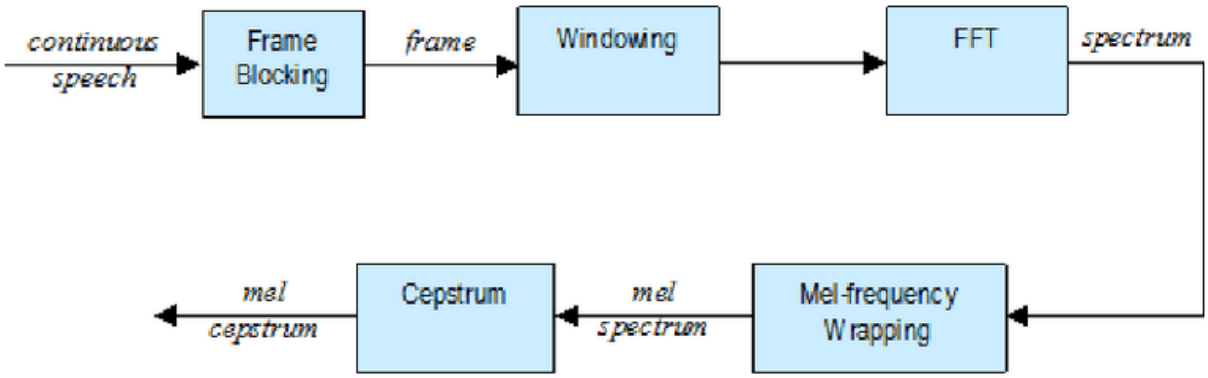
Music Feature Extraction in Python

Different type of audio features and how to extract them.

Sanket Doshi Dec 30, 2018 · 5 min read

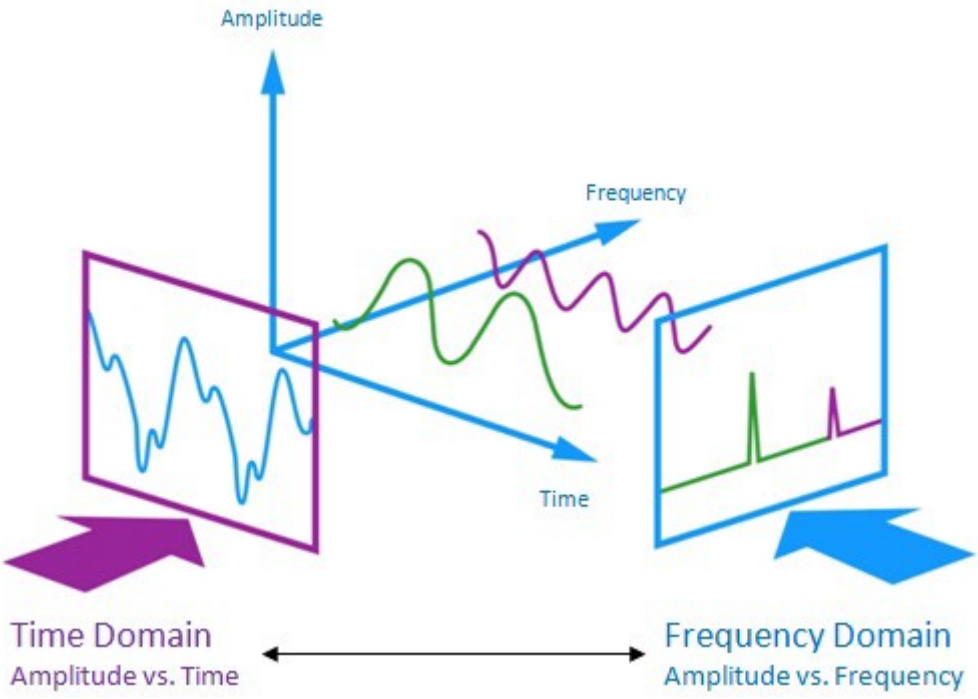


MFCC feature extraction

Extraction of features is a very important part in analyzing and finding relations between different things. The data provided of audio cannot be understood by the models directly to convert them into an understandable format feature extraction is used. It is a process that explains most of the data but in an understandable way. Feature extraction is required for classification, prediction and recommendation algorithms.

In this blog, we will extract features of music files that will help us to classify music files into different genres or to recommend music based on your favorites. We will learn different techniques used for extracting features of music.

The audio signal is a three-dimensional signal in which three axes represent time, amplitude and frequency.



Audio signal

Packages to be used

We'll be using librosa for analyzing and extracting features of an audio signal. For playing audio we will use pyAudio so that we can play music on jupyter directly.

