Speech Recognition에 필요한 Audio 기초

조희철

2020년 10월 27일

0.1 Audio 기초

♠ bit rate & sampling rate

- 파일 크기(byte): (재생시간) x (bit rate) / 8
- 여기서 bit rate(비트전송률)는 초당 얼마나 많은 data를 가지고 있는지를 의미한다. 예를 들어, CD audio는 2-channel, 16 bit(Quantization, bit depth), sampling rate(44100)[1].
- audio file의 비트전송률은 (Windows 10) 파일 속성에서 확인할 수 있다.

$$16 \times 2 \times 44100 = 1,411,200 \text{ bit}$$

- sampling rate: 1초에 들어 있는 data 수.
- bit depth(Quantization): sampling rate이 시간 축인 x축을 얼마나 세분하냐를 결정하는 것이라면, bit depth는 y축을 얼마나 정밀하게 세분하냐를 결정한다.

```
import sox # sox.exe를 설치후, pip install sox --> https://sourceforge.net/projects/sox/

sox.file_info.sample_rate(audio_filename)
sox.file_info.bitdepth(audio_filename)
sox.file_info.bitrate(audio_filename) # '352k'를 float로 변환하는 과정에 bug가 있다.
```

♠ Discrete Fourier Transforms

- Fourier Transform: 음파와 같은 시간에 대한 신호(함수)를 주파수 성분으로 분해할 수 있게 해준다. 주어진 신호를 (서로 다른 주기를 가진) 주기함수들의 결합으로 분해한다. 여기서 사용되는 주기함수는 복소 주기함수($\{e^{-2\pi ik}\}$)들로 구성된다.
- Discrete Fourier Transform(DFT): 컴퓨터에서 처리할 수 있는 data는 이산적이다. 이 이산적인 data에 Fourier Transform을 적용한 것이 Discrete Fourier Transform이다.
- FFT(Fast Fourier Transform): DTF을 좀 더 효율적으로 계산할 수 있게 해주는 알고리즘이다.
- STFT(Short Time Fourier Transform): 우리가 다루어야 하는 음성이나 음악은 하나의 발음이나 음이 계속되는 것이 아니다. 시간에 따라 변한다. 그렇게 때문에 음파를 시간단위로 나누어서 각각에 Fourier Transform

[1] https://en.wikipedia.org/wiki/Sampling_(signal_processing)#Sampling_rate

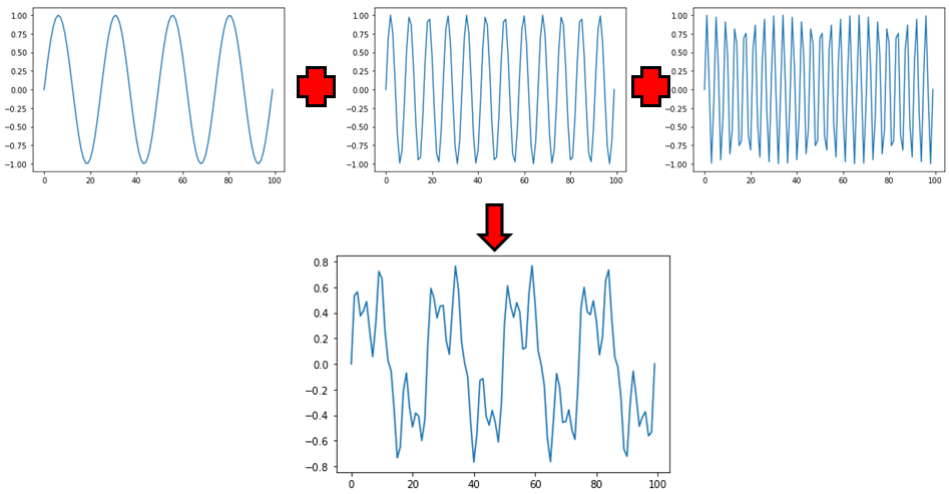


그림 1: Sound(음파)는 단순 Sine파의 결합으로 볼 수 있다. 이 결합된 음파가 어떤 단순파의 결합으로 만들어진 것인지 분석할 수 있게 해주는 것이 Fourier Transform이다. 이런 과정을 time domain(음파)를 frequency domain으로 변환한다고 한다. 그림에서는 각기 다른 주파수의 sine파가 5 : 3 : 2로 결합된 음파를 보여주고 있다. Fourier Transform을 통해 어떤 주파수의 성분이 얼마나 결합되어 있는지 알아낼 수 있다. 이산적인 data를 다루어야 하기 때문에, Discrete Fourier Transform을 사용한다.

을 적용하는 것을 STFT라 한다. 자르는 단위를 frame-length(또는 window-length)라 한다. 그리고 자를때 겹치지 않게 자르지 않고, data의 손실을 막기 위해 겹치는 방식(overlap)으로 자른다. 잘라진 하나의 구간을 frame이라 부른다. 이 frame을 이동시키는 간격을 hop-length라 한다. 당연히, hop-length는 frame-length보다 작아야 frame이 서로 겹치게 된다.

- DTF의 결과로 복수소로 이루어진 벡터가 생성되는데, 복소수를 다루기 어렵기 때문에 절대값이나 절대값의 제곱을 취한다. 절대값 취한 것을 magnitude spectrogram, 절대값의 제곱한 것을 power spectrogram이라 부른다.

```
N = 400; T = 1.0 / 800.0
x = np.linspace(0.0, N*T, N)
y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
yf = np.fft.fft(y) #shape: (400,) complex numbers
```

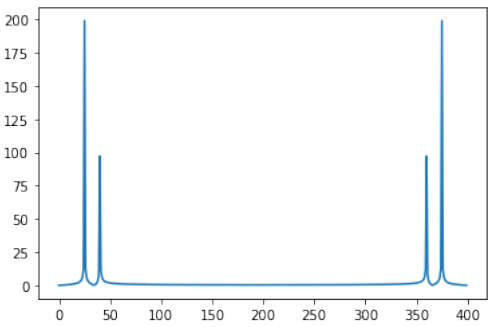


그림 2: DTF: FFT의 absolute 값의 그래프. Nyquist critical frequency[2]에 의해 $-\frac{1}{2T} \sim \frac{1}{2T}$의 주파수 영역이 구해지고, 양의 주파수/음의 주파수에 대한 값이 conjugate가 된다. 대칭적인 결과가 redundant하므로, STFT에서는 절반 크기의 output을 만들어 낸다.

```
scale, sr = librosa.load(audio_filename)
```

[2]sampling time이 $T = \frac{1}{\text{sr}}$ 일 때, $\frac{1}{2T} = \frac{\text{sr}}{2}$ 이상의 주파수는 측정할 수 없다(aliasing-디지털 신호 처리 과정에서 발생하는 노이즈).

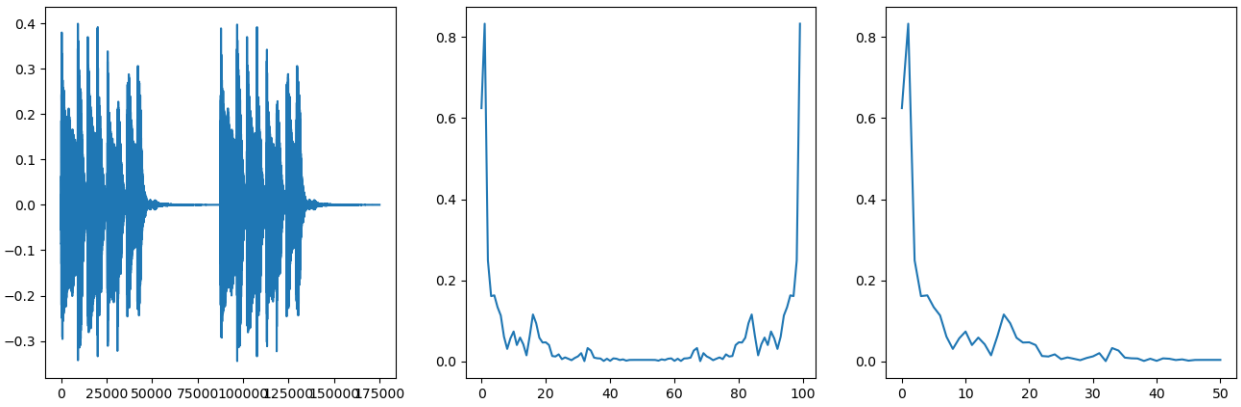


그림 3: fft vs rfft: rfft는 대칭적인 부분을 고려해서 계산량을 줄인다. librosa.stft는 애초에 절반만 return한다.

```
n_fft = 100
fft = np.fft.fft(scale,n=n_fft) # (100,)
rfft = np.fft.rfft(scale,n=n_fft) # (51,)
print(np.allclose(fft[:n_fft//2+1],rfft)) # True
```

```
N = 5
signal = np.random.rand(N)
fft = np.fft.fft(signal)

result = []
for k in range(N):
    temp = 0
    for m in range(N):
        temp += signal[m]*np.exp(-2*np.pi*1j*m*k/N)
    result.append(temp)
```

