Введение

Аудиоанализ — область, включающая автоматическое распознавание речи (ASR), цифровую обработку сигналов, а также классификацию, тегирование и генерацию музыки — представляет собой развивающийся поддомен приложений глубокого обучения. Некоторые из самых популярных и распространенных систем машинного обучения, такие как виртуальные помощники Alexa, Siri и Google Home, — это продукты, созданные на основе моделей, извлекающих информацию из аудиосигналов.

Обзор аудиофайлов

Аудио фрагменты представлены в формате .wav. Звуковые волны оцифровываются путем выборки из дискретных интервалов, известных как частота дискретизации (как правило, 44,1 кГц для аудио с CD-качеством, то есть 44 100 семплов в секунду).

Каждый семпл представляет собой амплитуду волны в определенном временном интервале, где глубина в битах (или динамический диапазон сигнала) определяет, насколько детализированным будет семпл (обычно 16 бит, т.е. семпл может варьироваться от 65 536 значений амплитуды).

В обработке сигналов семплинг — это преобразование непрерывного сигнала в серию дискретных значений. Частота дискретизации — это количество семплов за определенный фиксированный промежуток времени. Высокая частота дискретизации приводит к меньшей потере информации, но к большим вычислительным затратам.

Звуковая волна в цифровом формате обозначена красным цветом, а синим — результат семплинга и 4-битного квантования. Справа находится результирующий массив. Приложения по обработке звука

К ним можно отнести:

Индексирование музыкальных коллекций согласно их аудиопризнакам. Рекомендация музыки для радиоканалов. Поиск сходства для аудиофайлов (Shazam). Обработка и синтез речи — генерирование искусственного голоса для диалоговых агентов. Обработка аудиоданных с помощью Python

Звук представлен в форме аудиосигнала с такими параметрами, как частота, полоса пропускания, децибел и т.д. Типичный аудиосигнал можно выразить в качестве функции амплитуды и времени.

Время/частота. Некоторые устройства могут улавливать эти звуки и представлять их в машиночитаемом формате. Примеры этих форматов:

wav (Waveform Audio File) mp3 (MPEG-1 Audio Layer 3) WMA (Windows Media Audio)
Процесс обработки звука включает извлечение акустических характеристик, относящихся к поставленной задаче, за которыми следуют схемы принятия решений, которые включают обнаружение, классификацию и объединение знаний. К счастью, некоторые библиотеки Python помогают облегчить эту задачу.

```
! pip install librosa
```

```
    Requirement already satisfied: librosa in /usr/local/lib/python3.6/dist-pac
    Requirement already satisfied: joblib>=0.12 in /usr/local/lib/python3.6/dis
    Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dis
    Requirement already satisfied: resampy>=0.2.0 in /usr/local/lib/python3.6/d
    Requirement already satisfied: decorator>=3.0.0 in /usr/local/lib/python3.6
    Requirement already satisfied: numpy>=1.8.0 in /usr/local/lib/python3.6/dis
    Requirement already satisfied: audioread>=2.0.0 in /usr/local/lib/python3.6
    Requirement already satisfied: six>=1.3 in /usr/local/lib/python3.6/dist-pa
    Requirement already satisfied: numba>=0.38.0 in /usr/local/lib/python3.6/di
    Requirement already satisfied: scikit-learn!=0.19.0,>=0.14.0 in /usr/local/
    Requirement already satisfied: llvmlite<0.32.0,>=0.31.0dev0 in /usr/local/l
    Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-
```

```
import librosa
audio_data = 'vocaroo.mp3'       # https://voca.ro/iMPozIbzB8T
x , sr = librosa.load(audio_data)
print(type(x), type(sr))
print(x.shape, sr)
```

```
    <class 'numpy.ndarray'> <class 'int'>
    (94316,) 22050
```

Этот фрагмент возвращает звуковой временной ряд в качестве массива numpy с частотой дискретизации по умолчанию 22 кГц моно. Это поведение можно изменить с помощью повторного семплинга на частоте 44,1 кГц.

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

```
    (array([-6.2426478e-03, -6.0658096e-03, -4.9178000e-03, ...,
            -2.3266051e-05, -1.3288871e-05, -4.5128440e-06], dtype=float32), 44
```

Повторный семплинг также можно отключить:

```
librosa.load(audio_data, sr=None)
```

```
    (array([-6.4697266e-03, -1.0070801e-03,  5.7983398e-04, ...,
             0.0000000e+00, -3.0517578e-05, -3.0517578e-05], dtype=float32), 11
```

Проигрывание аудио:

