Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления» (ИУ)

КАФЕДРА «Системы обработки информации и управления» (ИУ5)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Классификация пола человека по голосовым записям с помощью нейронных сетей»

Студент группы ИУ5-31М _____ Р.В. Фонканц

Руководитель курсового проекта _____ Ю.Е. Гапанюк

2022 г.

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ5

_____ В.И. Терехов

« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по теме «Классификация пола человека по голосовым записям с помощью нейронных сетей»

Студент группы ИУ5-31М

Фонканц Роман Викторович

Направленность курсового проекта (учебная, исследовательская, практическая, производственная, др.)

исследовательская

Источник тематики (кафедра, предприятие, НИР) _____

График выполнения курсового проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Техническое задание: Рассмотреть назначение и структуру WAV-файлов, а также описание осциллограммы и спектрограммы аудиосигналов; исследовать принцип работы преобразования Фурье и его применимость к предобработке звуковых данных, изучить мел-шкалу и способ получения мел-кепстральных коэффициентов; разработать модуль на основе технологии нейронных сетей, реализующий классификацию пола человека по записи его голоса. _____

Оформление курсового проекта:

Расчетно-пояснительная записка на 27 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « ___ » _____ 2022 г.

Руководитель курсового проекта _____ Ю.Е. Гапанюк
Студент _____ Р.В. Фонканц

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

Введение

Невозможно представить нормальную жизнь человека без способности полноценно пользоваться речевым аппаратом. С помощью речи мы не только передаем информацию другим людям, но и воспринимаем огромный пласт информации об окружающем нас мире. Речь – важнейший инструмент коммуникации между людьми. Работа с естественным языком — одно из важнейших направлений развития искусственного интеллекта, и в последние несколько лет технологии развиваются крайне стремительно, буквально одна революция следует за другой. С помощью различных методов машинного обучения решается широкий спектр задач по обработке и анализу естественного языка. Одной из таких задач является классификация пола человека по его голосу. Данной задаче будет посвящена настоящая работа.

В данной работе будут рассмотрены назначение и структура WAV-файлов, а также описание осциллограммы и спектрограммы аудиосигналов, исследован принцип работы преобразования Фурье и его применимость к предобработке звуковых данных, изучена мел-шкала и способ получения наиболее эффективных аудиопризнаков – мел-кепстральных коэффициентов. В практической части работы будет проведена обработка и анализ аудиозаписей речи людей разных полов и разработан модуль на основе технологии нейронных сетей, реализующий классификацию пола человека по записи его голоса.

WAV-файлы: тип, назначение, формат и структура

WAV (Waveform Audio file) – это специализированный формат, предназначенный для обработки, передачи и хранения аудиоданных в цифровом виде. WAV является одним из самых популярных и широко используемых аудио форматов. В формате WAV любая аудио информация хранится в оригинальном виде без потерь и сжатия в отличие от многих других аудио форматов, использующих алгоритмы сжатия с потерями (Например, MP3, AAC, OGG Vorbis и другие).

Формат WAV был разработан в 1991 году двумя крупнейшими американскими компаниями: IBM и Microsoft, ввиду чего применяется на платформах на базе операционной системы Windows, и, что интересно, является стандартным форматом записи компакт-дисков. [1] Практическое любое современное приложение, предназначенное для работы с различными аудио файлами, поддерживает и файлы в формате WAV.

В основе WAV лежит так называемая импульсно-кодовая модуляция или PCM (Pulse Code Modulation). По этой причине, область данных WAV файла содержит последовательную совокупность численных значений. Каждое такое значение равно уровню аналогового сигнала, который был оцифрован, в определенный момент времени. Учитывая эту особенность и тот факт, что при записи WAV файла данные не подвергаются процессам сжатия, на выходе мы получаем высокое качество звука (потери и искажения при оцифровке ничтожны и находятся за гранью чувствительности человеческого слуха) и большой объем записанных в формате WAV файлов.

В основном, WAV используется для первоначальной оцифровки аналогового сигнала с последующей конвертацией в другие аудио форматы. Как правило, конвертация производится в форматы с высоким коэффициентом

сжатия, но потерей части полезной информации (т.е. снижается качество звука). Соответственно, если целостность данных и качество для нас не менее важны, чем занимаемый объем памяти, перед нами встает задача кодирования/сжатия WAV файлов, при котором одновременно уменьшится занимаемый объем и не произойдет потеря или искажение оцифрованного сигнала.

