure:

- `data/speech/` → Clean speech files(here drum beats).
- `data/music/` → Clean music files(here piano tunes).
- `data/mixture/` → Mixture files speech + music (here drum + piano).

This structure ensures that the separation algorithm can easily locate and process the source and mixed audio signals.

3.3 Code Explanation

Imports

```
def stft(y, n_fft=1024, hop_length=None):
    if hop_length is None:
        hop_length = n_fft // 4
    return librosa.stft(y, n_fft=n_fft, hop_length=hop_length)
```

NMF is scikit-learn's Non-negative Matrix Factorization solver. **librosa** is used for audio processing. **soundfile** is used for saving audio.

STFT and ISTFT

These functions are responsible for converting time-domain audio to a spectrogram and back. The parameter n_fft=1024 corresponds to a 64 ms window at 16 kHz, which determines the frequency resolution.

```
def stft(y, n_fft=1024, hop_length=None):
    if hop_length is None:
        hop_length = n_fft // 4
    return librosa.stft(y, n_fft=n_fft, hop_length=hop_length)
```

```
def istft(S, hop_length=None):
    return librosa.istft(S, hop_length=hop_length)
```

```
def magnitude_phase(S):
    return np.abs(S), np.angle(S)
```

Spectrogram Plotting

An audio spectrogram is a visual representation of an audio signal's frequencies and their intensity over time. It displays frequency on the vertical (y) axis, time on the horizontal (x) axis, and the loudness of a particular frequency and time is shown by the color brightness of the point.

```
def plot_spectrogram(V, sr, hop_length, title, out_path):
    """Plot and save spectrogram as image"""
    plt.figure(figsize=(10, 4))
    librosa.display.specshow(
        librosa.amplitude_to_db(V, ref=np.max),
        sr=sr,
        hop_length=hop_length,
        y_axis='log',
        x_axis='time'
    )
    plt.colorbar(format='%+2.0f dB')
    plt.title(title)
    plt.tight_layout()
    plt.savefig(out_path)
    plt.close()
```

NMF Training

This function learns the dictionary W for either speech or music using Kullback-Leibler (KL) divergence.

```
def nmf_train(V, rank=20, n_iter=100):
    """Train NMF basis (dictionary) from magnitude spectrogram."""
    model = NMF(n_components=rank, init='random', max_iter=n_iter,
    solver='mu',
                 beta_loss='kullback-leibler', random_state=0)
    W = model.fit_transform(V + EPS)
    H = model.components_
    return W, H
```

Wiener Filter

This function creates soft masks for each source and returns the Wiener-filtered estimates, which refines the separated signals.

```python
def wiener_filter(V, estimates, eps=EPS):
    """
    Apply Wiener filtering given mixture V and estimated sources.
    estimates: list of magnitude estimates (|S_i|)
    """
    estimates = np.stack(estimates, axis=0)  # shape: (n_sources, F, T)
    denom = np.sum(estimates, axis=0) + eps
    masks = estimates / denom
    return [masks[i] * V for i in range(len(estimates))]
```

Separation Step

This part of the code splits the activations into speech and music components, which are then refined by the Wiener filter.

```python
def separate_sources(mix_file, B_s, B_m, sr=16000, n_fft=1024, hop_length=None,
out_dir="outputs", n_iter=200):
    """Perform source separation using fixed dictionaries B_s and B_m"""
    if hop_length is None:
        hop_length = n_fft // 4

    # Load mixture
    y, sr = librosa.load(mix_file, sr=sr)
    S = librosa.stft(y, n_fft=n_fft, hop_length=hop_length)
    V, phase = magnitude_phase(S)

    # Stack dictionaries
    B = np.concatenate([B_s, B_m], axis=1)  # (F, K_total)
    K_s = B_s.shape[1]

    # Initialize activations
    H = np.abs(np.random.rand(B.shape[1], V.shape[1])) + EPS

    # Multiplicative updates (KL divergence)
    for it in range(n_iter):
        V_hat = B @ H + EPS
        H *= (B.T @ (V / V_hat)) / (B.T.sum(axis=1)[:, None] + EPS)

    # Reconstruct estimates
    speech_est = B[:, :K_s] @ H[:K_s, :]
    music_est = B[:, K_s:] @ H[K_s:, :]

    # Apply Wiener filtering
    estimates = wiener_filter(V, [speech_est, music_est])

    os.makedirs(out_dir, exist_ok=True)

    # Reconstruct signals and save
    for est, name in zip(estimates, ["speech", "music"]):
        S_est = est * np.exp(1j * phase)
        y_est = istft(S_est, hop_length=hop_length)

        out_path = os.path.join(out_dir, f"{name}_from_{os.path.basename(mix_file).replace('.
        mp3', '.wav')}")
        sf.write(out_path, y_est, sr)

        plot_spectrogram(est, sr, hop_length, f"{name} spectrogram", out_path + ".png")

    # Save mixture spectrogram
    plot_spectrogram(V, sr, hop_length, "Mixture spectrogram",
                     os.path.join(out_dir, f"mixture_{os.path.basename(mix_file)}.png"))

    print(f" Separated {mix_file} → audio + spectrograms saved in {out_dir}")
```