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

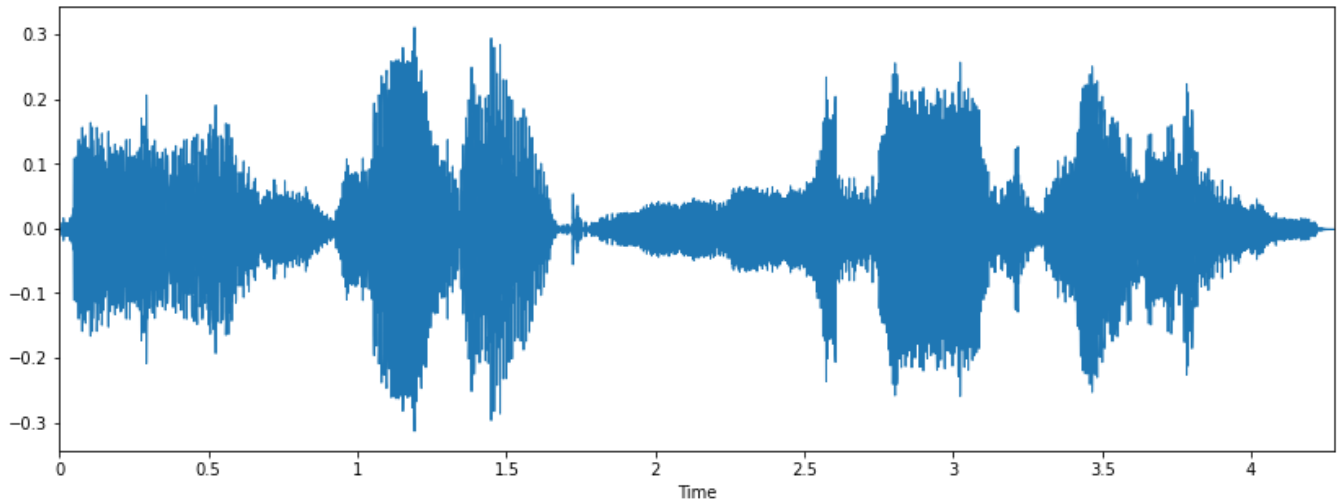
-0:04

Визуализация аудио

График массива аудио:

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

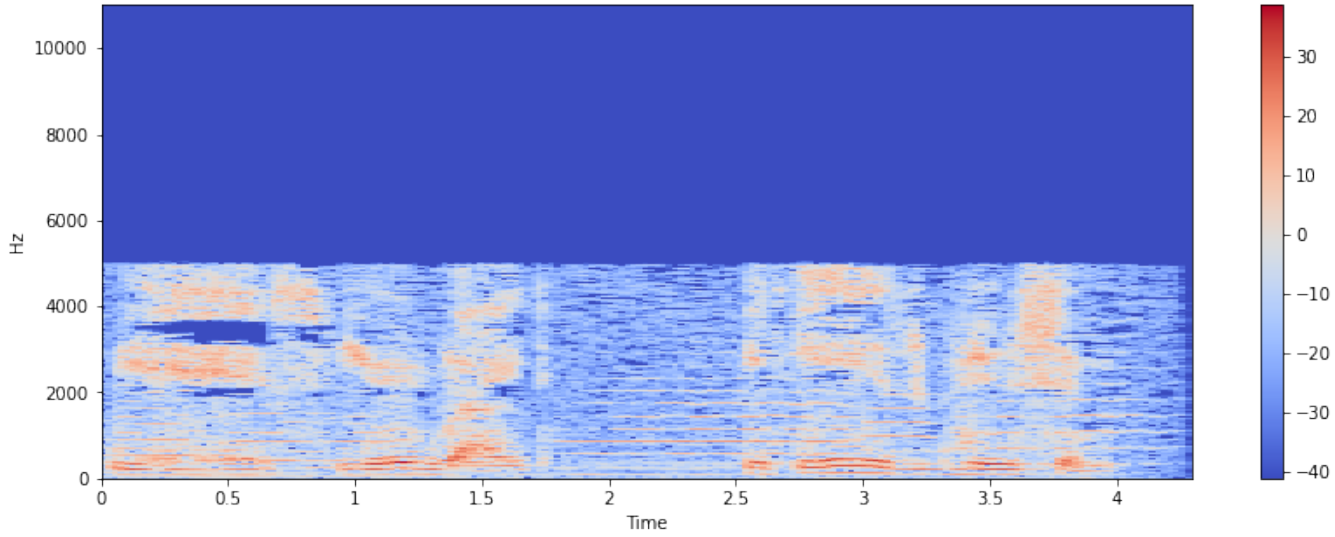
```
<matplotlib.collections.PolyCollection at 0x7f00f1c93198>
```



Спектограмма:

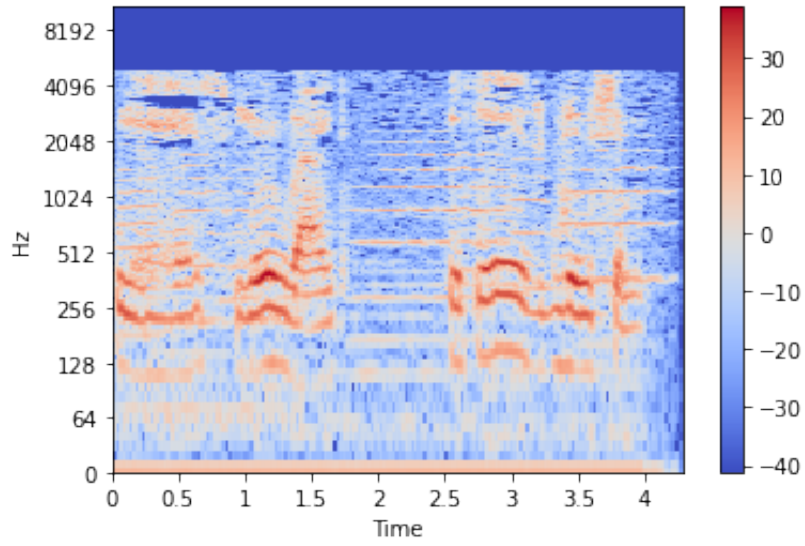
```
X = librosa.stft(x)   #   преобразует данные в кратковременное преобразование Фу
# С помощью STFT можно определить амплитуду различных частот, воспроизводимых в
Xdb = librosa.amplitude_to_db(abs(X))
plt.figure(figsize=(14, 5))
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='hz')  # отображение
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f00f179db00>
```




```
librosa.display.specshow(Xdb, sr=sr, x_axis='time', y_axis='log') # преобразован
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f00f0ee9ba8>
```



Создание аудиосигнала:

```
import numpy as np
sr = 22050 # частота дискретизации
T = 5.0    # секунды
t = np.linspace(0, T, int(T*sr), endpoint=False) # переменная времени
x = 0.5*np.sin(2*np.pi*220*t) # чистая синусоидная волна при 220 Гц
# проигрывание аудио
ipd.Audio(x, rate=sr) # загрузка массива NumPy
# сохранение аудио
librosa.output.write_wav('tone_220.wav', x, sr)
```

Извлечение признаков из аудио сигнала

Каждый аудиосигнал состоит из множества признаков.

Спектральные (частотные) признаки получаются путем преобразования временного сигнала в частотную область с помощью преобразования Фурье. К ним относятся частота основного тона, частотные компоненты, спектральный центроид, спектральный поток, спектральная плотность, спектральный спад и т.д.

Спектральный центроид

Указывает, на какой частоте сосредоточена энергия спектра или, другими словами, указывает, где расположен «центр масс» для звука. Схож со средневзвешенным значением:

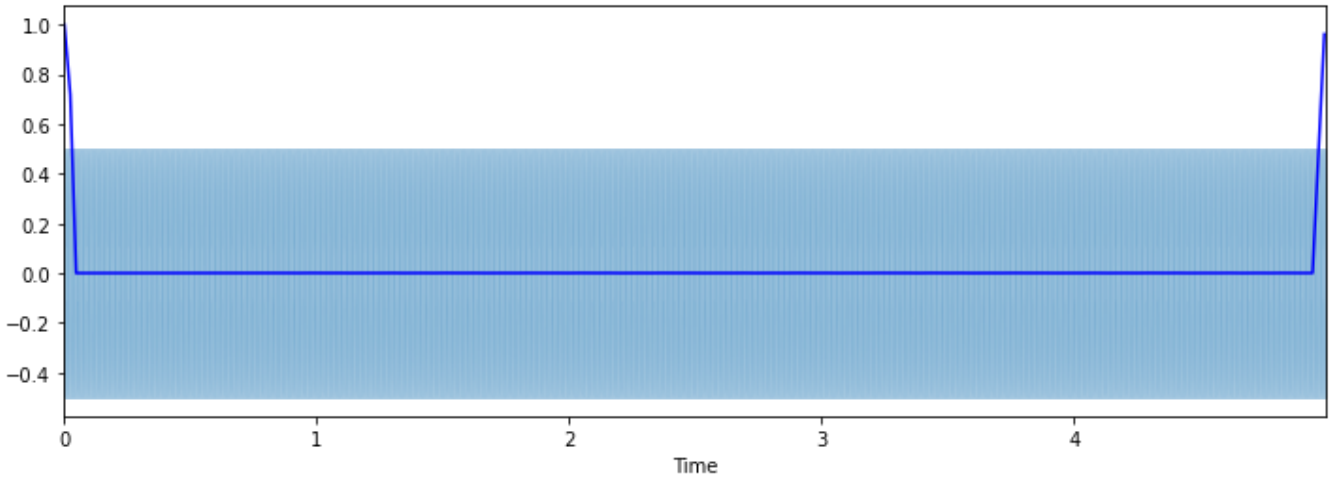
$$f_c = \frac{\sum_k S(k) f(k)}{\sum_k S(k)}$$