Чтобы лучше понимать особенности данного формата, рассмотрим устройство и структуру WAV файла. Важно отметить, что любой WAV файл использует стандартную RIFF-структуру, которая группирует содержимое файла из специальных отдельных секций - чанков (англ. chunk). [2] Каждая секция файла состоит из двух основных блоков: блок с названием секции и блок с информацией данной секции (упрощенный формат файла представлен на рисунке 1). Файл WAV может включать достаточно большое количество различных секций, но основные из них – это секция «fmt» и «data». Все остальные секции являются опциональными. Среди них могут быть те, которые определяют ключевые точки, перечисляют параметры инструментов, хранят информацию о приложении и т.д. [3]

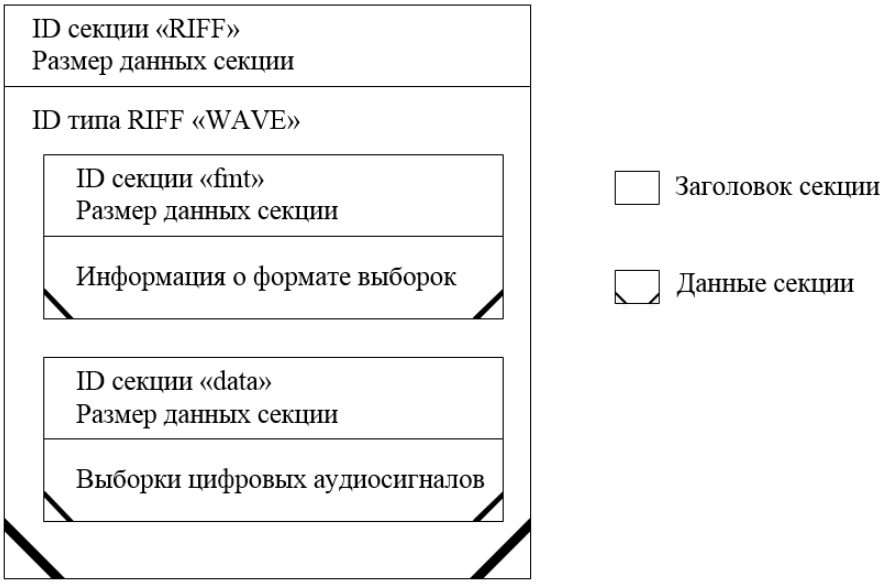


Рисунок 1 - Упрощенный формат WAV-файла

В секции «fmt» расположены параметры аудиоданных файла и информация для их воспроизведения. Секция «data», как ясно из названия, содержит непосредственно данные файла, т.е. оцифрованный сигнал. В таблице 1 представлена структура WAV файла с секциями «fmt» и «data». [4]

Таблица 1 – Структура WAV файла

Расположение в файле	**Название поля**	**Описание поля**
0…3 (4 байта)	chunkId	0x52494646 (Символы «RIFF»)
4…7 (4 байта)	chunkSize	Размер секции/файла без 8 байт
8…11 (4 байта)	format	0x57415645 (Символы «WAVE»)
12…15 (4 байта)	subchunk1Id	0x666d7420 (Символы «fmt»)
16…19 (4 байта)	subchunk1Size	Размер подсекции
20…21 (2 байта)	audioFormat	Формат хранимых аудиоданных
22…23 (2 байта)	numChannels	Количество аудиоканалов
24…27 (4 байта)	sampleRate	Частота квантования/дискретизации
28…31 (4 байта)	byteRate	Битрейт
32…33 (2 байта)	blockAlign	Размер выборки в байтах
34…35 (2 байта)	bitsPerSample	Размер одного кадра в битах
36…39 (4 байта)	subchunk2Id	0x64617461 (Символы «data»)
40…43 (4 байта)	subchunk2Size	Количество байт в области данных.
44…	data	Сами аудиоданные

Как видно, в заголовке WAV файла хранится достаточно много служебной информации, начиная от названий формата файла и секций и заканчивая количеством бит в выборке и кадре. Подо всю служебную информацию выделены первые 43 байта файла, а с 44 байта начинаются сами аудиоданные.

Осциллограмма и спектрограмма. Применение преобразования Фурье для предобработки аудиосигналов

С имеющимся потоком числовых отсчетов, хранящихся в аудиофайлах (применительно к данной работе – в WAV-файлах) крайне неудобно работать. Именно поэтому часто для обработки и анализа аудиоданных используют визуальное представление, а именно осциллограмму и спектрограмму аудиозаписи.

Осциллограмма (на англ. яз. — waveform) — это зависимость амплитуды звукового сигнала от времени. Иначе говоря, осциллограмма – «рисунок», форма звука, складывающаяся из частоты и амплитуды колебаний. [5] С помощью осциллограммы можно оценить различные параметры оцифрованного аудиосигнала — изменение громкости, паузы, иногда даже отдельные ноты. Пример осциллограммы приведен на рисунке 2.