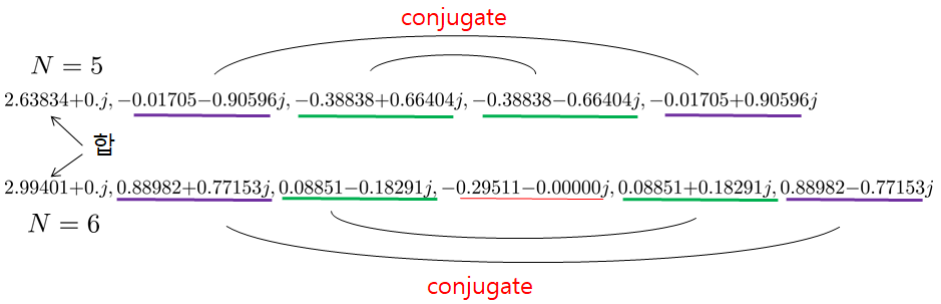
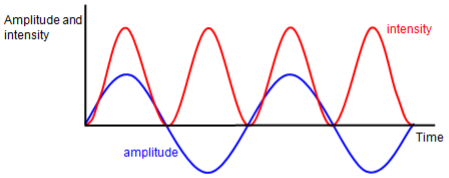


그림 4: FFT의 결과: 첫번째 값은 data의 합이 되고, 그 이후의 값들은 서로 대칭적인 conjugate관계이다. librosa의 stft에서는 hann window를 적용하기 때문에, numpy에서의 fft 결과와는 다르다. 참고로, python에서는 수학에서의 허수단위 $i = \sqrt{-1}$를 j로 표시한다.

♠ 소리의 요소들

- Intensity(I): 단위 면적당 에너지를 뜻한다. amplitude(A)의 제곱에 비례한다. $I \propto A^2$



Source	Intensity	Intensity level	× TOH
Threshold of hearing (TOH)	10^{-12}	0 dB	1
Whisper	10^{-10}	20 dB	10^{2}
Pianissimo	10^{-8}	40 dB	10^{4}
Normal conversation	10^{-6}	60 dB	10^{6}
Fortissimo	10^{-2}	100 dB	10^{10}
Threshold of pain	10	130 dB	10^{13}
Jet take-off	10^{2}	140 dB	10^{14}
Instant perforation of eardrum	10^{4}	160 dB	10^{16}

Table 1.1 from [Müller, FMP, Springer 2015]

그림 5: intensity는 amplitude의 제곱에 비례한다.

- Threshold of hearing(TOH): $10^{-12} W/m^2$
- Decibel: TOH를 기준으로 log값을 취한 것

$$dB(I) = 10 \times \log_{10}\left(\frac{I}{I_{TOH}}\right)$$

- Timbre: Color of sound, Difference between two sounds with same intensity, frequency, duration.
- **배음(Overtone)**: An overtone is any frequency greater than the fundamental frequency of a sound. Using the model of Fourier analysis, the fundamental and the overtones together are called **partials**. **Harmonics**, or more precisely, harmonic partials, are partials whose frequencies are numerical integer multiples of the fundamental(including the fundamental, which is 1 times itself). 이 배음들이 어떻게 결합되어지는가에 따라 음색이나 악기 고유의 소리가 결정된다.

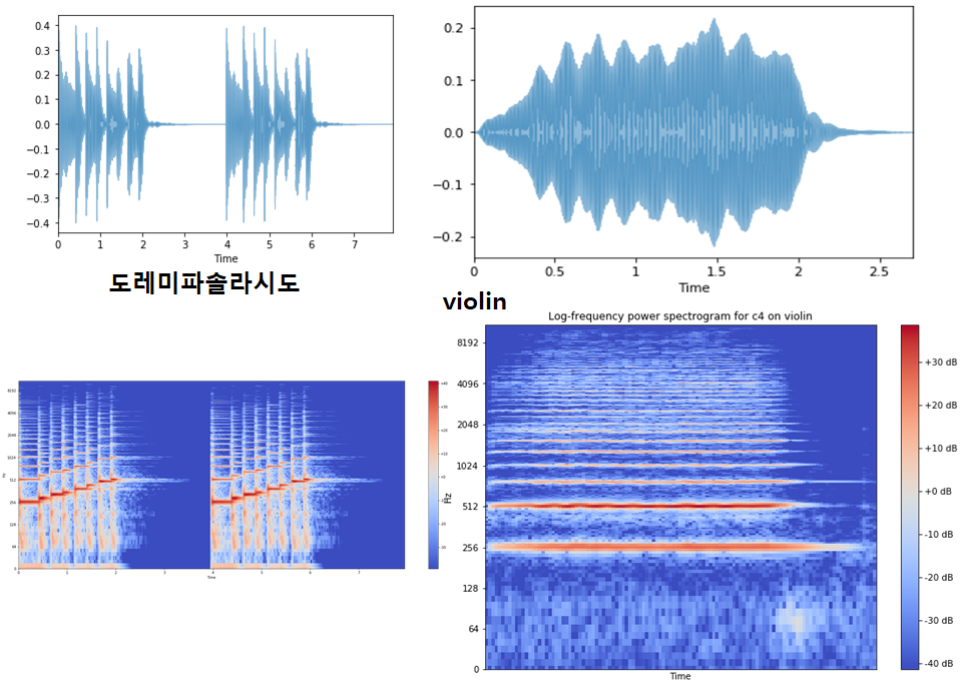


그림 6: 배음(overtone) 구조: fundamental frequency로 256이 보이고, 이것의 정수배에 해당하는 주파수들도 보인다.

- Formant: 음성학에서 인간 성대의 공명에서 발생하는 스펙트럼의 local maximum을 말한다.

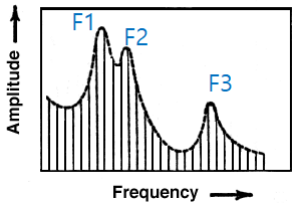


그림 7: Formant: local maximum을 왼쪽 부터 first formant frequency($F1$), second formant frequency($F1$), ... 로 부른다. formant frequency간의 간격이 중요하다. 이 간격(어떤 의미에서 주기)을 잘 파악하는 것이 MFCC이다. 여기서 y축의 Amplitude는 중요하지 않다.

- Additive Synthesis[3]: Additive synthesis is a sound synthesis technique that creates timbre by adding sine waves together.
- waveform 비교: 그림 (8)

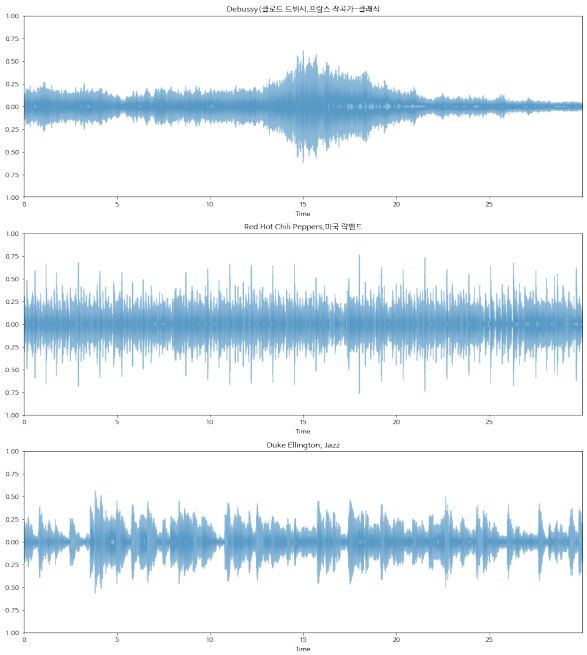


그림 8: waveform 비교: 위에서 부터 클래식음악, 락, 째즈.

♠ Time-domain audio Features

Time-domain Feature들은 전통적인 Machine Learning 분야에서 많이 활용되었으나, 딥러닝이 활성화되면서 활용도가 낮아졌다.

- Amplitude envelope: 각 frame의 최대값을 모아 놓은 것.
- RMS(Root Mean Square Energy): 각 frame에서 제곱값의 평균에 root를 취한 값. 각각의 frame마다 계산된다. 직접 계산할 수도 있고, librosa.feature.rms를 이용해도 된다.
- Zero Crossing Rate: Number of times a signal crosses the horizontal axis.
  - `librosa.feature.zero_crossing_rate`: 각 fram에서 zero crossing 비율. frame 길이 크기의 output이 return된다.
  - `librosa.zero_crossings`: signal의 부호가 바뀌는 point를 잡아낸다. sample 길이 크기의 [True,True, False, ...] 결과가 return된다.