где S(k) — спектральная величина элемента разрешения k, а f(k) — частота элемента k.

```
import sklearn
spectral_centroids = librosa.feature.spectral_centroid(x, sr=sr)[0]
spectral_centroids.shape
(775,)
# Вычисление временной переменной для визуализации
plt.figure(figsize=(12, 4))

frames = range(len(spectral_centroids))
t = librosa.frames_to_time(frames)
# Нормализация спектрального центроида для визуализации
def normalize(x, axis=0):
    return sklearn.preprocessing.minmax_scale(x, axis=axis)
# Построение спектрального центроида вместе с формой волны
librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_centroids), color='b')
```

```
[<matplotlib.lines.Line2D at 0x7f00f0ec39e8>]
```



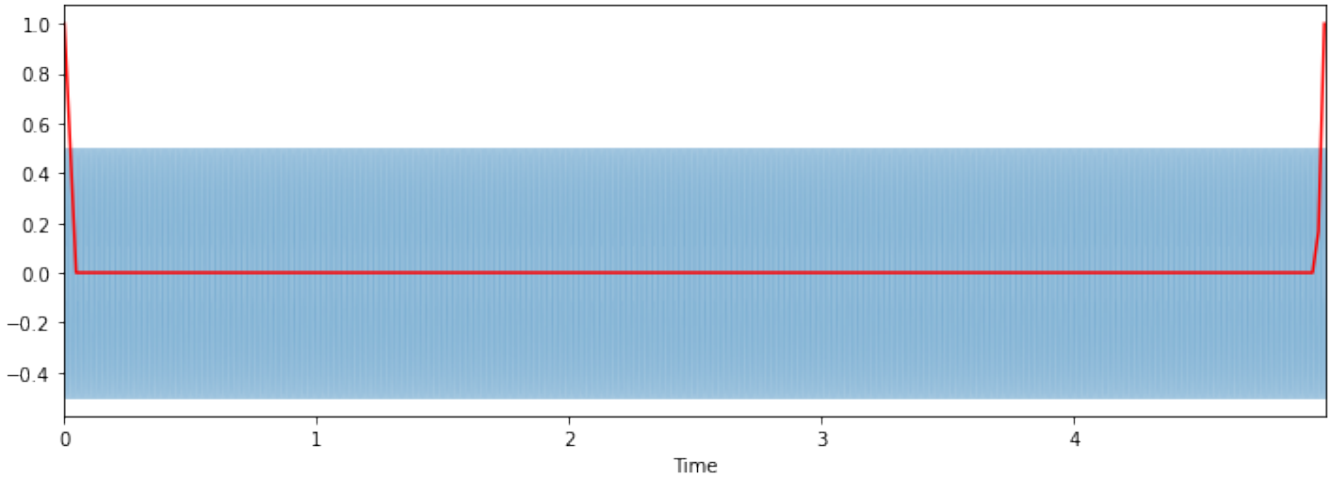
Спектральный спад

Это мера формы сигнала, представляющая собой частоту, в которой высокие частоты снижаются до 0. Чтобы получить ее, нужно рассчитать долю элементов в спектре мощности, где 85% ее мощности находится на более низких частотах.

```
spectral_rolloff = librosa.feature.spectral_rolloff(x+0.01, sr=sr)[0]
plt.figure(figsize=(12, 4))

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