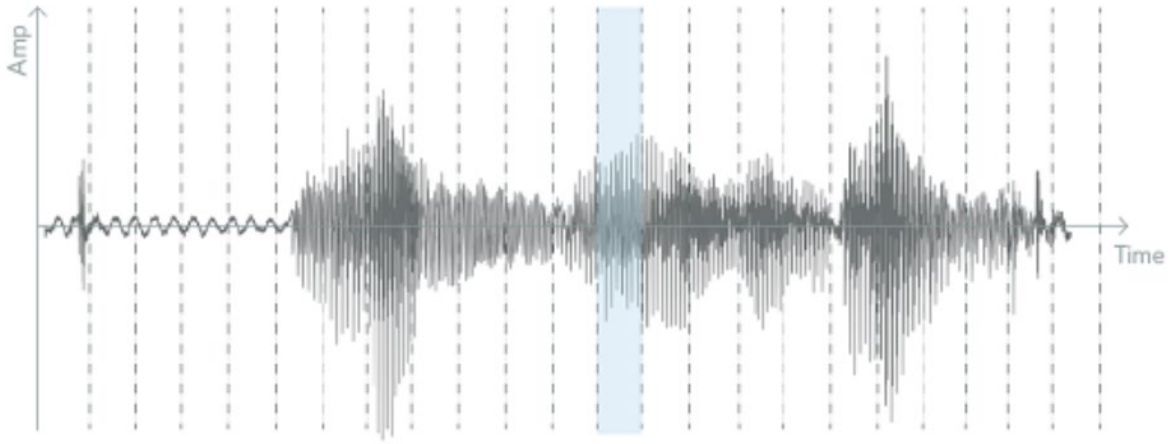


Рисунок 2 – Пример осциллограммы аналогового сигнала

Под спектрограммой понимают «изображение, показывающее зависимость спектральной плотности мощности сигнала от частоты во времени». [6]

Спектрограмму можно сформировать двумя способами: аппроксимировать как набор фильтров или рассчитать сигнал по времени с помощью оконного преобразования Фурье.

Распространенное представление спектрограммы – это двумерная диаграмма: на горизонтальной оси представлено время, по вертикальной оси — частота; третье измерение с указанием амплитуды на определенной частоте в конкретный момент времени представлено интенсивностью или цветом каждой точки изображения. Источники интенсивного звука отображаются красным цветом, а тихие области – темно-синим. Пример спектрограммы изображен на рисунке 3.

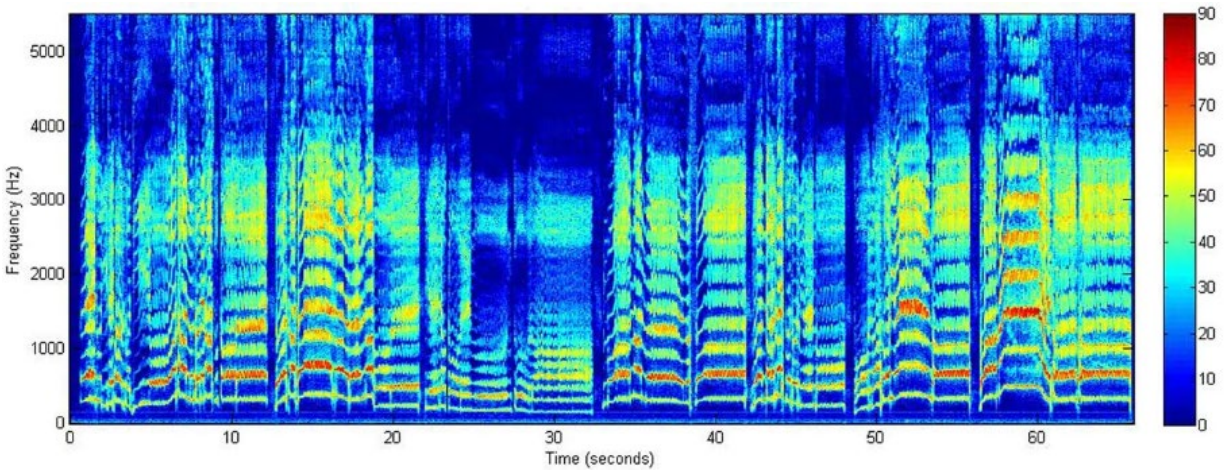


Рисунок 3 - Пример спектрограммы аналогового сигнала

Чаще всего спектрограмму получают из осциллограммы исходного аудиофайла с помощью преобразования Фурье. Рассмотрим его более подробно.