[3] `https://teropa.info/harmonics-explorer/`

♠ Mel Filter Bank

- $m = 2595 \log_{10}(1 + \frac{f}{700})$

$$M(f) = 2595 \log_{10}(1 + \frac{f}{700})$$

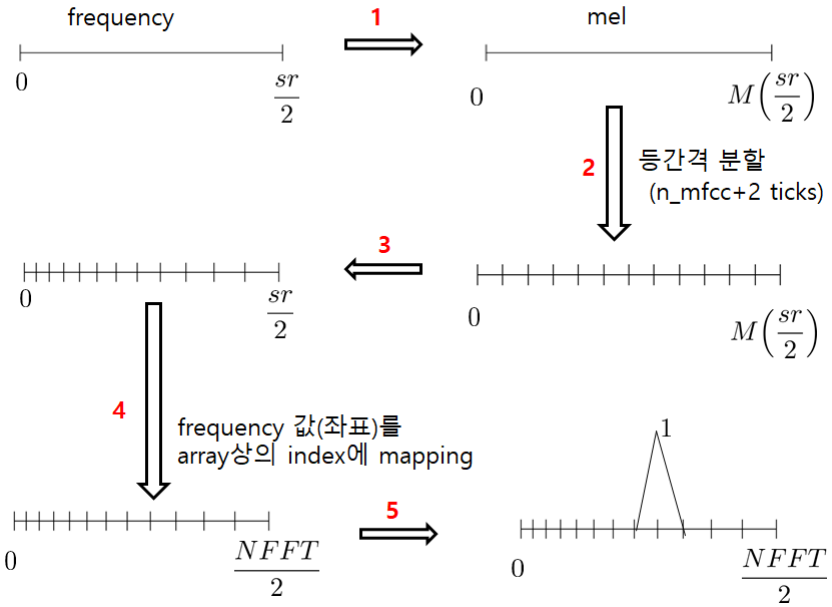


그림 9: mel filter bank(mel basis matrix) 생성 과정: 4번째 과정에서 선택된 주파수는 끝점 포함해서 (n_mfcc+2) 개이고, array index는 0부터 NFFT/2까지 있다. 따라서 $0, 1, 2, \cdots,$ NFFT/2 중에서 (n_mfcc+2) 개 만큼의 index만 선택된다. index중에서 선택된 곳에서는 0 또는 1의 값을 가지게 하고, 그 외의 index들에 대한 값은 (Λ)모양의 직선상에 있게 값을 잡아 줄 수 있다. 그리고, 처음에 최대 주파수를 지정해 주어야 되는데, default로 sampling-rate의 절반을 사용한다. 주파수는 상대적인 것이기 때문에 critical한 것은 아니다.

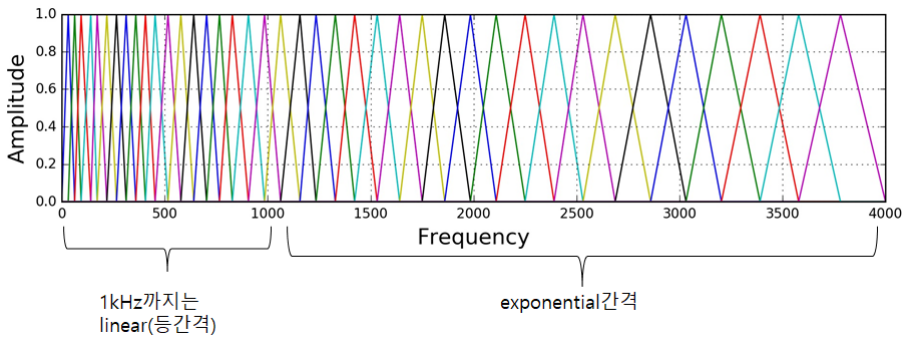


그림 10: mel filter bank

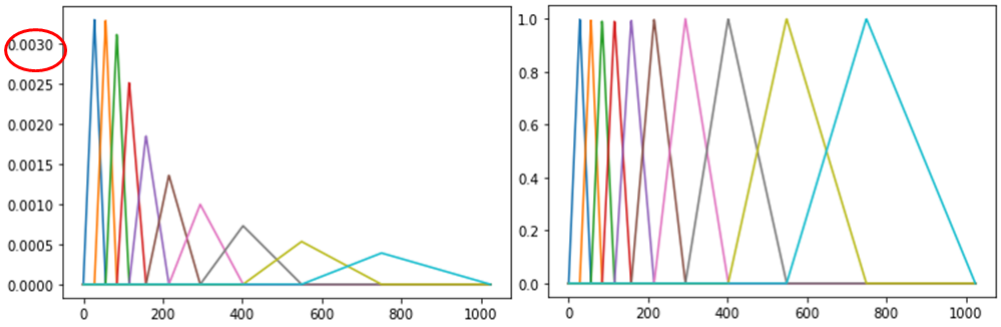


그림 11: librosa.filters.mel: 왼쪽 norm=1(default)로 하면, band의 폭(삼각형의 밑변 길이)으로 값을 나누어준다. 이 경우는 filter weight의 합이 동일하다. 즉 각 column합이 일정하다.
오른쪽: norm=None. normalization이 되지 않았기 때문에, mel spectrogram을 구하면 값이 크다.

♠ MFCC

- MFCC(Mel Frequency Cepstral Coefficients)는 Spectrum of Spectrum이라 할 수 있다. Frequency data에 다시 Discrete Cosine Transform(또는 Inverse DFT)을 적용하기 때문이다.
- Therefore, we can apply Discrete Cosine Transform (DCT) to decorrelate the filter bank coefficients and yield a compressed representation of the filter banks[4].
- MFCC는 log mel-spectrogram을 Discrete Cosine Transform하면 된다. Typically, for Automatic Speech Recognition (ASR), the resulting cepstral coefficients 2-13 are retained and the rest are discarded; `num_ceps = 12`. The reasons for discarding the other coefficients is that they represent fast changes in the filter bank coefficients and these fine details don't contribute to Automatic Speech Recognition (ASR).
- log mel-spectrogram의 feature dimension이 n이면, DCT를 적용해도 n차원이 된다. 이 중에서 앞 부분을 필요한 만큼 선택하면 된다. 첫번째 feature는 값(data의 합)이 너무 작아(음수), 다른 feature에 악영향을 주기 때문에 제거하는 경우도 있다.

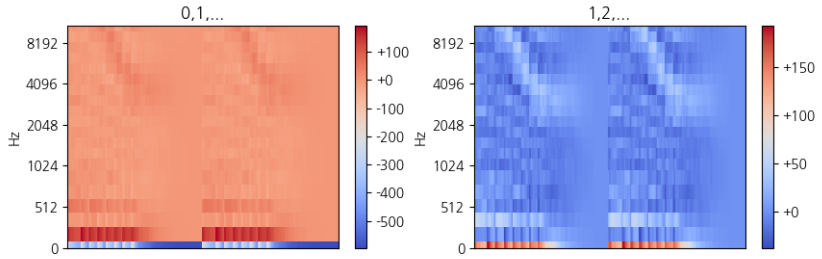
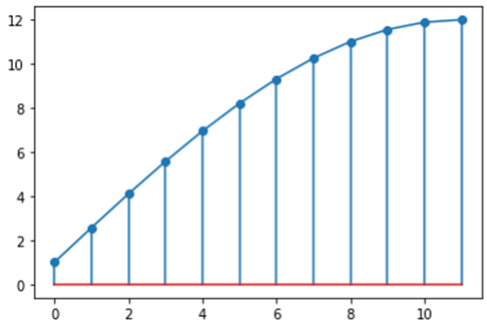


그림 12: MFCC의 첫번째 featrue는 입력 data의 합이다(Fourier/Cosine Transform의 첫번째 값은 data의 합). 첫번째 feature 제거 전/후 비교. 왼쪽은 제일 아래쪽 라인의 값이 너무 작아서 다른 값들이 의미를 가지지 못하게 된다.

- 딥러닝이 활성화되기 이전의 머신러닝에서는 Feature의 선택이 중요했기 때문에, mel filter bank에 기인하여 나타나는 Feature 간의 상관관계를 줄이기 위해서 MFCC가 활용되었다. GMM 같은 모델에서는 이런 상관관계가 제거된 Feature가 유용하게 활용된다.
- 딥러닝이 출현한 이후에는 Feature 간의 비선형 관계를 제거해 버리는 MFCC의 중요성이 낮아졌다.

[4] https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

- MFCC lifter: 결과의 상위 인덱스에 해당하는 값들이 작기 때문에, lifter를 곱해서 증폭시킨다.
  `librosa.feature.mfcc(..., lifter)`



```
(n_mfcc,n_frames) = (12,723)  # mfcc.shape
lifter = 22
n = np.arange(n_mfcc)
lift = np.sin(np.pi * (n+1) / lifter) * lifter / 2  # ==> (n_mfcc,)
#lift = 1 + np.sin(np.pi * (n) / lifter) * lifter / 2

plt.plot(lift)
plt.stem(lift)
```

그림 13: MFCC lifter: MFCC output에 sine 값을 곱해준다.