```
[<matplotlib.lines.Line2D at 0x7f00f0e32b38>]
```



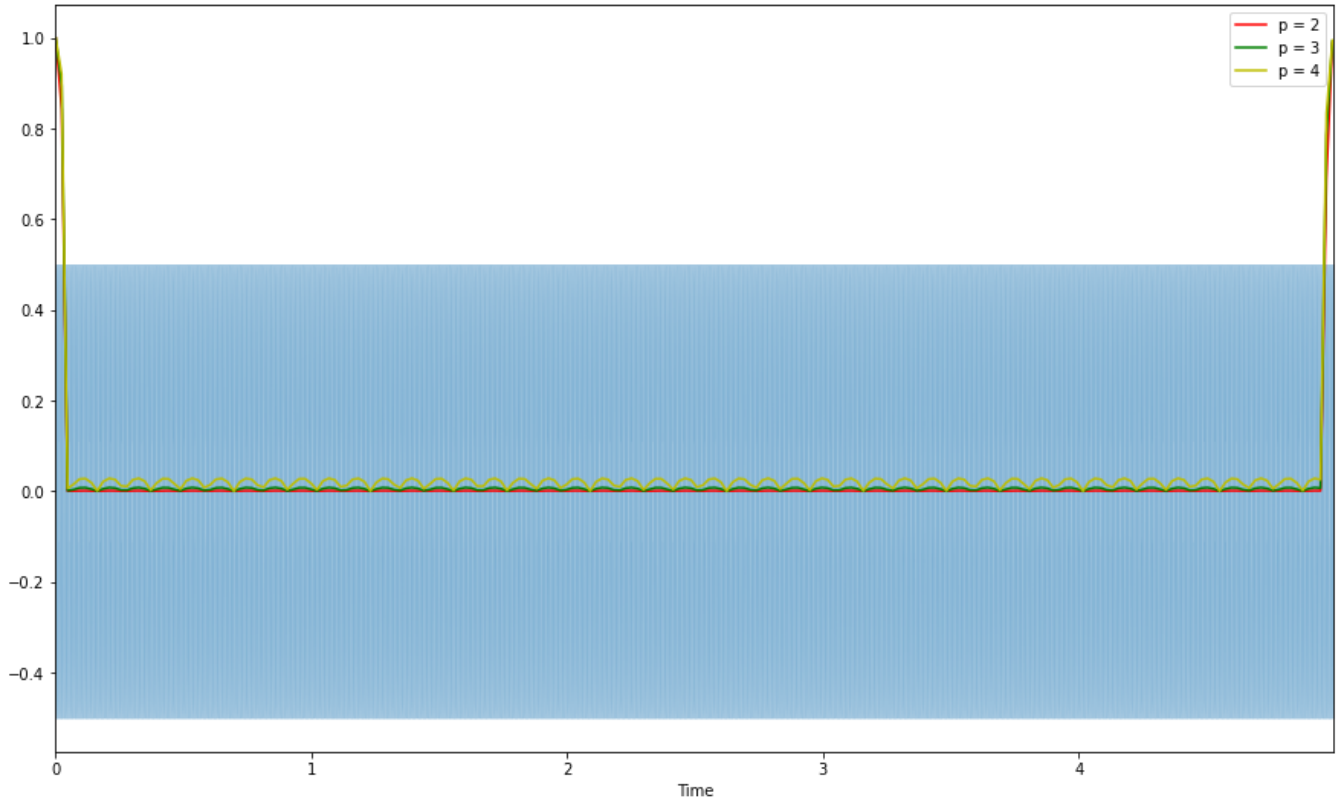
Спектральная ширина

Спектральная ширина определяется как ширина полосы света на половине максимальной точки (или полная ширина на половине максимума [FWHM]) и представлена двумя вертикальными красными линиями и λSB на оси длин волн

```
spectral_bandwidth_2 = librosa.feature.spectral_bandwidth(x+0.01, sr=sr)[0]
spectral_bandwidth_3 = librosa.feature.spectral_bandwidth(x+0.01, sr=sr, p=3)[0]
spectral_bandwidth_4 = librosa.feature.spectral_bandwidth(x+0.01, sr=sr, p=4)[0]
plt.figure(figsize=(15, 9))

librosa.display.waveplot(x, sr=sr, alpha=0.4)
plt.plot(t, normalize(spectral_bandwidth_2), color='r')
plt.plot(t, normalize(spectral_bandwidth_3), color='g')
plt.plot(t, normalize(spectral_bandwidth_4), color='y')
plt.legend(('p = 2', 'p = 3', 'p = 4'))
```