Преобразование Фурье, названное в честь французского математика Жозефа Фурье, представляет собой математическую процедуру, которая позволяет нам определить частотный состав функции. Для инженеров-электронщиков преобразование Фурье обычно применяется к функциям времени, которые мы называем сигналами.

График зависимости напряжения или тока от времени, который мы видим на экране осциллографа, представляет собой интуитивно понятное представление поведения сигнала. Однако это не единственное полезное представление. [7]

Во многих случаях (например, при проектировании радиочастотных систем) нас в первую очередь интересует периодическое поведение сигналов. В частности, нас интересует понимание сигнала относительно синусоидальной периодичности, потому что синусоиды – это уникальное математическое выражение «чистой» частоты.

Преобразование Фурье выявляет элементарную периодичность сигнала, раскладывая этот сигнал на составляющие его синусоидальные частоты и определяя амплитуды и фазы этих составляющих частот.

Слово «разложение» здесь имеет решающее значение. Преобразование Фурье учит нас думать о сигнале во временной области как о сигнале, который состоит из базовых синусоидальных сигналов с различными амплитудами и фазами. [7]

Например, прямоугольный сигнал может быть разложен на бесконечную последовательность синусоид с постоянно уменьшающимися амплитудами и постоянно увеличивающимися частотами. Точная последовательность для прямоугольного сигнала, с развязкой по постоянному току, с периодом T и амплитудой A, может быть записана следующим образом:

$$f_{\text{прямоуг}}(t) = \frac{4A}{\pi} \sum_{k \in \{1,3,5,\ldots\}} \frac{1}{k} \sin\left(\frac{2\pi k t}{T}\right)$$

Мы можем преобразовать это в следующую форму, которая немного более интуитивна:

$$f_{\text{прямоуг}}(t) = \frac{4A}{\pi}\left(\sin(2\pi f t) + \frac{1}{3}\sin(6\pi f t) + \frac{1}{5}\sin(10\pi f t) + \ \ldots\right),$$

где f – частота прямоугольного сигнала в герцах.

На рисунке 4 синим цветом показан исходный прямоугольный сигнал и первые восемь синусоид в бесконечной последовательности.

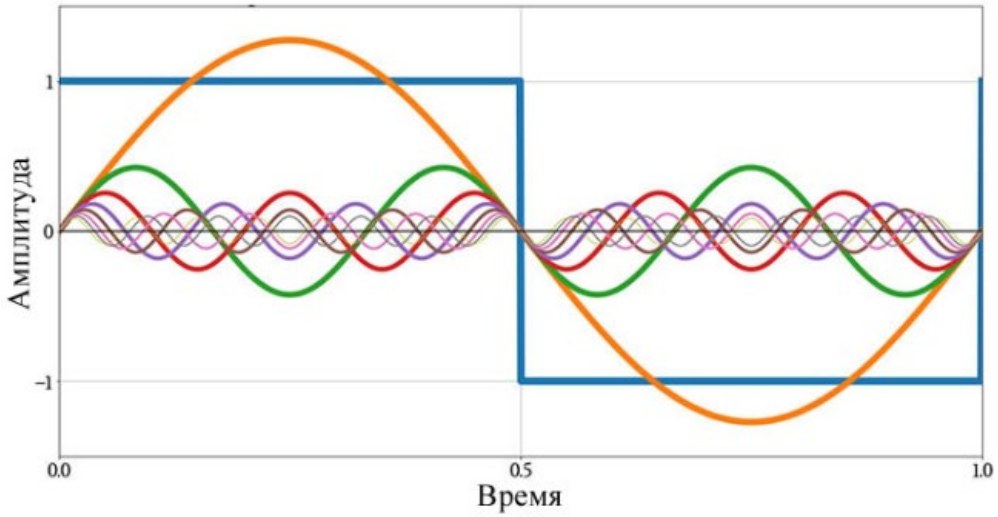


Рисунок 4 - Прямоугольный сигнал и составляющие его синусоиды

Рисунок 5 визуально лучше изображает исходный прямоугольный сигнал и форму сигнала, полученную путем сложения всех составляющих синусоид, показанных выше.

