**Porównywanie plików audio w domenie częstotliwości**

Do porównania plików audio w domenie częstotliwości możemy wykorzystać:

* LibROSA - pakiet Pythona do analizy muzyki i dźwięku. Zapewnia elementy składowe niezbędne do tworzenia systemów wyszukiwania informacji muzycznych.
* proces o nazwie FFT ( szybka transformata Fouriera ) do wydobywania danych częstotliwości z plików audio

Ładowanie danych

W celu przeniesienie danych audio do projektu wykorzystujemy pakiet LibROSA.

dzwiek = AudioAnalyzer ( „audio.wav” , input\_sr = 44100 , fft\_size = 44100 )

* input\_sr - częstotliwość próbkowania i jest ustawione na 44100 dla każdego pliku.
* fft\_size FFT - sposób na pobieranie danych w dziedzinie czasu i stworzenie określonej liczby pojemników , w których do informacji częstotliwości miejsce. Dopasowujemy rozmiar FFT do częstotliwości próbkowania tak, aby każdy z tych przedziałów miał dokładnie 1 Hz.

|  |
| --- |
|  |
|  |  |
|  | class AudioAnalyzer:  def \_\_init\_\_(self, filename, fft\_size=256, input\_sr=41000): |
|  |
|  |
|  | y, sr = librosa.load(filename, sr=input\_sr) |
|  | self.y = y |
|  | self.sr = sr |
|  | self.fft\_size = fft\_size |
|  |
|  |

* nazwa pliku: ścieżka względna do pliku audio
* fft\_size: liczba przedziałów częstotliwości dla analizy fft. Domyślnie 256
* input\_sr: częstotliwość próbkowania wejściowego pliku audio. Domyślnie 41000

Aby uzyskać najlepsze wyniki analityczne, musimy znać częstotliwość próbkowania każdego z naszych plików, a najlepiej, jeśli te częstotliwości próbkowania są zgodne między plikami. Tutaj wszystkie mają 44,1 kHz (standardowa częstotliwość próbkowania).

klasa AudioAnalyzer wprowadza dźwięk do librosa i przekształca go w domenę częstotliwości.

Po załadowaniu dźwięku, przepuszczamy go przez przetwarzanie FFT, a następnie uśredniamy amplitudy każdego przedziału częstotliwości, aby uzyskać pojedyncze widmo dla każdego pliku.

dzwiek . plot\_spectrum ( min\_freq = 20 , max\_freq = 1000 , title = "audio" )

Określamy minimalną i maksymalną częstotliwość, abyśmy mogli wyzerować określony zakres, w którym będziemy mogli zobaczyć wszystko nieco wyraźniej.

|  |  |
| --- | --- |
|  | def plot\_spectrum(self, min\_freq=0, max\_freq=None, fill=False,  title="Spectrogram - Average Frequency"): |
|  | max\_freq = max\_freq or 20000. |
|  | window = self.df.loc[(self.df.bins \* 1000. >= min\_freq) &  (self.df.bins \* 1000. <= max\_freq)].copy() |
|  | window['scaled\_amplitude'] = np.interp(window.average\_amplitude,  (0., max(window.average\_amplitude)), (0., 1.)) |
|  | window.plot(x='bins', y='scaled\_amplitude', figsize=(16,4)) |
|  | if fill: |
|  | plt.fill\_between(self.df.bins, self.df.average\_amplitude) |
|  |  |
|  | legend = plt.legend() |
|  | legend.remove() |
|  | plt.xlabel("Frequency (kHz)", fontsize=20) |
|  | plt.ylabel("Amplitude (scaled)", fontsize=20) |
|  | plt.title(title, fontsize=26) |
|  |  |

Wykreśla pojedynczy spektrogram uśrednionych częstotliwości dla wszystkich przedziałów FFT. Używa wygenerowanej ramki danych jako źródła.