```
<matplotlib.legend.Legend at 0x7f00f0e816a0>
```



Скорость пересечения нуля

Простой способ измерения гладкости сигнала — вычисление числа пересечений нуля в пределах сегмента этого сигнала. Голосовой сигнал колеблется медленно. Например, сигнал 100 Гц будет пересекать ноль 100 раз в секунду, тогда как «немой» фрикативный сигнал может иметь 3000 пересечений нуля в секунду.

$$zcr = \frac{1}{T-1}\sum_{t=1}^{T-1}\mathbb{I}\{s_t s_{t-1} < 0\}$$

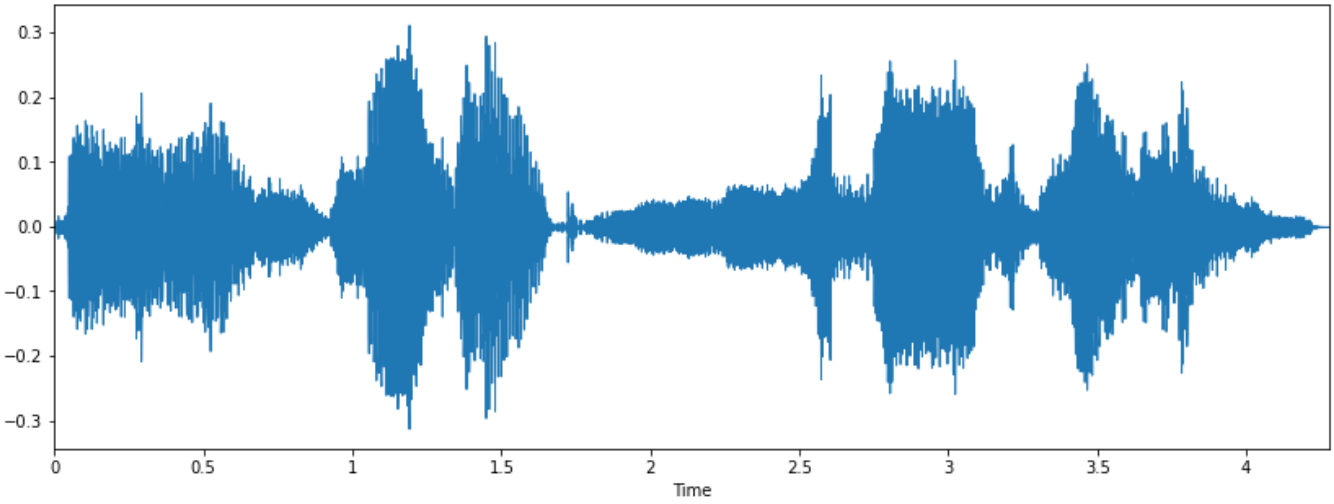
Fig. 4. Formula to calculate the Zero Crossing Rate

$$s_t \text{ is the signal of length } t$$
$$II\{X\} \text{ is the indicator function } (=1 \text{ if } X \text{ true, else } =0)$$

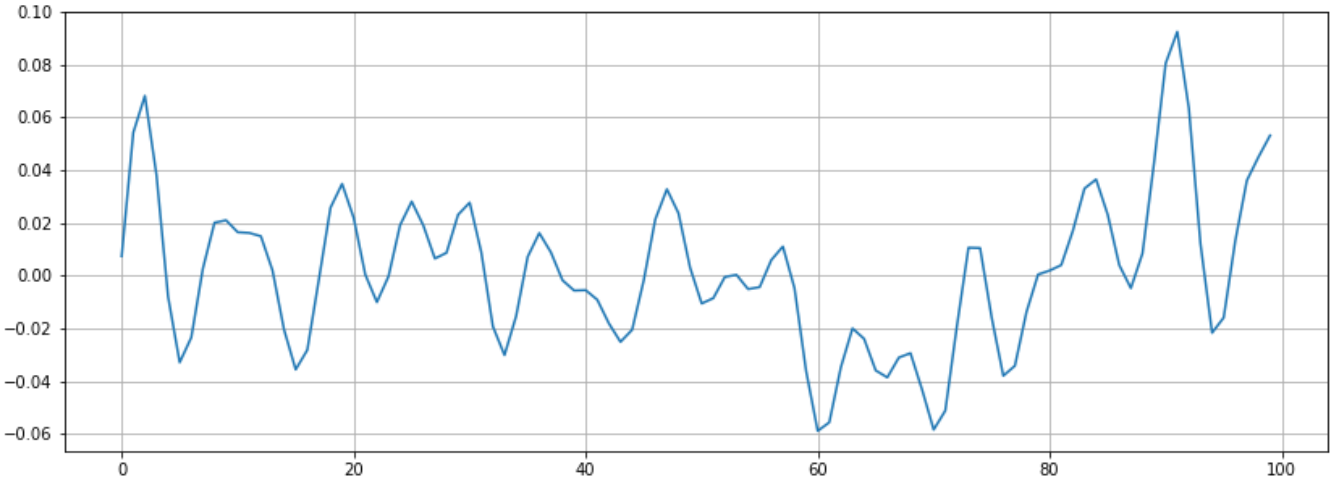
Формула для расчета скорости пересечения нуля, где St — сигнал длины t, II{X} — функция-индикатор (=1 if X true, else =0). Более высокие значения наблюдаются в таких высоко ударных звуках, как в металле и роке. Теперь визуализируем этот процесс и рассмотрим вычисление скорости пересечения нуля.

```
x, sr = librosa.load(audio_data)
# Построение графика сигнала:
plt.figure(figsize=(14, 5))
librosa.display.waveplot(x, sr=sr)
```