- MFCC의 장점:
  - spectrum의 디테일은 무시하고, 크게 보는 feature.
  - 배음 구조를 잘 파악한다.
  - 음정(pitch)의 차이는 무시한다. 음정을 무시해도 되는 task에 적합하다. 악보를 그리는 것과 같은 task에는 적합하지 않다.
  - formant[5] 구조를 잘 파악한다.
- MFCC의 단점:
  - noise에 약하다.
  - 음성합성에 적합하지 않다. 압축성이 강한 MFCC로 부터 audio를 생성하거나 복원하는 것이 어렵다.

♠ Preemphasis

- Pre-emphasis is a very simple signal processing method which increases the amplitude of high frequency bands and decrease the amplitudes of lower bands. In simple form it can be implemented as

$$y_t := x_t - \alpha x_{t-1}$$

- 고주파의 노이즈 제거에서 더욱 중요했는데, 요즘 speech recongnition에서는 불필요하다는 견해도 있다[6].
- STFT를 적용하기 전 waveform에 적용하면 된다.

```
from scipy import signal
N = 5; k = 0.97
wav = np.random.rand(N)

result1 = signal.lfilter([1, -k], [1], wav)
result2 = wav - np.array([0,]+list(k*wav[:-1]))

print(np.allclose(result1,result2))
```

[5] https://en.wikipedia.org/wiki/Formant

[6] https://www.quora.com/Why-is-pre-emphasis-i-e-passing-the-speech-signal-through-a-first-order-high-pass-filter-required-i

♠ Windowing Function

- Hann window: 각 프레임의 처음과 끝에서의 불연속을 최소화하기 위해, Hann window function을 곱해서 STFT를 구한다.

$$w(n) = 0.5\Big(1 - \cos\big(\frac{2\pi n}{M-1}\big)\Big), \quad 0 \le n \le M-1$$

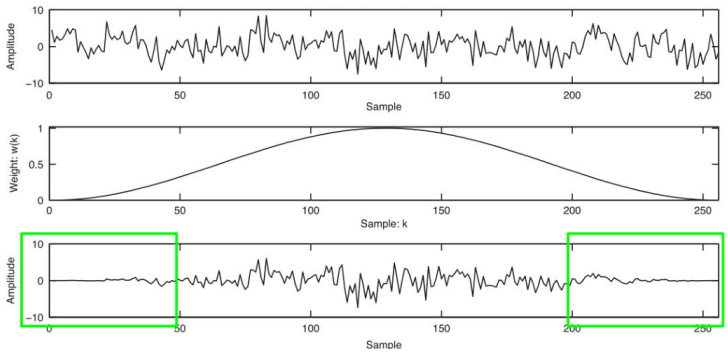


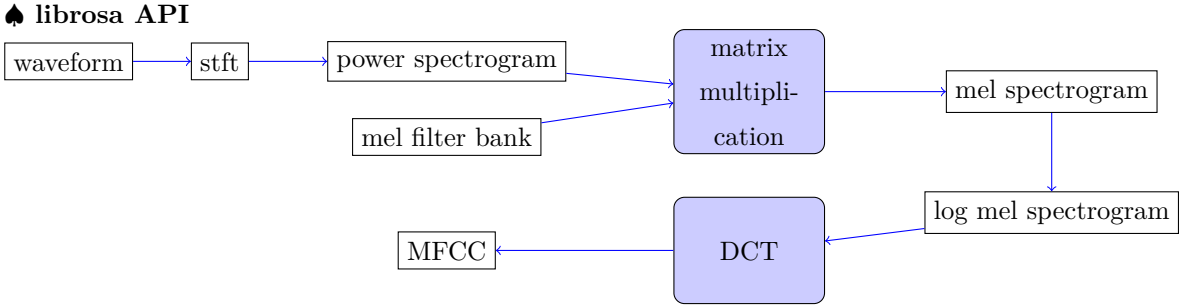
그림 14: Hann Window

- time domain에서 곱해지는 Hann window function은 frequency domain에서의 convolution이 된다. 따라서 frequency를 filtering하는 효과가 있다.

- Hamm window:

$$w(n) = 0.54 - 0.46\cos\big(\frac{2\pi n}{M-1}\big), \quad 0 \le n \le M-1$$

♠ librosa API



	librosa	Tacotron
sampling rate	22,050	24,000
n_fft	2,048	2,048
win_length	n_fft = 2048	1200
hop_length	(win_length)×0.25 = 512	300

표 1: lirosa default argument

- librosa feature: `https://librosa.org/doc/latest/feature.html`

- librosa.load: wav파일 load. $-1 \sim 1$ 사이 값. duration(sec)을 옵션으로 줄 수 있다. duration이 주어지면, min(duration, data-length) 만큼 load한다.

librosa.load	librosa.effects.trim	librosa.display.waveplot
librosa.specshow	librosa.stft	librosa.display.power_to_db
librosa.amplitude_to_db	librosa.core.magphase	librosa.feature.melspectrogram
librosa.feature.mfcc	librosa.feature.delta	librosa.cqt
librosa.feature.chroma_stft	librosa.feature.chroma_cqt	librosa.feature.spectral_centroid
librosa.feature.spectral_bandwidth	librosa.feature.spectral_rolloff	librosa.effects.hpss
librosa.effects.harmonic	librosa.effects.percussive	librosa.beat.beat_track
librosa.beat.tempo	librosa.feature.rms	librosa.frames_to_time
librosa.onset.onset_detect		

표 2: librosa 주요 API

  - scipy.io.wavfile.read(filename): 파일의 sampling rate대로 읽어 온다. return 값은 정수인데, 파일의 bit depth에 따라 값의 범위가 다르다[7]. bit depth $= D$ 이면, $-2^{D-1} \sim 2^{D-1}$

- librosa.effects.trim(silence 제거): Trim leading and trailing silence from an audio signal.
- librosa.display.waveplot(y,sr,alpha=0.5): `sample_rate`을 넣어줌으로써, x축이 시간축으로 잘 보여진다. alpah는 선명도.
- librosa.display.specshow: 세로축(axis=0, 아래에서 위), 가로축(axis=1, 왼쪽에서 오른쪽).
  - x축: `sample_rate, hop_length`을 넣어줌으로써, x축이 시간축으로 잘 보여진다. (입력길이 `* hop_length/sample_rate`) = duration(time). `x_axis='time'`이 설정되어야 눈금이 표시된다.
  - y축: `y_axis='log'`로 설정하면, log scale로 보여진다. y축 값은 0 ~ 8000 정도의 값이 입력되는 data에 상관없이 표시된다. `y_coords`를 넣어주면 y축 눈금이 맞게 표시된다. 어쨌든 상대적인 위치를 본다고 하면, `y_coords`를 넣지 않아도 된다.
  - `librosa.fft_frequencies(sr,n_fft=16000)`: (8001,)
- librosa 아닌 다른 package API:
  - plt.specgram:
  - scipy.signal.spectrogram: return되는 값의 범위가 librosa.stft와 완전히 다르다.

```
nfft = 2048; hop_length = 512; sr=22050 # sampling rate
noverlap = nfft - hop_length
spectrum, freqs, t, im = plt.specgram(y, NFFT=nfft, Fs=sr, Fc=0,noverlap=noverlap,
    sides='default', mode='default', scale='dB')
print(spectrum.shape,freqs.shape,t.shape) # (1025, 338) (1025,) (338,)

freqs, t, spec = signal.spectrogram(y, fs=sr, window='hann',
    nperseg=nfft,noverlap=noverlap,detrend=False)
print(freqs.shape, t.shape,spec.shape) # (1025,) (338,) (1025, 338)
```

- `librosa.stft`: $\Longrightarrow$ ($\frac{\text{n\_fft}}{2} + 1$, # of frames)
  - `n_fft` $\geq$ `win_length` 이어야 한다.

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.read.html

- `win_length` 크기의 hann window를 만든 후, 양쪽에 zero padding을 하여, `n_fft` 크기로 만든다.
- 내부적으로 `np.fft.rfft`를 사용하는데, 각 frame마다 $\frac{\text{n\_fft}}{2} + 1$ 크기의 결과가 return된다.
- frame의 크기는 `win_length`가 아니고, `n_fft` 크기로 만들어진다. 여기에 hann window를 곱한다. 당연히 hann window의 크기도 `n_fft`이지만, 가운데 `win_length` 만큼만 살아있기 때문에, 계산에 반영되는 부분은 `win_length` 크기 만큼이다.