Loading an audio

```
import librosa
audio_path = 'audio-path'
x , sr = librosa.load(audio_path)
print(type(x), type(sr))
```

`.load` loads an audio file and decodes it into a 1-dimensional array which is a time series `x` , and `sr` is a sampling rate of `x` . Default `sr` is 22kHz. We can override the `sr` by

```
librosa.load(audio_path, sr=44100)
```

We can disable sampling by:

```
librosa.load(audio_path, sr=none)
```

Playing an audio

```
import IPython.display as ipd
ipd.Audio(audio_path)
```

`IPython.display` allow us to play audio on jupyter notebook directly. It has a very simple interface with some basic buttons.

```
#display waveform
%matplotlib inline
import matplotlib.pyplot as plt
import librosa.display
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

`librosa.display` is used to display the audio files in different formats such as wave plot, spectrogram, or colormap. Waveplots let us know the loudness of the audio at a given time. Spectogram shows different frequencies playing at a particular time along with it's amplitude. Amplitude and frequency are important parameters of the sound and are unique for each audio. `librosa.display.waveplot` is used to plot waveform of amplitude vs time where the first axis is an amplitude and second axis is time.

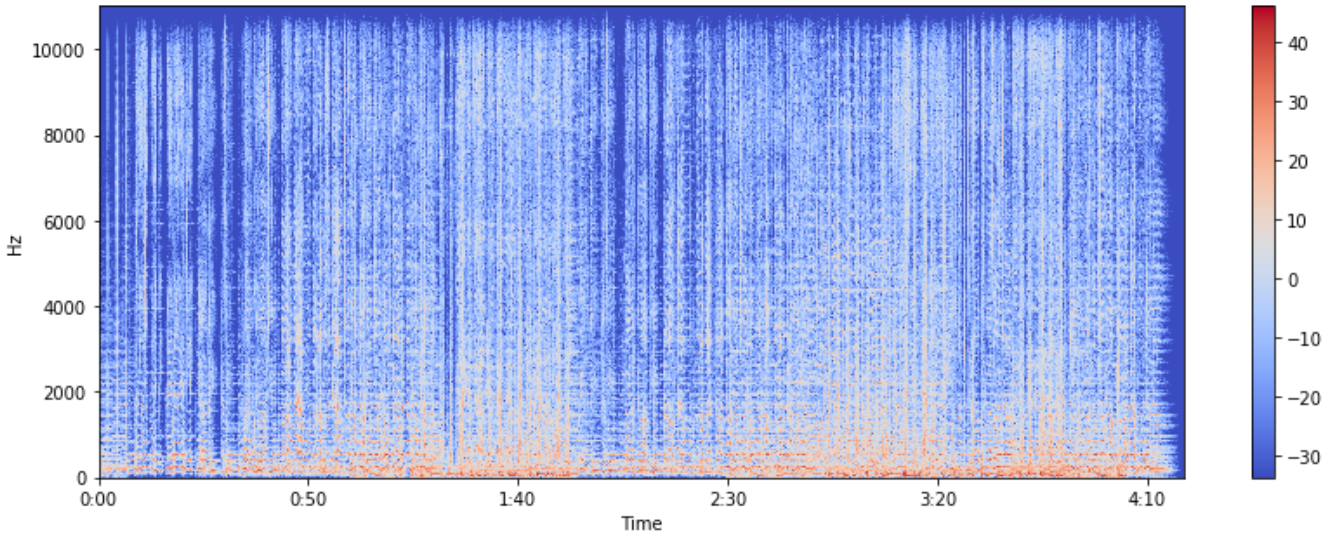
Spectogram

A spectrogram is a visual representation of the spectrum of frequencies of sound or other signals as they vary with time. It's a representation of frequencies changing with respect to time for given music signals.

```
#display Spectrogram
X = librosa.stft(x)
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')
#If to pring log of frequencies
#librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log')
plt.colorbar()
```

`.stft` converts data into short term Fourier transform. STFT converts signal such that we can know the amplitude of given frequency at a given time. Using STFT we can determine the amplitude of various frequencies playing at a given time of an audio signal. `.specshow` is used to display spectogram.

The output would be like:



spectrum

Feature Extraction

Zero Crossing Rate

The zero crossing rate is the rate of sign-changes along a signal, i.e., the rate at which the signal changes from positive to negative or back. This feature has been used heavily in both speech recognition and music information retrieval. It usually has higher values for highly percussive sounds like those in metal and rock.

```
x, sr = librosa.load(audio_path)
#Plot the signal:
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

Zooming in:

Here we will zoom or print spectrum for 100 array columns only.

```
# Zooming in
n0 = 9000
n1 = 9100
plt.figure(figsize=(14, 5))
plt.plot(x[n0:n1])
plt.grid()
```

The plot looks like:

As we can see there are three zero crossings in the given graph.

We can also calculate zero crossings using a given code:

```
zero_crossings = librosa.zero_crossings(x[n0:n1], pad=False)
print(sum(zero_crossings))
```

Spectral Centroid

It indicates where the ”centre of mass” for a sound is located and is calculated as the weighted mean of the frequencies present in the sound. If the frequencies in music are same throughout then spectral centroid would be around a centre and if there are high frequencies at the end of sound then the centroid would be towards its end.