```
<matplotlib.collections.PolyCollection at 0x7f00f0d43208>
```




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



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

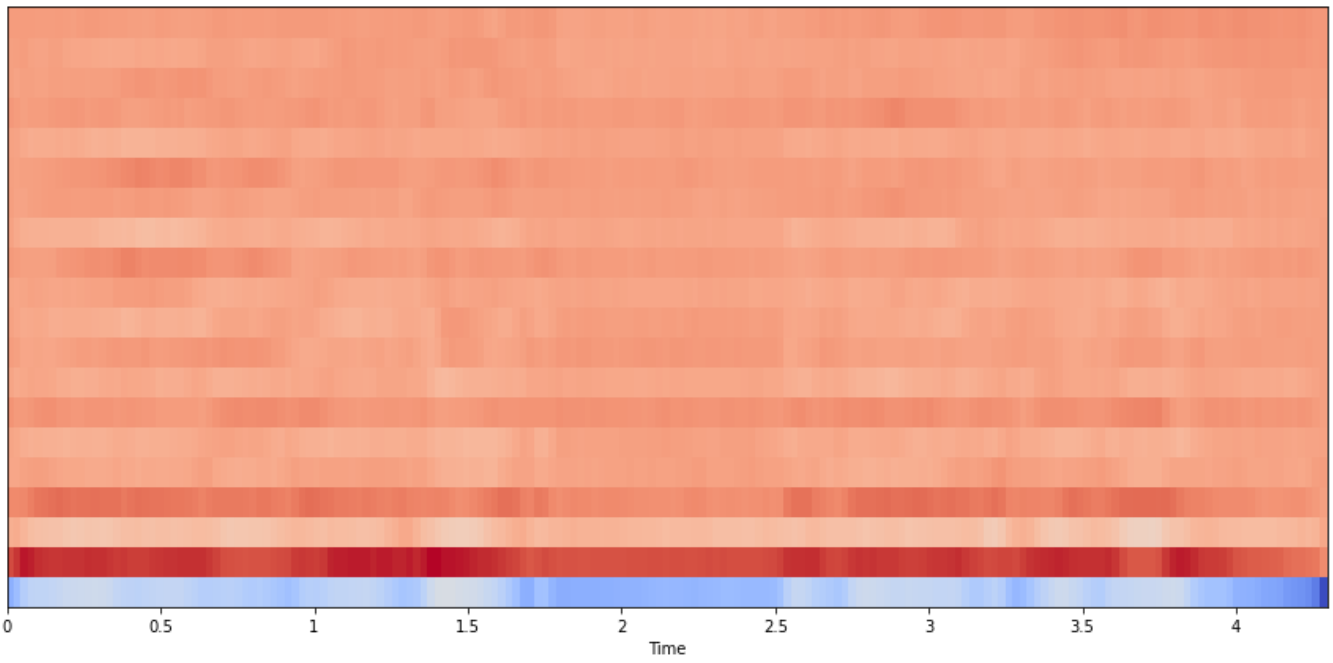
```
22
```

Мел-частотные кепстральные коэффициенты (MFCC)

Представляют собой небольшой набор признаков (обычно около 10–20), которые кратко описывают общую форму спектральной огибающей. Они моделируют характеристики человеческого голоса.

```
mfccs = librosa.feature.mfcc(x, sr=sr)
print(mfccs.shape)
# Отображение MFCC:
plt.figure(figsize=(15, 7))
librosa.display.specshow(mfccs, sr=sr, x_axis='time')
```

```
(20, 185)
<matplotlib.axes._subplots.AxesSubplot at 0x7f00efc69fd0>
```

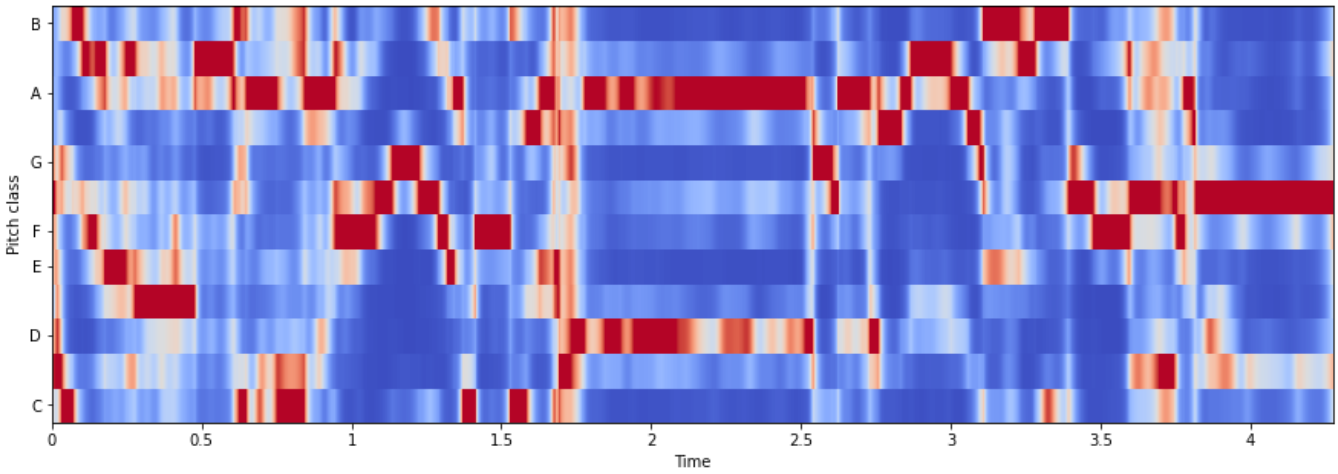


Цветность

Признак или вектор цветности обычно представлен вектором признаков из 12 элементов, в котором указано количество энергии каждого высотного класса {C, C#, D, D#, E, ..., B} в сигнале. Используется для описания меры сходства между музыкальными произведениями.