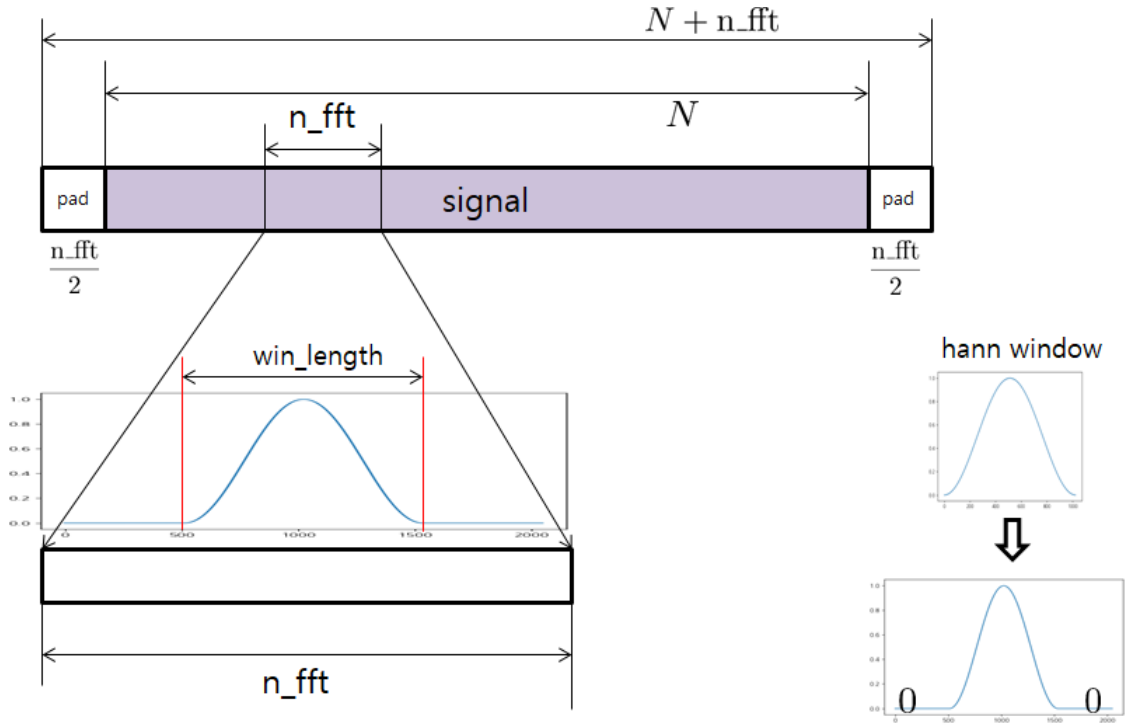


그림 15: librosa.stft: 입력 data의 길이가 N이면, 양쪽에 padding을 붙혀 N + n_fft 크기로 바꾸어서 STFT가 적용된다. frame의 크기는 n_fft 만큼 생성되고, hann windows가 곱해지면서 win_length 만큼만 남기는 역할을 한다.

- `librosa.power_to_db`: Convert a power spectrogram (amplitude squared) to decibel (dB) units. 즉 제곱한 것을 넣어야 한다.

$$y = 10 \log_{10} \left(\frac{x}{\text{ref}} \right)$$

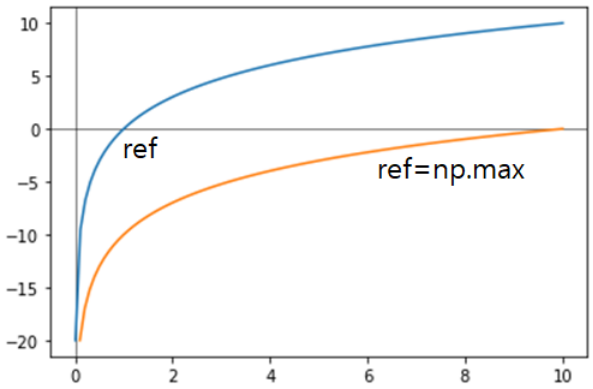


그림 16: Decibel: ref=np.max로 설정하여 결과를 음수로 만들 수 있다. Tacotron에서는 ref=1로 하는 대신 결과에서 20을 뺐다. $\max(-100, 20 \log_{10} x) - 20$

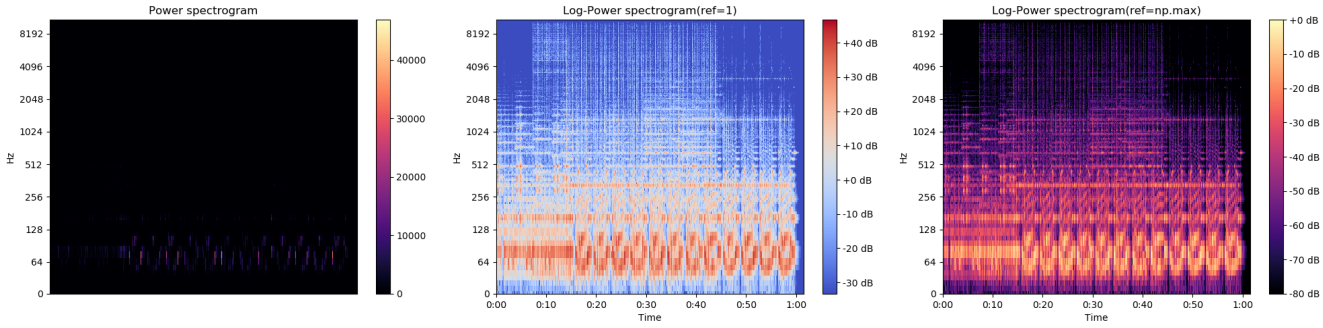


그림 17: power spectrogrma vs log-power spectrogram: 오른쪽이 log를 취하지 않은 power spectrogram. 가운데, 왼쪽이 librosa.power_to_db(S**2). 각각 ref=1, np.max.

- `librosa.amplitude_to_db`: Convert an amplitude spectrogram to dB-scaled spectrogram.This is equivalent to `power_to_db(S**2)`, but is provided for convenience. 제곱하지 않은 것을 넣으면, 내부에서 제곱해 준다.
- `librosa.core.magphase`: Separate a complex-valued spectrogram D into its magnitude (S) and phase (P) components, so that D = S * P.
- amplitude vs magnitude: amplitude는 부호를 가진 vector이고, magnitude는 scalar 값이다.

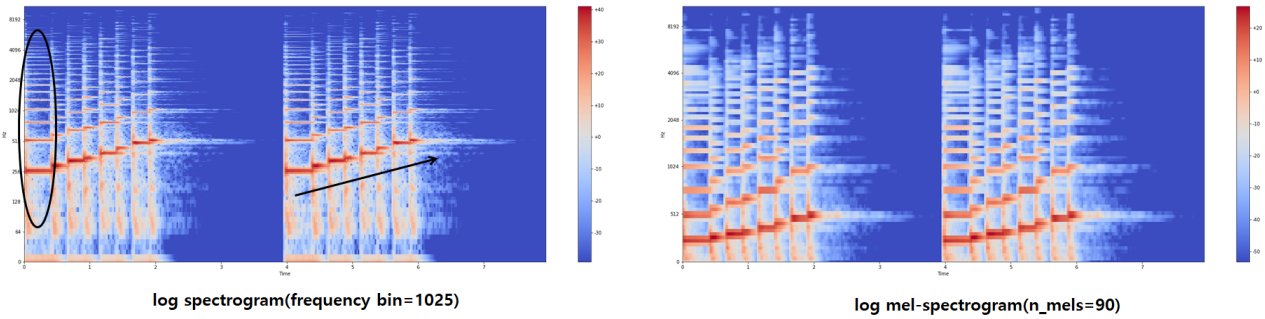


그림 18: log spectrogram vs log mel-spectrogram: '도레미파솔라시도'가 두번 나오는 sound[8]에 대한 spectrogram. 배음 구조(fundamental frequency의 정수배)와 주파수가 올라가는 구조가 잘 보인다. 오른쪽의 mel spectrogram은 차원을 줄이면서도 spectrogram의 특징을 잘 반영하고 있다. MFCC는 더 압축되는 방식이라, 이런 특징이 시각적으로 잘 보이지 않는다.

- librosa.feature.melspectrogram: waveform을 입력하면, magnitude spectrogram을 power(default=2)해서 mel filter bank를 곱한다.

```
# melspectrogram를 한번에 구하는 것과, mel filter bank를 거쳐서 구한 것의 결과가 동일하다.
scale, sr = librosa.load(wav_file) # sr=22050, (174943,)
mel_spectrogram = librosa.feature.melspectrogram(scale, sr=sr, n_fft=2048, hop_length=512,
    n_mels=10) # (10,342)

S_scale = librosa.stft(scale, n_fft=2048, hop_length=512) # (1025, 342)
filter_banks = librosa.filters.mel(n_fft=2048, sr=22050, n_mels=10) # mel matrix
mag_spectrogram = librosa.magphase(S_scale,power=2)[0] # (magnitude,phsase)[0]
mel_spectrogram2 = np.matmul(filter_banks,mag_spectrogram) # (10, 342)
```

[8] https://www.dropbox.com/s/3nmwun0s1dd25tw/scale.wav?dl=0

```
print(np.allclose(mel_spectrogram,mel_spectrogram2)) # True
```

output의 크기는 (n_mels, # of frames)=(D,T). melspectrogram을 conv1d 또는 conv2d에 넣을 수 있다. pytorch의 conv1d에 넣기 위해서는 batch data의 shape은 (N, D, T)로 되어야 하도, Tensorflow의 conv1d에 넣기 위해서는 transpose하여 (N, T, D)로 변형해야 한다(tacotron tensorflow구현에서도 transpose해서 사용).