Porównywanie dźwięku

Klasa SpectrumCompare obsługuję porównanie dwóch plików audio.

porownanie = SpectrumCompare (dzwiek , dzwiek2)

def \_\_init\_\_(self, original, modified):

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  | self.original\_df = original.df |
|  | self.modified\_df = modified.df |
|  |  |
|  | self.dfs = [self.original\_df, self.modified\_df] |
|  |  |
|  | self.get\_max\_average(self) |
|  | self.scale\_amplitudes(self) |
|  | self.get\_ratio\_df(self) |

Porównuje informacje o częstotliwości i amplitudzie dwóch instancji klasy AudioAnalyzer.

Możemy to teraz przedstawić graficznie aby następnie obliczyć różnicę między dwoma wykresami .

Wykreślamy spektrogramy z dwóch plików jeden na drugim.

porownanie.plot\_spectrum\_group(

frange=(20,1000),

ratio=True,

threshold=True,

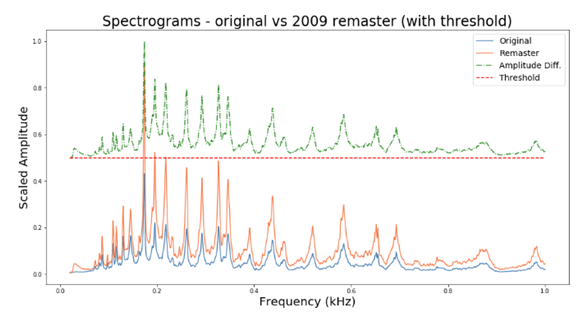
title="Spectrograms - original vs 2009 remaster (with threshold)",

legend=("Original", "Remaster", "Amplitude Diff.", "Threshold")

)

def plot\_spectrum\_group(self, spec\_1=True, spec\_2=True, title="Comparison of Frequency Amplitudes", xlabel="Frequency (kHz)", ylabel="Scaled Amplitude", frange=None, ratio=False, legend=["Spectrum 1", "Spectrum 2", "Signal Diff.", "Threshold"], threshold=False):

|  |  |
| --- | --- |
|  |  |
|  |  | dfs = self.dfs.copy()  fig = plt.figure(figsize=(16,8)) |
|  |  |  |
|  |  | if frange: |
|  |  | for i, df in enumerate(dfs): |
|  |  | dfs[i] = df.loc[(df.bins \* 1000. >= frange[0]) &  (df.bins \* 1000. <= frange[1])] |
|  |  | else: |
|  |  | dfs = self.dfs |
|  |  |
|  |  |
|  |  | if spec\_1: plt.plot(dfs[0].bins,  dfs[0].scaled\_amplitude, label=legend[0], c="steelblue") |
|  |  | if spec\_2: plt.plot(dfs[1].bins,  dfs[1].scaled\_amplitude, label=legend[1], c="coral") |
|  |  |  |
|  |  | if ratio: |
|  |  | if frange: |
|  |  | ratio\_df = self.ratio\_df[(self.ratio\_df.bins \* 1000. >=  frange[0]) & (self.ratio\_df.bins \* 1000. <= frange[1])] |
|  |  | plt.plot(ratio\_df['bins'], ratio\_df['scaled\_amplitude'],  label=legend[2], c="forestgreen", linestyle="-.") |
|  |  | else: |
|  |  | plt.plot(self.ratio\_df['bins'],  self.ratio\_df['scaled\_amplitude'],  label=legend[2], c="forestgreen") |
|  |  |  |
|  |  | if threshold and frange: |
|  |  | plt.plot([frange[0] / 1000, frange[1] / 1000], [0.5, 0.5],  linestyle="--", label=legend[3], c="r") |
|  |  |  |
|  |  | plt.title(title, fontsize=24) |
|  |  | plt.xlabel(xlabel, fontsize=20) |
|  |  | plt.ylabel(ylabel, fontsize=20) |
|  |  | plt.legend(fontsize=14) |