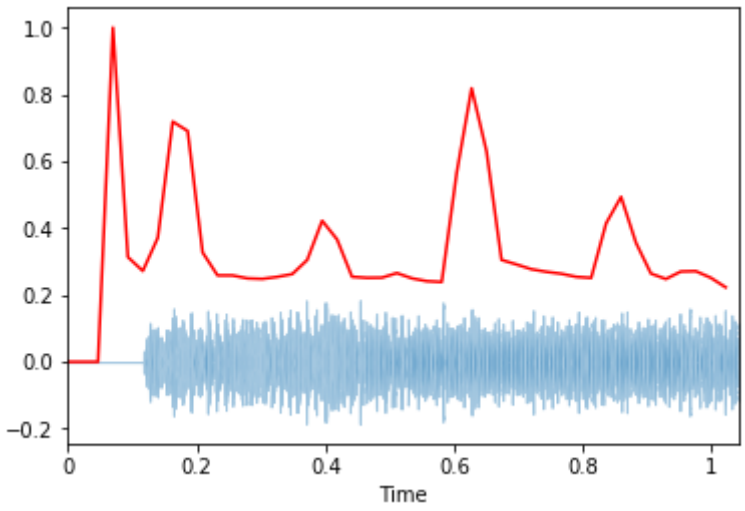
```
#spectral centroid -- centre of mass -- weighted mean of the
frequencies present in the sound
import sklearn
spectral_centroids = librosa.feature.spectral_centroid(x, sr=sr)[0]
spectral_centroids.shape
```

```
# Computing the time variable for visualization
frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)
# Normalising the spectral centroid for visualisation
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
#Plotting the Spectral Centroid along the waveform
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='r')
```

`.spectral_centroid` is used to calculate the spectral centroid for each frame. So it'll return an array with columns equal to a number of frames present in your sample.

`.frames_to_time` converts frame to time. time[i] == frame[i].

We're normalizing so that we can visualize data easily.



spectral centroid

Similar to the zero crossing rate, there is a spurious rise in spectral centroid at the beginning of the signal. That is because the silence at the beginning has such small amplitude that high-frequency components have a chance to dominate.

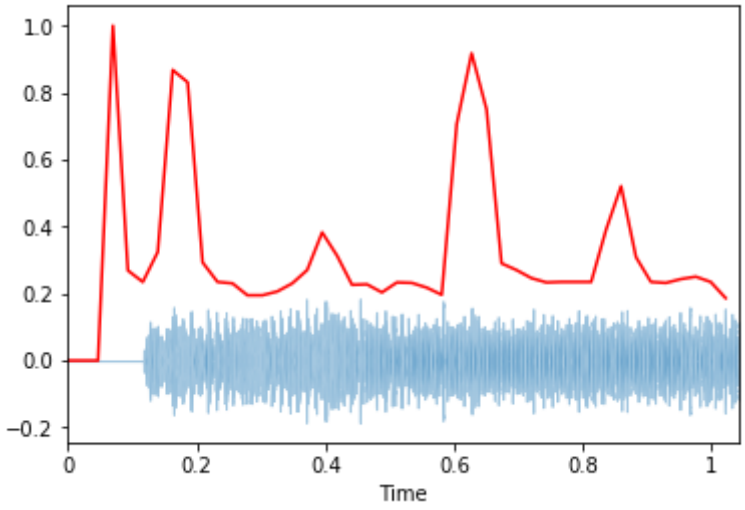
Spectral Rolloff

Spectral rolloff is the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies.

It also gives results for each frame.

```
spectral_rolloff = librosa.feature.spectral_rolloff(x, sr=sr)[0]
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_rolloff), color='r')
```

`.spectral_rolloff` is used to calculate rolloff for a given frame.



Spectral Rolloff

MFCC — Mel-Frequency Cepstral Coefficients

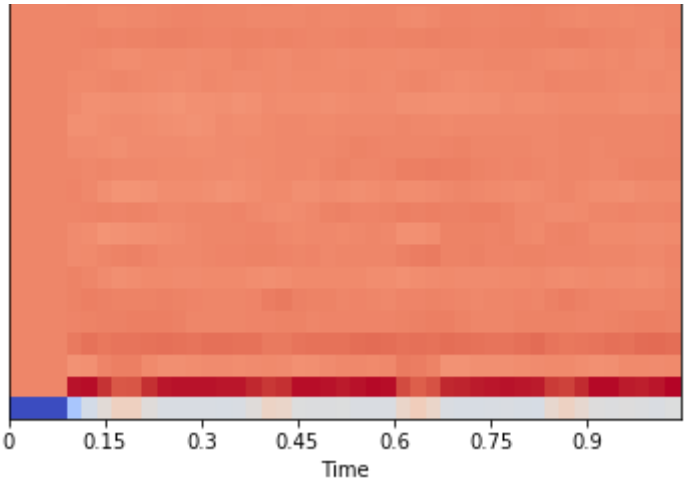
This feature is one of the most important method to extract a feature of an audio signal and is used majorly whenever working on audio signals. The mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope.

```
mfccs = librosa.feature.mfcc(x, sr=sr)
print(mfccs.shape)

#Displaying  the MFCCs:
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

`.mfcc` is used to calculate mfccs of a signal.

By printing the shape of mfccs you get how many mfccs are calculated on how many frames. The first value represents the number of mfccs calculated and another value represents a number of frames available.



MFCC

Now, we have extracted the features of music signals. We can use this feature extracted in various use cases such as classification into different genres. We'll implement that in our next blog.