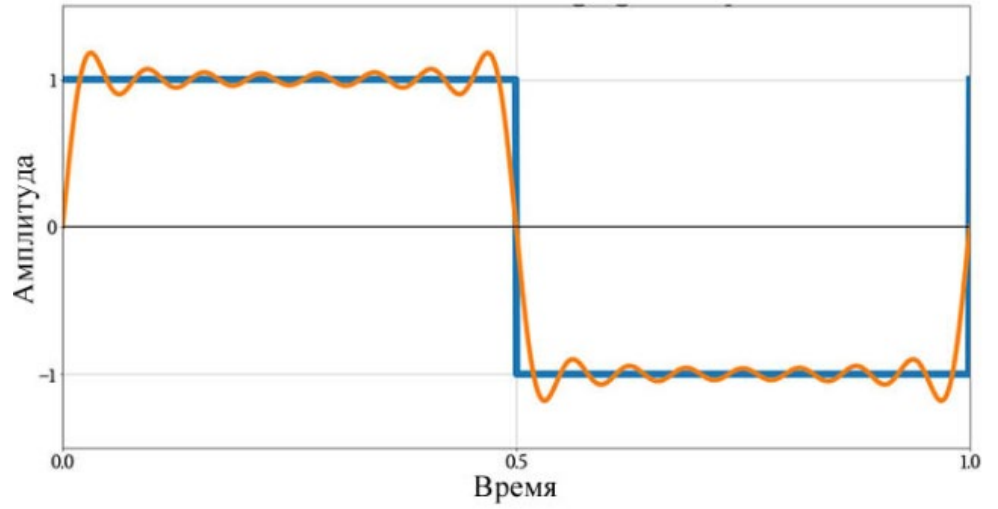


Рисунок 5 - Исходный прямоугольный сигнал и результат сложения синусоид

Когда мы вычисляем преобразование Фурье, мы начинаем с функции времени *f(t)*, а с помощью математического разложения получаем функцию частоты *F(ω)* (обычно в теоретических обсуждениях преобразования Фурье мы используем угловую частоту). Оценка функции *F(ω)* на некоторой определенной угловой частоте, скажем 100 рад/с, дает нам величину и фазу синусоидальной составляющей *f(t)*, частота которой равна 100 рад/с. Если *f(t)* не имеет синусоидальной составляющей на 100 рад/с, то ее амплитуда будет равна нулю.

Преобразование Фурье создает *комплексную* функцию, что означает, что результат самого преобразования не является ни амплитудами частотных компонентов в *f(t)*, ни фазами этих компонентов. Как и с любым комплексным числом, чтобы определить амплитуду или фазу, мы должны выполнить дополнительные вычисления. [8]

Идея комплексного преобразования несколько более интуитивна, когда мы работаем с *дискретным* преобразованием Фурье, а не со «стандартным» преобразованием, в котором мы начинаем с символической функции времени и заканчиваем символической функцией частоты.

Дискретное преобразование Фурье работает с последовательностью числовых значений и создает последовательность **коэффициентов** Фурье. Эти коэффициенты – это типовые комплексные числа (т.е. они имеют форму $a+jb$), и обычно, при анализе частотных составляющих сигнала мы используем амплитуду этих комплексных чисел, вычисленную как $\sqrt{a^2 + b^2}$.

В технических описаниях, отчетах об испытаниях, учебниках и т.д. очень часто встречаются графики частотных составляющих. График зависимости амплитуды от частоты мы часто называем спектром (также спектр может быть трехмерный, если изображается изменение зависимости амплитуды от частоты во времени). Именно спектрограмма является результатом применения

преобразования Фурье к исходному сигналу, имеющему амплитудно-временное представление. [8]

На рисунке 6 показан спектр прямоугольного сигнала с единичной амплитудой и частотой 1 Гц.

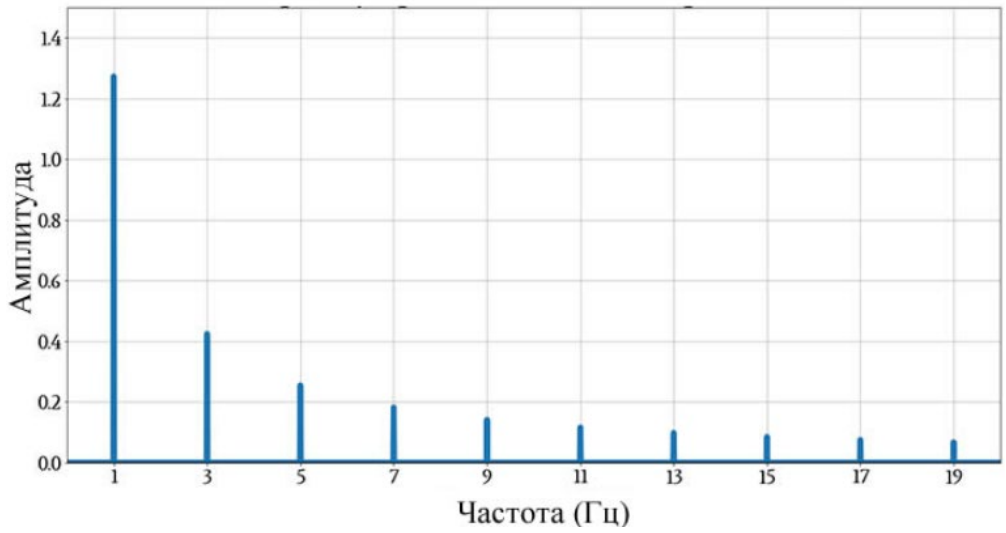


Рисунок 6 - Спектр прямоугольного сигнала с единичной амплитудой и частотой 1 Гц

Если сравнить нанесенные на график амплитуды частотных «всплесков» с амплитудами соответствующих синусоидальных составляющих в бесконечном ряду, описанном ранее, то можно заметить, что они равны.