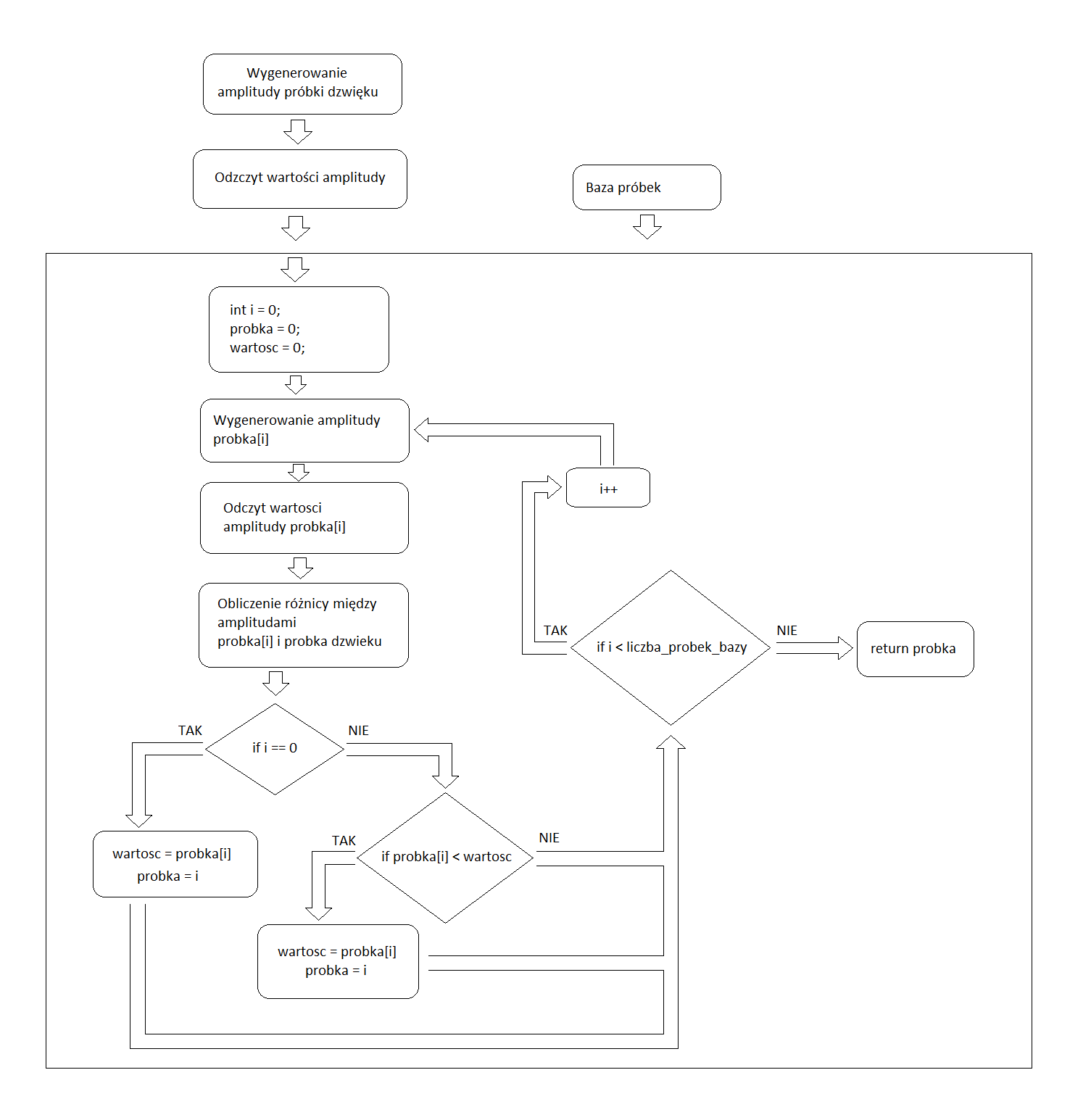
Przedstawiony poniżej wykres jest to spektogram który możemy wyświetlić za pomocą funkcji plot\_spectrum\_group aby zobaczyć różnicę między plikami audio

To co nas najbardziej interesuje jest to zielona linia ukazująca różnicę w skalowanych amplitudach oraz czerwona linia która jest progiem. Linia pomarańczowa i niebieska przedstawiają amplitudy próbek dźwięku.