- `librosa.feature.mfcc(y,n_mfcc=40)`: waveform을 mel-spectrogram으로 변환 후, mfcc를 만든다. 내부적으로는 `librosa.filters.mel`, `librosa.feature.melspectrogram`의 default parameter들이 사용된다.

```
# mfcc를 한번에 구하는 것과, melspectrogram을 거쳐서 구한 것의 결과가 동일하다.
n_mfcc = 20

mfcc = librosa.feature.mfcc(scale,n_mfcc=n_mfcc) # hop_length=512,n_fft=2048,
    n_mels(default=128) 크기로 생성후, 앞쪽
    return
mfcc_ = librosa.feature.mfcc(scale,n_mfcc=80,hop_length=512,n_fft=2048) # n_mels(default=128) 크
    기로 생성후, 앞쪽 80개 return

mel_spectrogram = librosa.feature.melspectrogram(scale, sr=sr) # default: n_fft=2048,
    hop_length=512, n_mels=128
log_mel_spectrogram = librosa.power_to_db(mel_spectrogram)
print('shape of log_mel_spectrogram: ', log_mel_spectrogram.shape)
mfcc2 = librosa.feature.mfcc(S=log_mel_spectrogram,n_mfcc=n_mfcc) # output shape, min(n_mels,
    n_mfcc)

print(mfcc.shape,mfcc2.shape)
print("2가지 mfcc 결과 비교: ", np.allclose(mfcc,mfcc2)) # True

mfcc3 = dct(log_mel_spectrogram, type=2, axis=0, norm='ortho')[1:n_mfcc, :]
print(mfcc3.shape)
print("직접 구한 결과와 비교: ", np.allclose(mfcc[1:],mfcc3)) # True
```

- librosa.feature.delta(mfcc, order=1, axis=-1): delta of mfcc, delta delta of mfcc를 구할 때 사용. 시간에 대한 미분이 필요하므로, axis=-1
- `librosa.cqt`: Constant Q-Transform[9]은 Fourier Transform의 변형이고, Morlet wavelet transform과도 관련있다. 음악을 분석하는데 적합한 것으로 알려져 있다. 결론적으로는 mel-spectrogram과 유사하다.
- `librosa.feature.chroma_stft`: Chromagram[10]은 power spectrogram을 octave에 상관없이 12(n_chroma) 음정에 mapping 시킨다.
- `librosa.feature.chroma_cqt`: Constant-Q transform 결과를 octave에 상관없이 12(n_chroma) 음정에 mapping 시킨다.
- `librosa.feature.spectral_centroid`: Each frame of a magnitude spectrogram is normalized and treated as a distribution over frequency bins, from which the mean (centroid) is extracted per frame. 수식으로 표

[9] https://en.wikipedia.org/wiki/Constant-Q_transform

[10] https://en.wikipedia.org/wiki/Chroma_feature

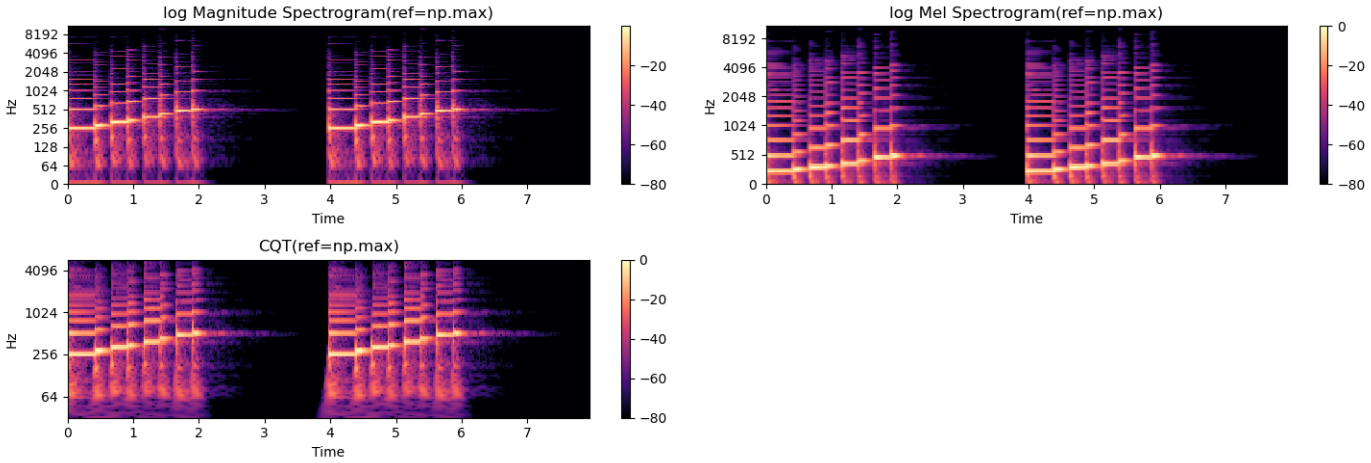


그림 19: Constant Q Transform. 좌측: log spectrogram. 가운데: log-mel-spectrogram: 오른쪽:log-CQT

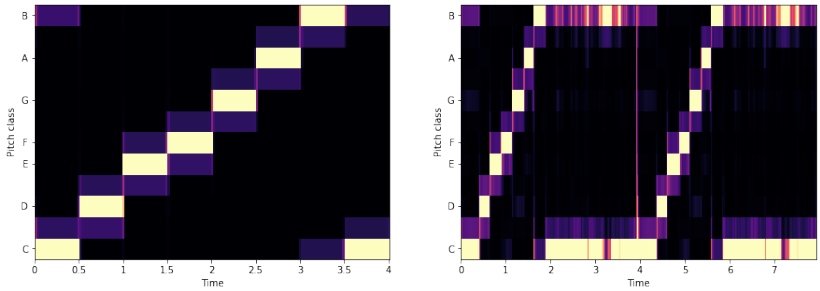


그림 20: Chromagram: '도레미파솔라시도'의 chromagram. 왼쪽은 단순 sine파 합성이고, 오른쪽을 파아노로 (2번 반복) 연주된 것이다.

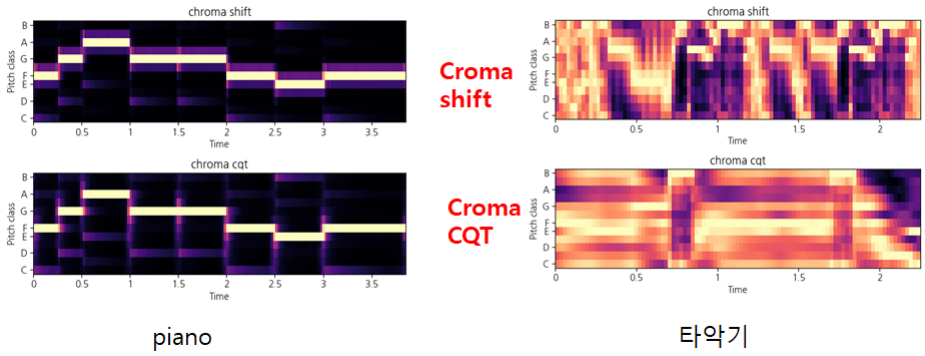


그림 21: Chromagram of Spectrogram vs Chromagram of CQT: Chomagram은 power spectrogram을 n_chroma에 대응시킨 것 vs Constant Q-Transform을 Chromagram으로 변환한 것.

현해보자. i 번째 frame의 (주파수) j 번째 magnitude spectrogram을 $S(i,j)$ 라 하자. 그리고 j 번째 주파수를 $f(j)$ 라 하면, spectral centroid $C(i)$ 는 다음과 같이 정의된다.

$$C(i) = \frac{\sum_j S(i,j)f(j)}{\sum_j S(i,j)} = \sum_j \frac{S(i,j)}{A_i} f(j), \quad A_i = \sum_j S(i,j)$$

여기서 $\{f(j)\}$ 는 구간 $[0, \frac{\text{sr}}{2}]$ 를 $\frac{\text{n\_fft}}{2}$ 등분하면 $\frac{\text{n\_fft}}{2}+1$ 개의 frequency가 구해진다. spectral centroid는 소리의 brightness를 측정한다. `librosa.fft_frequencies(sr,n_fft=2048)`

- `librosa.feature.spectral_bandwidth`: 각 frame i 에 대하여,

$$\left[\sum_j \frac{S(i,j)}{A_i} \left|f(j) - C(i)\right|^p\right]^{\frac{1}{p}}, \quad A_i = \sum_j S(i,j)$$

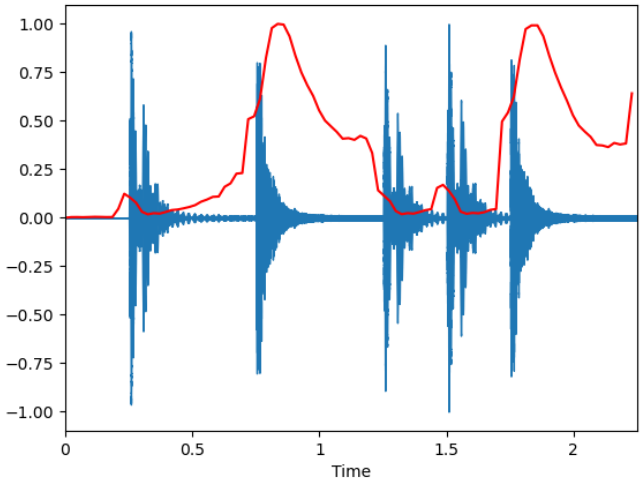


그림 22: spectral centroid를 nomalize하여, waveform과 같이 그린 그림.

librosa의 default $p = 2$이고, A_i로 나누어주는 부분은 빠질 수도 있다(default norm=True).