Reconstruction

The estimated magnitude is combined with the original phase information from the mixture, and the inverse STFT is performed to convert the spectrogram back to a time-domain waveform.

```python
# Reconstruct signals and save
for est, name in zip(estimates, ["speech", "music"]):
    S_est = est * np.exp(1j * phase)
    y_est = istft(S_est, hop_length=hop_length)

    out_path = os.path.join(out_dir, f"{name}_from_{os.path.basename(mix_file).replace('.
mp3', '.wav')}")
    sf.write(out_path, y_est, sr)

    plot_spectrogram(est, sr, hop_length, f"{name} spectrogram", out_path + ".png")

# Save mixture spectrogram
plot_spectrogram(V, sr, hop_length, "Mixture spectrogram",
                 os.path.join(out_dir, f"mixture_{os.path.basename(mix_file)}.png"))

print(f" Separated {mix_file} → audio + spectrograms saved in {out_dir}")
```

3.3 Running the Code

In terminal:

```
cd "D:\project_root␣-␣Copy"
python nmf4_wiener.py --speech_folder data/speech --music_folder data/
    music --mixture_folder data/mixture --out_dir results
```

Outputs:

- `results/speech_from_mix.mp3` : The separated Drum beat.
- `results/music_from_mix.mp3` : The seaparated Piano tones.
- Spectrogram plots (speech/music/mixture). : The spectrogram of each of the three- only drum, only piano, both drum and piano mixed.

The code, datasets, outputs and spectrograms along with this report is attached together in the zip file.

4 Results

Using this approach, the mixture audio (of Drum and Piano mixed)can be separated into speech(only Drum) and music(only Piano) components. The quality depends on:

1. The rank K chosen for NMF(in our approach we used rank 20).
2. Whether dictionaries are trained separately or clustered(in our approach dictionaries are trained separately).
3. Post-processing such as Wiener filtering.

The Spectrograms of the resulting audio sources are shown below:

1. Mixed Signal(both audio sources)

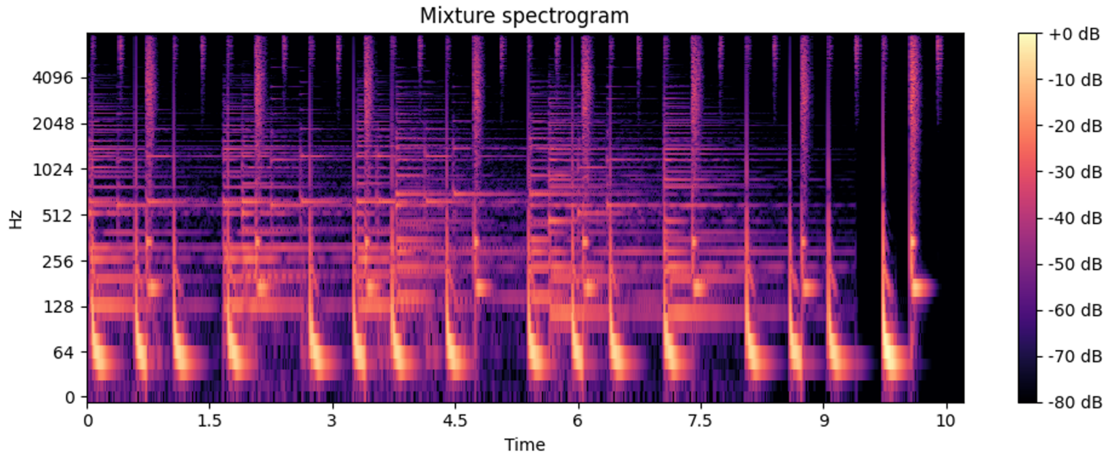


Figure 4: Mixed Signal Spectrogram

2. Separated Signal(first audio source - Drum beats)

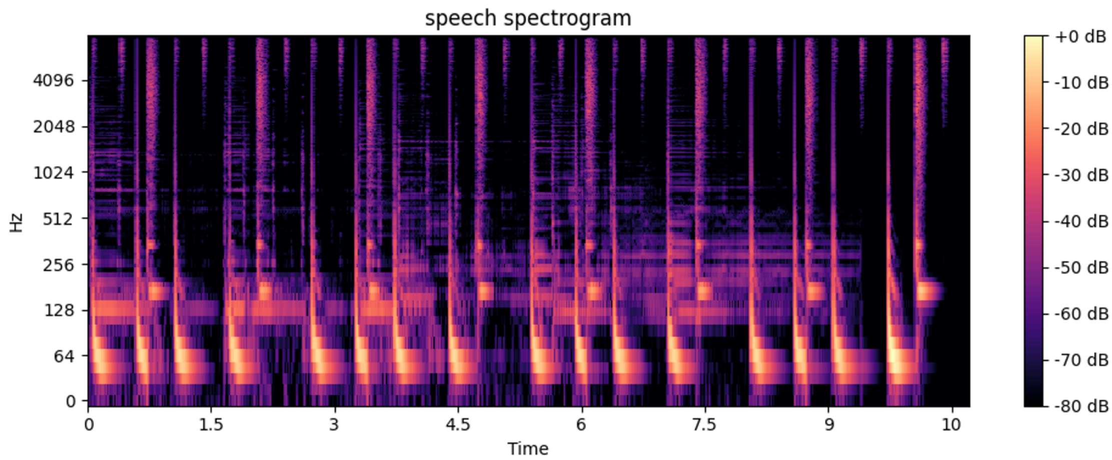


Figure 5: Source 1 (Drum)

3. Mixed Signal(second audio source- Piano)

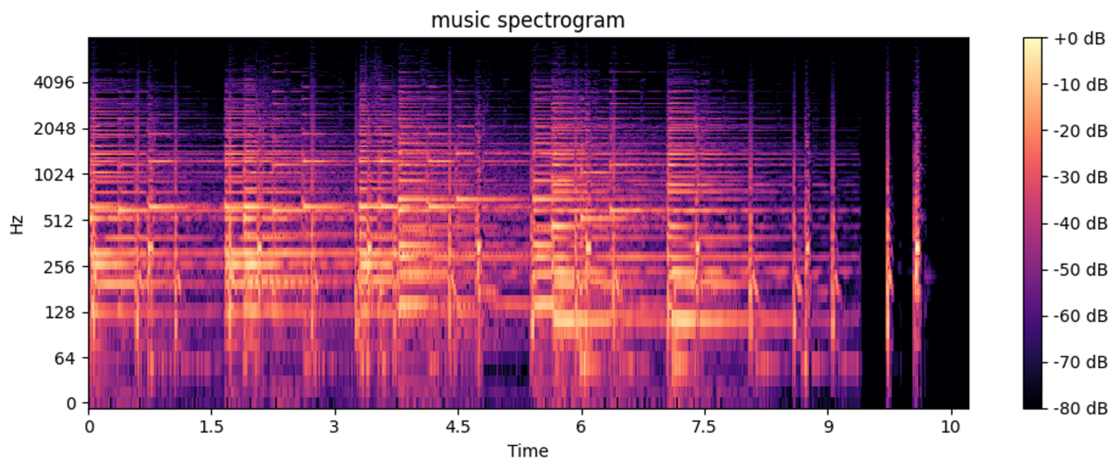


Figure 6: Source 2 (Piano)

5 Conclusion

This report demonstrated how NMF, combined with Wiener filtering, can effectively separate two source audio signals from a mixture of each of the source signals. Also we get the spectrograms of the resulting separated audio tracks.

6 Future improvements

Future works can include:

1. Extending the use case to real world scenarios of separation human speech audio signals from the background noise or music.
2. Using supervised NMF with pre-trained dictionaries.
3. Exploring deep learning-based extensions.

References

[1] S. Abdali and B. NaserSharif, "Non-negative matrix factorization for speech/music separation using source dependent decomposition rank, temporal continuity term and filtering," *Biomedical Signal Processing and Control*, vol. 36, pp. 168–175, 2017.