Мел-шкала. MFCC коэффициенты

Важнейший шаг в анализе речевых данных – это выделение признаков, которые являются "хорошими" для идентификации лингвистического содержания и отбрасыванием всех остальных признаков, отвечающих за шум и эмоции.

Главное, что нужно понять о речи, это то, что звуки, воспроизводимые человеком, определяются формой голосового тракта, включая язык, зубы и т. д. Если мы сможем точно определить форму голосового тракта, то мы будем иметь точное представление о производимой фонеме. Форма голосового тракта описывается огибающей спектра, и задача MFCC состоит в том, чтобы точно представить эту огибающую.

Мел-кепстральные коэффициенты были введены S. Davis и P. Mermelstein. [9] До этого основными характеристиками для распознавания речи были линейные коэффициенты предсказания и линейные кепстральные коэффициенты предсказания.

Шкала Мел (рисунок 7) соотносит воспринимаемую частоту или высоту чистого тона (мел) с фактической измеренной частотой (Гц). Люди гораздо лучше различают небольшие изменения высоты звука на низких частотах, чем на высоких. Эта зависимость не совсем линейная и описывается следующей формулой:

$$M(f) = 1127,01048 \ln(1 + f/700)$$

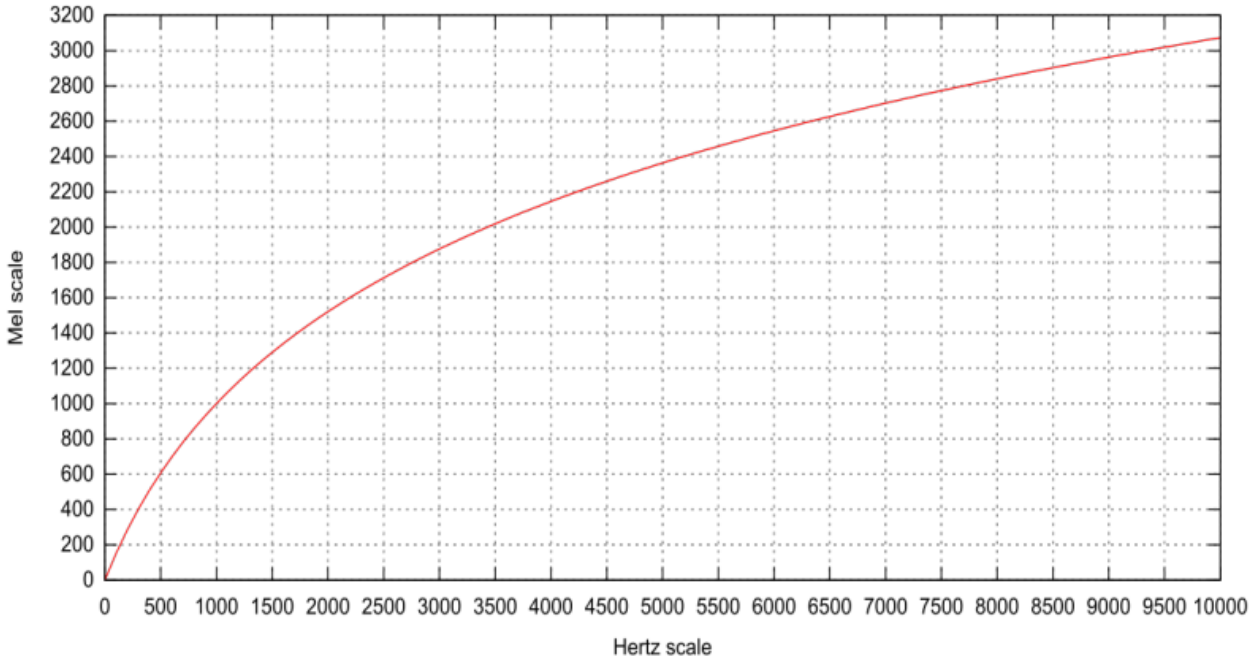


Рисунок 7 - График зависимости частоты от мел

Обратное преобразование из мел-шкалы в частоту выглядит следующим образом:

$$M^{-1}(m) = 700(e^{m/1127{,}01048} - 1)$$

Вычисление мел-частотных кепстральных коэффициентов включает в себя следующие шаги:

1. Необходимо разделить исходный сигнал на фреймы. Размер фрейма обычно выбирается от 20 до 40 мс, так как считается, что речевой сигнал на этом промежутке не сильно меняется. Следующие шаги применяются для каждого отдельного фрейма.
2. Речевой сигнал конечен и не является периодическим, поэтому из-за разрывов на его концах при применении преобразования Фурье проявляется эффект утечки. Для того, чтобы снизить его влияние на результат, каждый кадр умножается на оконную функцию Хемминга.
3. Вычисляем периодограмму для каждого фрейма.

4. Вычисляем блок мел-фильтров. Для этого треугольные фильтры (от 20 до 40) умножаются на периодограмму и суммируются. В результате мы получим энергии набора фильтров.

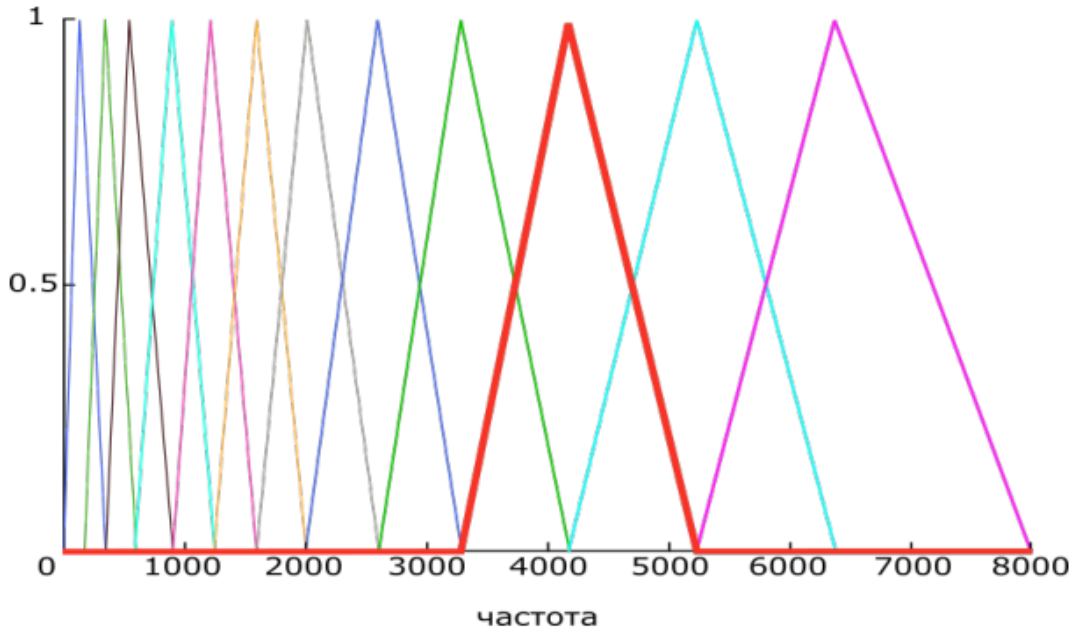


Рисунок 8 - Фильтры собираются в области низких частот, обеспечивая более высокое "разрешение" там, где это необходимо для распознавания

5. Полученные энергии логарифмируются. Это также мотивируется человеческим слухом: мы не слышим громкость в линейном масштабе. Обычно, чтобы удвоить воспринимаемую громкость звука, нам нужно затратить в 8 раз больше энергии. Это означает, что большие колебания энергии могут звучать не так уж и по-другому, если звук с самого начала громкий. Эта операция сжатия делает наши функции более близкими к тому, что на самом деле слышат люди.
6. Далее, используя дискретное косинусное преобразование, получим мел-кепстральные коэффициенты.

Разработка классификатора пола человека по речевым сигналам

Далее будет приведен код jupyter-ноутбука, реализующий классификацию пола человека по записям голоса. Для загрузки, первичного анализа голосовых записей, построения осциллограммы и спектрограммы будут использоваться библиотеки *librosa* и *numpy*. Для выходного классификатора взята свёрточная нейронная сеть, написанная с помощью *pytorch*. Для визуализации графиков использован *matplotlib*, метрики оценки взяты из *sklearn*.

В качестве данных для обучения будет использован *TIMIT* - корпус фонематически и лексически транскрибированной речи носителей американского английского языка разного пола и диалектов.