- `librosa.feature.spectral_rolloff`: 각 frame별 roll-off frequency를 구해준다.

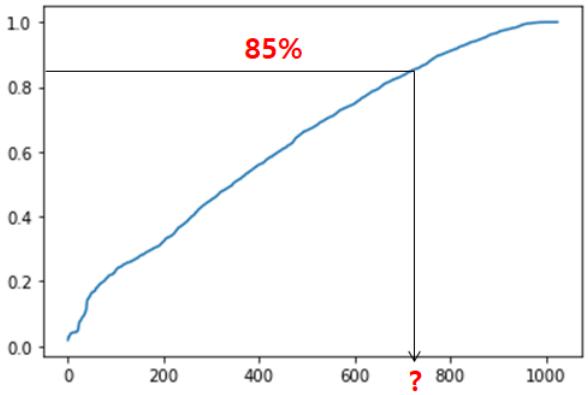


그림 23: 각각의 frame이 대하여 magnitude(not power) spectrogram의 누적합이 85%를 넘어가는 frequency 구한다. 이때, frequency의 x 축 값은 spectral centroid에서의 $\{f(j)\}$이다.

```
import librosa
import librosa.display
audio_filename = librosa.util.example_audio_file() # 61.4588초 길이
y, sr = librosa.load(audio_filename)
rolloff = librosa.feature.spectral_rolloff(y+0.01, sr=sr, roll_percent=0.85)[0] # maximum
    frequencies with roll_percent=0.85
#rolloff = librosa.feature.spectral_rolloff(y+0.01, sr=sr, roll_percent=0.1)[0] # minimum
    frequencies with roll_percent=0.1
librosa.display.waveplot(y, sr=sr, alpha=0.4)
t = librosa.frames_to_time(range(len(rolloff)))
normed_rolloff = sklearn.preprocessing.minmax_scale(rolloff, axis=0)
plt.plot(t, normed_rolloff, color='r')
```

- `librosa.effects.hpss`(librosa.effects.harmonic + librosa.effects.percussive): harmonic percussive source

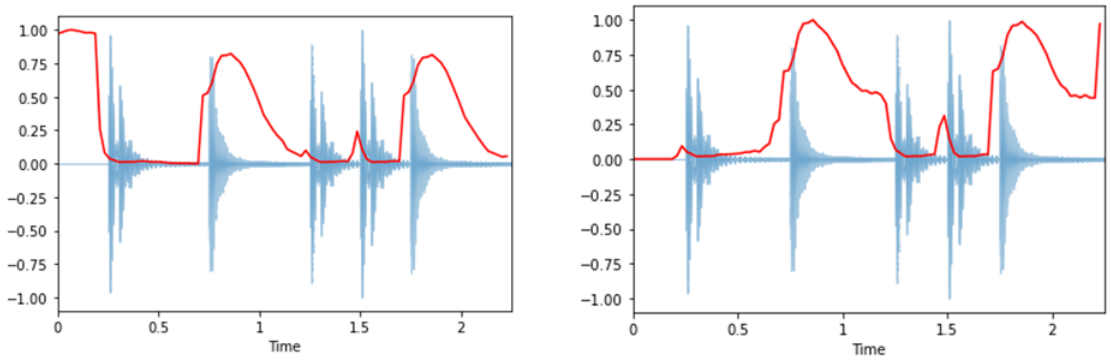


그림 24: 시작 부분의 silence 때문에, rolloff값이 왜곡된다(왼쪽 그림). 그래서 rolloff를 구할 때, 0.01을 더해준다(오른쪽 그림).

separation. Decompose an audio time series into harmonic and percussive components[11]. percussive components는 음악 장르 분류에 가장 중요한 feature이다.

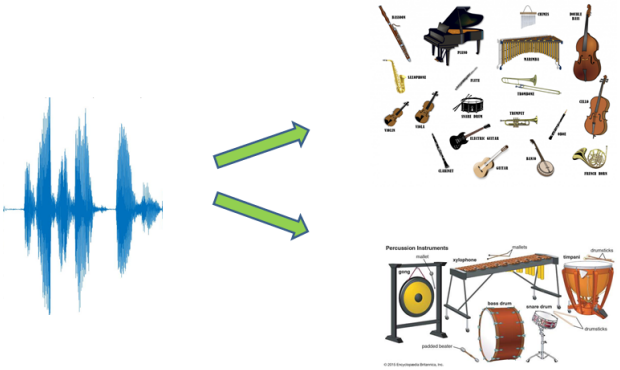


그림 25: harmonic percussive source separation

```
audio_filename = ...
y, sr = librosa.load(audio_filename)
harmonic,percussive = librosa.effects.hpss(y)

print(harmonic.shape, percussive.shape) # y와 같은 길이
print('mean: ',np.mean(y), np.mean(harmonic), np.mean(percussive))

plt.subplot(3,1,1); librosa.display.waveplot(y, sr=sr, alpha=0.4); plt.ylim(-1,1)
plt.subplot(3,1,2); librosa.display.waveplot(harmonic, sr=sr, alpha=0.4); plt.ylim(-1,1)

plt.subplot(3,1,3); librosa.display.waveplot(percussive, sr=sr, alpha=0.4); plt.ylim(-1,1)
```

- `librosa.beat.beat_track`: tempo와 beats를 return한다. 예: tempo=129.19, beats는 길이 129인 정수 array인데, beat가 발생한 frame 위치이다. tempo값은 `librosa.beat.tempo`와 동일하다.
- `librosa.beat.tempo`: Estimate the tempo (beats per minute).

```
import librosa
audio_filename = librosa.util.example_audio_file() # 61.4588초 길이
y, sr = librosa.load(audio_filename)
```

[11] http://mir.ilsp.gr/harmonic_percussive_separation.html

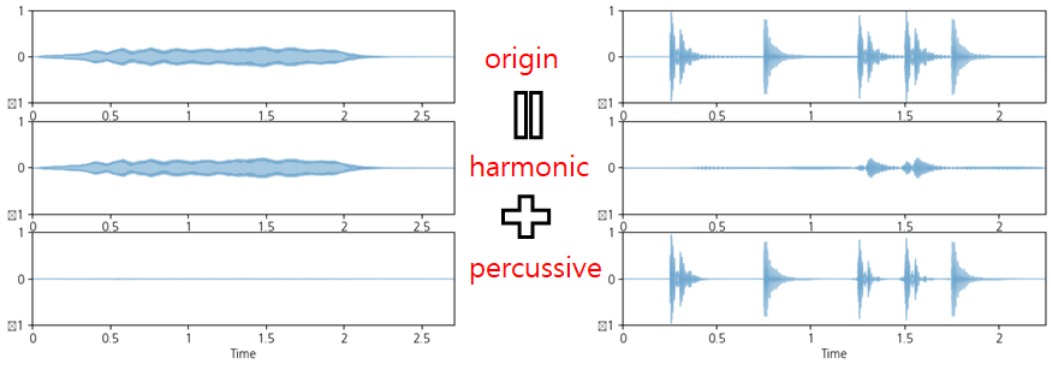


그림 26: 왼쪽은 violine 소리를 분해한 결과인데, harmonic part가 대부분이다. 반면, 오른쪽은 타악기 소리를 분해한 것인데, harmonic part가 약하다.

```
tempo, beats = librosa.beat.beat_track(y=y, sr=sr,hop_length=512) # tempo: 129.19921875,
    beats.shape: (129,)
# beats now contains the beat *frame positions*
# convert to timestamps like this:
beat_times = librosa.frames_to_time(beats, sr=sr) # beat가 발생한 frame들을 순서대로 구해준다.
```

- `librosa.output.write_wav('sound.wav', data, sr)`: array를 wav 파일로 저장. librosa 0.8에서는 이 API가 제공되지 않는다.

```
import soundfile
sr = 22050
data = ...

# Write out audio as 24bit PCM WAV
soundfile.write('stereo_file.wav', data, sr, subtype='PCM_24')

# Write out audio as 24bit Flac
soundfile.write('stereo_file.flac', data, sr, format='flac', subtype='PCM_24')

# Write out audio as 16bit OGG
soundfile.write('stereo_file.ogg', data, sr, format='ogg', subtype='vorbis')
```