```
hop_length = 12
chromagram = librosa.feature.chroma_stft(x, sr=sr, hop_length=hop_length)
plt.figure(figsize=(15, 5))
librosa.display.specshow(chromagram, x_axis='time', y_axis='chroma', hop_length=
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f00e7c17f98>
```



Классификация жанров музыки с помощью ANN

Датасет - http://marsyas.info/downloads/datasets.html

Набор данных состоит из 1000 звуковых треков, длина каждого составляет 30 секунд. Он содержит 10 жанров, каждый из которых представлен 100 треками. Все дорожки — это монофонические 16-битные аудиофайлы 22050 Гц в формате .wav.

Жанры, представленные в наборе:

Блюз

Классика

Кантри

Диско

Хип-хоп

Джаз

Метал

Поп

Регги

Рок

```
# Импортируем все необходимые библиотеки.

import librosa
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
from PIL import Image
import pathlib
import csv

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler

import keras
from keras import layers
from keras import layers
import keras
from keras.models import Sequential

import warnings
warnings.filterwarnings('ignore')


# Конвертируем файлы аудиоданных в PNG или извлекаем спектрограмму для каждого а

cmap = plt.get_cmap('inferno')
plt.figure(figsize=(8,8))
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'.split
for g in genres:
    pathlib.Path(f'img_data/{g}').mkdir(parents=True, exist_ok=True)
    for filename in os.listdir(f'./drive/My Drive/genres/{g}'):
        songname = f'./drive/My Drive/genres/{g}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=5)
        plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='d
        plt.axis('off');
        plt.savefig(f'img_data/{g}/{filename[:-3].replace(".", "")}.png')
        plt.clf()

    <Figure size 576x576 with 0 Axes>


# Создание заголовка для файла CSV.

header = 'filename chroma_stft rmse spectral_centroid spectral_bandwidth rolloff
for i in range(1, 21):
    header += f' mfcc{i}'
header += ' label'
header = header.split()
```

```
# Извлекаем признаки из спектрограммы: MFCC, спектральный центроид, частоту пере

file = open('dataset.csv', 'w', newline='')
with file:
    writer = csv.writer(file)
    writer.writerow(header)
genres = 'blues classical country disco hiphop jazz metal pop reggae rock'.split
for g in genres:
    for filename in os.listdir(f'./drive/My Drive/genres/{g}'):
        songname = f'./drive/My Drive/genres/{g}/{filename}'
        y, sr = librosa.load(songname, mono=True, duration=30)
        rmse = librosa.feature.rmse(y=y)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        to_append = f'{filename} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean
        for e in mfcc:
            to_append += f' {np.mean(e)}'
        to_append += f' {g}'
        file = open('dataset.csv', 'a', newline='')
        with file:
            writer = csv.writer(file)
            writer.writerow(to_append.split())


# Выполняем предварительную обработку данных, которая включает загрузку данных С

data = pd.read_csv('dataset.csv')
data.head()

# Удаление ненужных столбцов
data = data.drop(['filename'],axis=1)

# Создание меток
genre_list = data.iloc[:, -1]
encoder = LabelEncoder()
y = encoder.fit_transform(genre_list)

# Масштабирование столбцов признаков
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data.iloc[:, :-1], dtype = float))

# Разделение данных на обучающий и тестовый набор
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# Создаем модель ANN.

model = Sequential()
model.add(layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


# Подгоняем модель:

classifier = model.fit(X_train,
                       y_train,
                       epochs=100,
                       batch_size=128)
```

```
7/7 [==============================] - 0s 4ms/step - loss: 0.0332 - accurac
Epoch 72/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0339 - accurac
Epoch 73/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0318 - accurac
Epoch 74/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0362 - accurac
Epoch 75/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0307 - accurac
Epoch 76/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0307 - accurac
Epoch 77/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0271 - accurac
Epoch 78/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0264 - accurac
Epoch 79/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0240 - accurac
Epoch 80/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0273 - accurac
Epoch 81/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0248 - accurac
Epoch 82/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0289 - accurac
Epoch 83/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0246 - accurac
Epoch 84/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0226 - accurac
Epoch 85/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0208 - accurac
Epoch 86/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0198 - accurac
Epoch 87/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0173 - accurac
Epoch 88/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0175 - accurac
```

```
Epoch 89/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0243 - accurac
Epoch 90/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0286 - accurac
Epoch 91/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0195 - accurac
Epoch 92/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0196 - accurac
Epoch 93/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0189 - accurac
Epoch 94/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0150 - accurac
Epoch 95/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0148 - accurac
Epoch 96/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0171 - accurac
Epoch 97/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0142 - accurac
Epoch 98/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0133 - accurac
Epoch 99/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0131 - accurac
Epoch 100/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0152 - accurac
```