Open in app

Justin Mitchell

18 Followers · About Follow

Generating Audio Features with Librosa

Justin Mitchell Jul 15, 2019 · 3 min read

When beginning a machine learning project that works with audio data or other forms of time dependent signals, it can be difficult to know where to start. Working with raw audio signals is usually not the best approach. Raw audio can be visualized as a waveplot, an example of which is shown below.

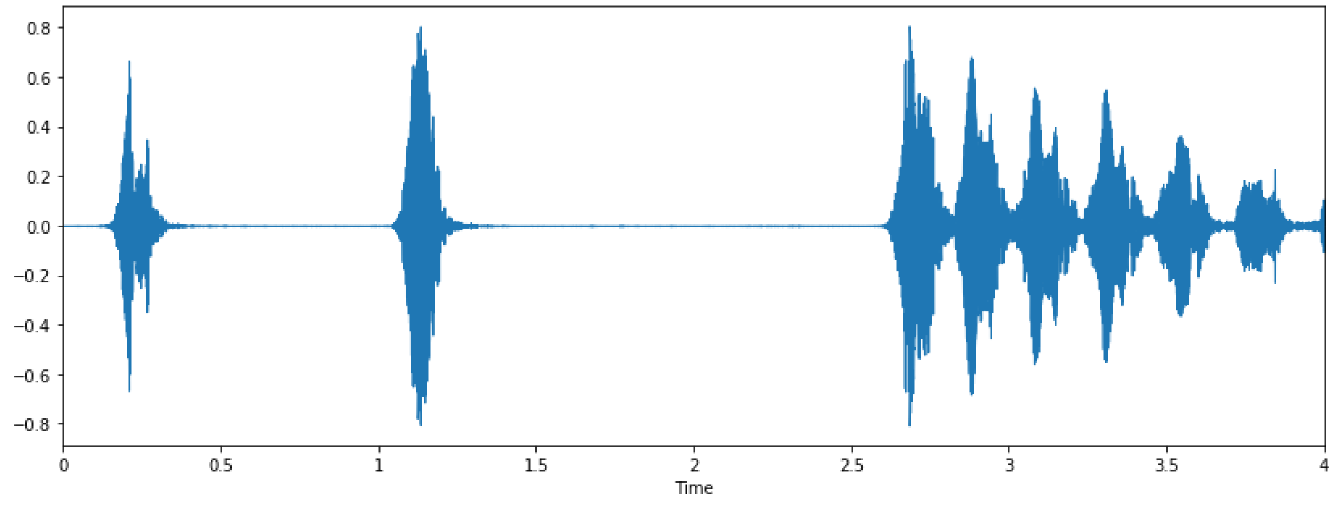

Fig 1. Waveplot of a barking dog

A waveplot plots a signal's amplitude envelope against time. It can be useful to visualize what a signal looks like, but is typically not useful to machine learning models in making predictions. In order to make a signal useful, it is necessary to extract less obvious features. Commonly extracted features can generally be divided into two categories: temporal, which deals with time dependent features, and spectral, which deals with frequency dependent features.

Zero-crossing-rate is an example of a temporal feature whereas mel-frequency cepstral coefficients (MFCCs) and spectral centroid are examples of spectral features. This post will not go into too much detail on the definitions of features and will instead focus on how to extract them.

We will begin by importing the necessary libraries.

```
import librosa # for working with audio in python
import librosa.display # for waveplots, spectograms, etc
import soundfile as sf # for accessing file information
import IPython.display as ipd # for playing files within python

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

I'll be working with the UrbanSound8k dataset, which is a very useful dataset for getting started with audio or, more generally, signal focused projects.

Now that we have the required libraries, let's examine how to extract some basic info about the file we will be working with.

```
info = sf.info('UrbanSound8K/audio/fold1/101415-3-0-2.wav')
print(info)

UrbanSound8K/audio/fold1/101415-3-0-2.wav
samplerate: 48000 Hz
channels: 1
duration: 4.000 s
format: WAV (Microsoft) [WAV]
subtype: Signed 16 bit PCM [PCM_16]
```

The next step is to actually load an audio file using librosa. This can be accomplished with the code below.

```
x, sample_rate = librosa.load('UrbanSound8K/audio/fold1/101415-3-0-
2.wav', sr=None)
```

The sampling rate is the number of times the file is sampled per second. It can be set to None to preserve the original file's sample rate or it can be resampled at a specified rate. The sampling rate must be twice the frequency of the highest frequency that is desired to be captured according to the Nyquist-Shannon sampling theorem.

```
print(f'sample rate * duration = 4(s)*48000(1/s) =
{info.samplerate*info.duration}')

sample rate * duration = 4(s)*48000(1/s) = 192000.0
```

If you're working in a jupyter notebook, you can use iPython.display to listen to the audio file you've loaded, as seen below.

```
ipd.Audio(data=x, rate=sample_rate) # load a local WAV file
```

Fig 2. iPython audio display. Useful for EDA.

To create a waveplot like the one seen in fig 1, use the code below.

```
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sample_rate)
```

The code below can be used to generate some common features after loading an audio file.

```
mfccs = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc=40)

chroma = librosa.feature.chroma_stft(S=stft, sr=sample_rate)
```

```
mel = librosa.feature.melspectrogram(X, sr=sample_rate)

contrast = librosa.feature.spectral_contrast(S=stft, sr=sample_rate)

tonnetz =
librosa.feature.tonnetz(y=librosa.effects.harmonic(X),sr=sample_rate
)
```

If you'd like to become more confident in your audio based machine learning projects, you should strive to develop an understanding of the discrete short time fourier transform (STFT). The STFT can be used to obtain the frequencies that are present in discrete time windows of an audio file. The reason this is more useful than just a fourier transform of the entire audio file is that it allows one to see at what point in time certain frequencies occur and, by extension, allows the discernment of where in time frequencies change.

Python Signal Processing Audio Features Librosa