- 기타 API

API	설명
librosa.onset.onset_detect	

표 3: lirosa 기타 API

0.2 Music Genre Classification

- GTZAN[12] Dataset: [blues,classical,country,disco,hiphop,jazz,metal,pop,reggae,rock] 10개 장르에 30초짜리 음악이 100개씩 있다.
  - wav file이 10개 디렉토리로 나누어져 있다.
  - mel spectrogram이 이미지 형태(432 x 288)로 저장되어 있다.
  - `features_3_sec.csv`(30초를 10개로 나누어 3초 sound에 대한 feature 추출). `features_30_sec.csv`(각 파일별 feature 추출)
  - (filename,length,chroma_stft_mean,chroma_stft_var,rms_mean,rms_var,spectral_centroid_mean, spectral_centroid_var,...,mfcc19_mean,mfcc19_var,mfcc20_mean,mfcc20_var,label) 모두 60개 column으로 되어 있다.
- PCA, tSNE(Stochastic Neighbor Embedding), UMAP(Uniform Manifold Approximation and Projection)

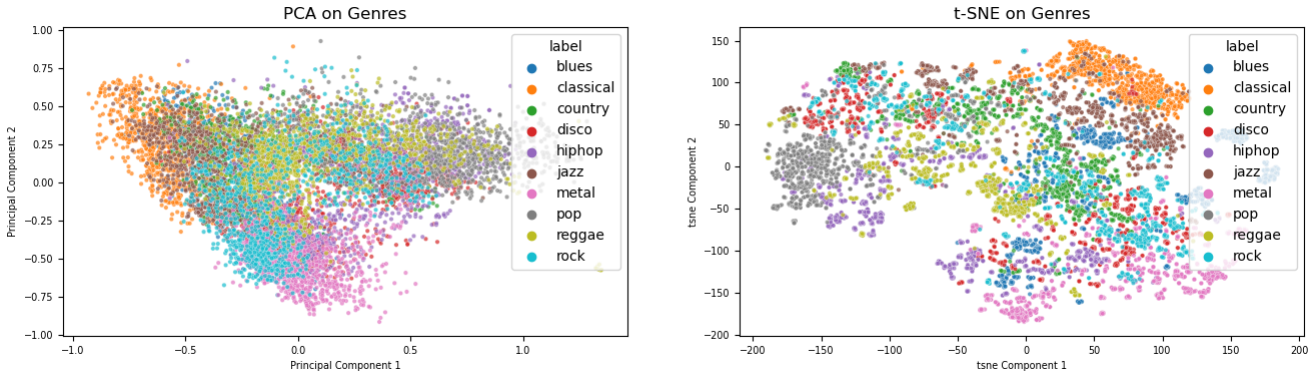


그림 27: PCA vs tSNE: tSNE는 perplexity=20,n_iter=10000으로 했을 때 결과이다. 값에 따라 결과가 달라진다.

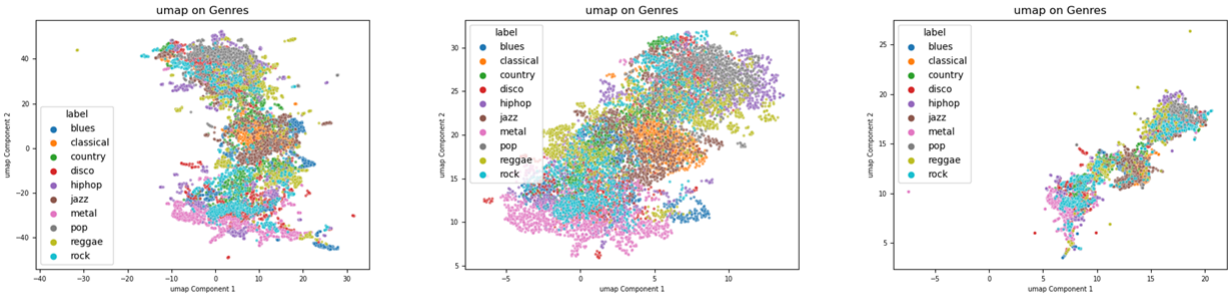


그림 28: umap은 spread, min_dist에 따라 결과가 달라진다. spread는 값의 scale을 결정하고, min_dist는 spread보다 작은 값으로 설정해야 되고, 작을수록 뭉치게 된다(n_epochs=5000).

```
audio_filename = ...
y, sr = librosa.load(audio_filename)

melspectrogram = librosa.feature.melspectrogram(y,n_mels=90)
print(melspectrogram.shape) # (90, 342)
```

[12] https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification

```
librosa.display.specshow(librosa.power_to_db(melspectrogram,ref=np.max),x_axis='time',y_axis='mel')
plt.axis('off') # 이게 없으면 좀 더 큰 사이즈로 저장.
plt.savefig('mel_spec.png',bbox_inches='tight', pad_inches=0) # (w,h) = (496 x 369) 크기로 저장
```

Listing 1: librosa를 통해 저장하면, mel-spectrogram의 크기보다 더 큰 사이즈의 이미지로 저장된다.

MLP(mfcc)	train acc = 0.9986 , test acc = 0.6335
MLP+dropout+L2(mfcc)	train acc = 0.82 , test acc = 0.62 droprate을 올리면, train acc는 더 내려가지만 test acc는 향상 안됨.
CNN(mfcc)	mfcc를 2d array를 이미지로 취급 lr = 0.0001, train acc = 0.91, test acc = 0.73 lr = 0.001, train acc = 0.975, test acc = 0.762
RNN(mfcc)	lr=0.0001, train acc = 0.87, test acc = 0.70 lr=0.001, train acc = 1.0, test acc = 0.87
xgboost(mfcc → frame 평균)	test acc = 0.766
xgboost(mfcc → frame 평균, 분산)	test acc = 0.81
xgboost(`features_3_sec.csv`)	test acc = 0.90224
LightGBM(`features_3_sec.csv`)	test acc = 0.9239
MLP+dropout(`features_3_sec.csv`)	300 epoch. lr=0.0001, train acc = 0.95 , test acc = 0.91 lr=0.001, train acc = 0.957 , test acc = 0.9279
xgboost(feature 직접 생성)	test acc = 0.96663
MLP+dropout(feature 직접 생성)	300 epoch. lr=0.0001, train acc = 0.933 , test acc = 0.9435 lr=0.001, train acc = 0.97 , test acc = 0.9627
LightGBM(feature 직접 생성)	test acc = 0.97931

표 4: 모델별 Accuracy: 1. 3초 data로 mfcc(n_mfcc=13) 생성. 100 epoch씩 train.
2. 만들어진 feature를 download 받은 것보다 직접 만들었을 때에 가장 좋은 결과가 나온다. 직접 만들었을 때에 mfcc의 variance는 추가하지 않았다.
3. mfcc variance를 추가했을 때 xgboost test acc는 0.96663 → 0.9576로 조금 낮아졌다.
4. n_mfcc=40으로 했을 때는 n_mfcc=13에 비해 더 좋아지는 것은 없다.

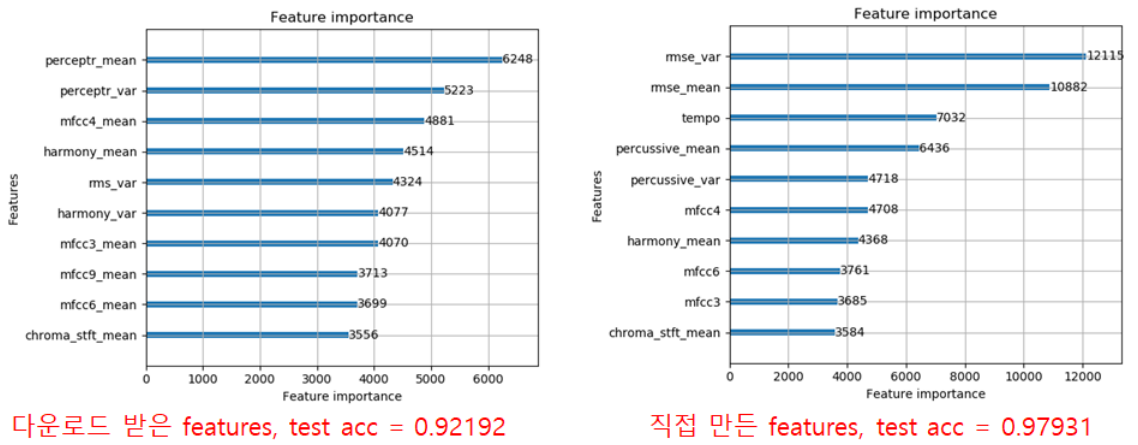


그림 29: LightGBM에서의 Feature Importance 비교.