Практикум на ЭВМ.
Аудио.
Активность голоса

Баев А.Ж.

Казахстанский филиал МГУ

14 марта 2020

Скачаем данные из youtube

Установим youtube data loader

```
sudo apt install youtube-dl
```

Посмотрим, в каком виде можно скачать 16-секундный фрагмент «лошадью ходи» (всё в одну строку):

```
youtube-dl https://www.youtube.com/watch?
v=sd_KfEdQdic -F
```

Выбираем аудио формат со сжатием (кодеком) opus. Данный кодек, кстати, используется и в телеграмм для аудио сообщений.

```
...
251 webm  audio only tiny 92k,
    opus @160k (48000Hz), 180.99KiB
...
```

Скачиваем (укажем код 251)

```
youtube-dl https://www.youtube.com/watch?
v=sd_KfEdQdic --extract-audio -f 251
```

Конвертируем файл

Установим `ffmpeg` (конвертирует аудио/видео в разные форматы).

```
sudo apt install ffmpeg
```

Сконвертируем файл в формат wav (без сжатия), с которым и будем работать в коде. Для более простой работы сконвертируем в моноканальное аудио (audio channel = 1).

```
ffmpeg -i file.opus -ac 1 file.wav
```

Изучаем формат wav

Установим sox (позволяет работать с аудио данными в терминале).

```
sudo apt install sox
```

Глянем на файл с которым будет

```
soxi file.wav
```

Информация

```
Input File     : 'file.wav'
Channels       : 1
Sample Rate    : 48000
Precision      : 16-bit
Duration       : 00:00:16.13 = 774373 samples
File Size      : 1.55M
Bit Rate       : 768k
Sample Encoding: 16-bit Signed Integer PCM
```

Пояснения wav

1. Исходный аудио сигнал — одномерный массив (временной ряд).
2. Каждая секунда звучания — это 48000 чисел (частота дискретизации).
3. Каждое число занимает 16 бит.
4. Общая длительность 16.13 секунд определяется из размера массива (774373 числа) и (частоты дискретизации).
5. Размер файла определяется из размера одного числа и частоты дискретизации.
6. Каждое число — 16-битное знаковое целое число (short).

```
Input File      : 'file.wav'
Channels        : 1
Sample Rate     : 48000
Precision       : 16-bit
Duration        : 00:00:16.13 = 774373 samples
File Size       : 1.55M
Bit Rate        : 768k
Sample Encoding: 16-bit Signed Integer PCM
```

Код на C++

Цель: написать упрощённый вариант программы soxi.

1. Как устроен wav файл

wav файл относится к классу riff файлов. То есть это бинарный формат, в котором первые несколько байт определяют заголовок, а далее идут данные. Читать header умеет встроенная утилита file:

```
file file.wav
```

```
file.wav: RIFF (little-endian) data, WAVE audio,
Microsoft PCM, 16 bit, mono 48000 Hz
```

1. Как устроен wav файл

Для чтения заголовка создадим структуру

```
struct WavHeader {
  char chunkId[4];              // 'RIFF'
  unsigned int chunkSize;       // file size - 8
  char format[4];               // 'WAVE'
  char subchunk1Id[4];          // 'fmt '
  unsigned int subchunk1Size; // 16 for 'pcm'
  unsigned short audioFormat; // 1 for 'pcm'
  unsigned short numChannels; // 1 for 'mono'
  unsigned int sampleRate;
  unsigned int byteRate;
  unsigned short blockAlign;  // byte per sample
  unsigned short bitsPerSample;
  char subchunk2Id[4];          // 'data'
  unsigned int subchunk2Size; // byte (size - 44)
};
```

https://audiocoding.ru/articles/2008-05-22-wav-file-structure

1. Как устроен wav файл

Как читать заголовок аудио файла

```
std::string wav_file_path("file.wav");
std::ifstream input_stream(wav_file_path.c_str(),
                           std::ios::binary);
if (!input_stream) {
  print(wav_file_path + ": cannot open file");
  exit(1);
}

WavHeader header;
input_stream.read((char *) &header,
                  sizeof(header));
if (strncmp(header.format, "WAVE", 4) != 0) {
  print(wav_file_path + ": file is not wav.");
  exit(2);
}
```

1. Как устроен wav файл

Как получить значения из soxi?

```
header.numChannels
header.sampleRate
header.bitsPerSample
get_duration_seconds(header) ???
get_duration_samples(header) ???
header.subchunk2Size
```

1. Как устроен wav файл

Читаем непосредственно данные

```
int samples_number = get_duration_samples(header);
std::vector<int> data(samples_number);
for (int i = 0; i < samples_number; i++) {
  short sample;
  input_stream.read((char *) &sample,
                    sizeof(short));
  data[i] = sample;
}
input_stream.close();
```

2. Выделяем фрагменты с голосом

Считаем интенсивность одного фрагмента $[start; end)$.

```
double energy = 0;
for (int i = start; i < end; i++)
  energy += data[i] * data[i] / (end - start);
energy = sqrt(energy) / 32768;
```

$$energy = \frac{1}{32768}\sqrt{\frac{1}{n}\sum_{i=1}^{n} d_i^2}$$

2. Выделяем фрагменты с голосом

Считаем интенсивность всех фрагментов с шагом `segment_duration`.

```
std::vector<double> get_segments_energy(
  const std::vector<int> &data,
  int segment_duration
)
```

2. Оставляем фрагменты выше некого порога

Оставляем только то, что выше порога `threshold`. То есть при `threshold = 0.1` массив (0.07 0.11 0.80 0.30 0.05 0.13 0.43 0.00) получим (01110110)

```
std::vector<bool> get_vad_mask(
  const std::vector<double> &data,
  double threshold
)
```

2. Выделяем фрагменты с голосом

Выполняем сжатие маски (01110110) на (1-4, 5-7)

```
struct Segment {
  int start;
  int stop;
};
std::vector<Segment> mask_compress(
  const std::vector<bool> &data
)
```

2. Выделяем фрагменты с голосом

Сохранить нарезанные части файла.

```
int save_wav(
  std::string file_path,
  const WavHeader &header,
  const std::vector<int> &data,
  int start,
  int stop
)
```