Porównywanie różnic między dźwiękami

Najprostszym sposobem aby przyrównać wybrany dźwięk do bazy próbek jest przyrównanie amplitudy z dźwięku do amplitudy poszczególnej próbki. Para w której różnice amplitudowe będą najmniejsze będzie dopasowaniem. Możemy również w tym miejscu ustalić wyjątek – maksymalną wartość jaką wynosić może ta różnica.

Do odczytu wartości z amplitudy możemy posłużyć się przekazaniem ich bezpośrednio do zmiennej – wszystkie wartość które zostają wyrysowane na osi Y przekazywane są do np. utworzonej tablicy.



Graficzne porównywanie plików audio oraz działających algorytmów w tym zakresie:

<https://towardsdatascience.com/a-data-scientists-approach-to-visual-audio-comparison-fa15a5d3dcef>

Gotowy kod służący do graficznego porównywanie plików audio:

https://github.com/QED0711/audio\_analyzer

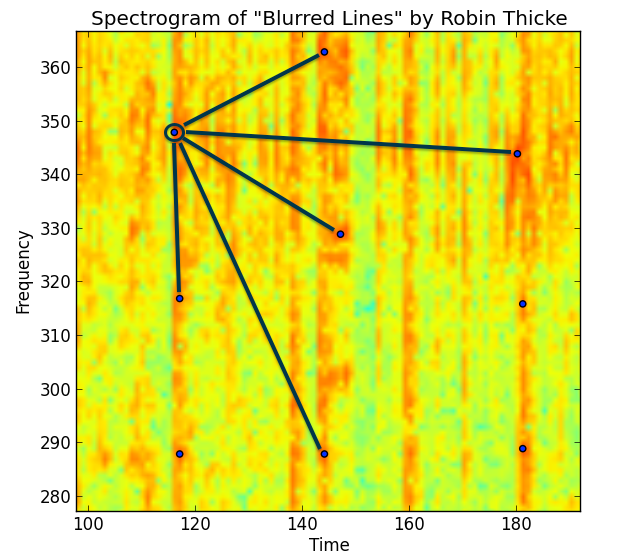
**Spektrogram jako ”odcisk głosu”**

Spektrogram to wizualne przedstawienie widma częstotliwości od dźwięku. Nazywane są czasami ”odciskami głosu”. Spektrogram możemy wyświetlić za pomocą pakietu LibROSA funkcją *librosa.display.specshow.*

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')   
plt.colorbar ()

Gdy mamy już spektrogram możemy go użyć do jednoznacznej identyfikacji piosenki. Możemy zacząć od znalezienia „szczytów” amplitudy. Definiujemy pik jako parę (czas, częstotliwość) odpowiadającą wartości amplitudy, która jest największa w lokalnej „okolicy” wokół niego. Inne pary (czas, częstotliwość) wokół niego mają mniejszą amplitudę, a zatem to hałas.

Po wyodrębnieniu szczytów znaleźliśmy interesujące miejsca w piosence, która je identyfikuje – ”odcisk głosu” . nPatrząc na szczyty spektrogramu i łącząc częstotliwości szczytowe z ich różnicą czasową między nimi, możemy stworzyć skrót, reprezentujący unikalny „odcisk głosu” dla tej piosenki.



Odcisk głosu powinien posiadań między innymi skrót, identyfikator utworu oraz przesunięcie. Odpowiada to oknu czasowemu ze spektrogramu, z którego pochodzi skrót. Odciski nie powinny się również duplikować aby nie marnować miejsca i czasu na niepotrzebne dopasowywania. Po zebraniu odcisków należy je po prostu przyrównywać przy przeszukiwaniu utworów.

Materiał: <https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/>