Для оценки качества полученного классификатора будут использованы следующие метрики: Accuracy и BCELoss (бинарная перекрестная энтропия).

```
!pip3 install timit-utils==0.9.0
```

```
!pip3 install torchaudio==0.11.0
```

```
! wget https://ndownloader.figshare.com/files/10256148
```

```
!unzip -q 10256148
```

```
import timit_utils as tu
import os
import librosa
import numpy as np
from tqdm import tqdm

import torch
import torch.nn as nn
from torch.optim import Adam
import torch.nn.functional as F

import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

import IPython
_TIMIT_PATH = 'data/lisa/data/timit/raw/TIMIT'
```

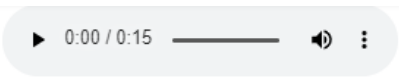
```
! wget https://audio-previews.elements.envatousercontent.com/files/6319559/preview.mp3 -O sample_f.mp3
! wget https://audio-previews.elements.envatousercontent.com/files/256324900/preview.mp3 -O sample_m.mp3
```

```
import librosa
data, sr = librosa.load('sample_f.mp3', 16000)
```

```
data.shape
```

```
(24715,)
```

```
import IPython
IPython.display.Audio(data, rate=1600)
```



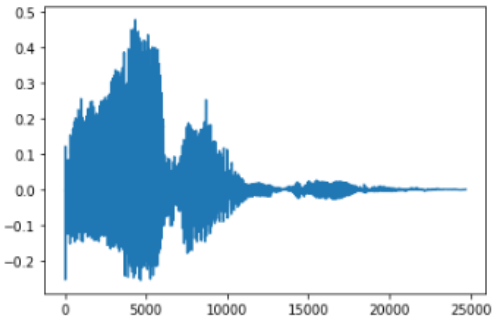

```
1  data.shape
```

```
(24715,)
```

```
1  import librosa
2  data, sr = librosa.load('sample_f.mp3', 16000)
```

```
1  import matplotlib.pyplot as plt
2  plt.plot(data)
```

```
[<matplotlib.lines.Line2D at 0x7f508b6a2d90>]
```




```
1  import soundfile as sf
```

```
1  #data, sr = librosa.load(...)
```

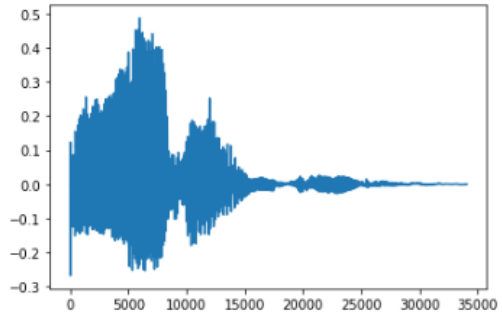
```
1  #_sr, _data = sf.read('tmp.wav')
```

```
1  amplitudes, sample_rate = librosa.load('tmp.wav')
2  print(f"{len(amplitudes)} points, {len(amplitudes) / sample_rate} sec, sr {sample_rate}")
```

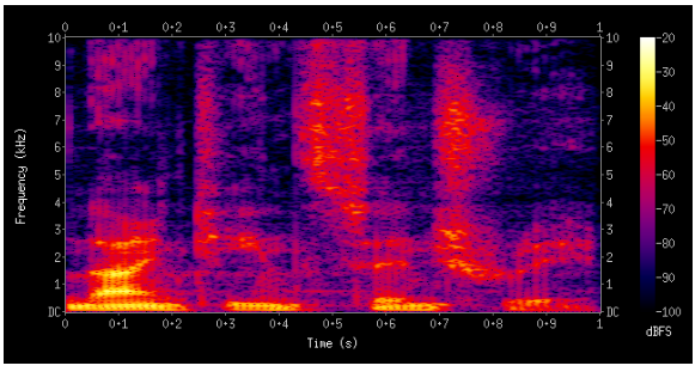
```
34061 points, 1.5447165532879819 sec, sr 22050
```

```
1  plt.plot(amplitudes)
```

```
[<matplotlib.lines.Line2D at 0x7f5089a37310>]
```



В полутора секундах более 34000 точек! Работать неудобно, надо конвертировать в формат, с которым умеем работать (в изображение). Используем **оконное преобразование Фурье** чтобы понять, какие же именно компоненты преобладают в некоторый момент времени.



Не забудем, что перед преобразование Фурье мы используем к фреймам по отдельности и потом склеиваем все в единую картину. Также следует помнить, что если использовать просто спектрограмму без перехода в мел-пространство, то мы можем сломать нашу будущую сетку слишком большими значениями.

Алгоритм построения мел-спектрограммы по набору частот:

1. Разделяем вход на пересекающиеся фреймы
2. Применяем к каждому преобразование Фурье
3. Переводим полученную спектрограмму в мел-пространство

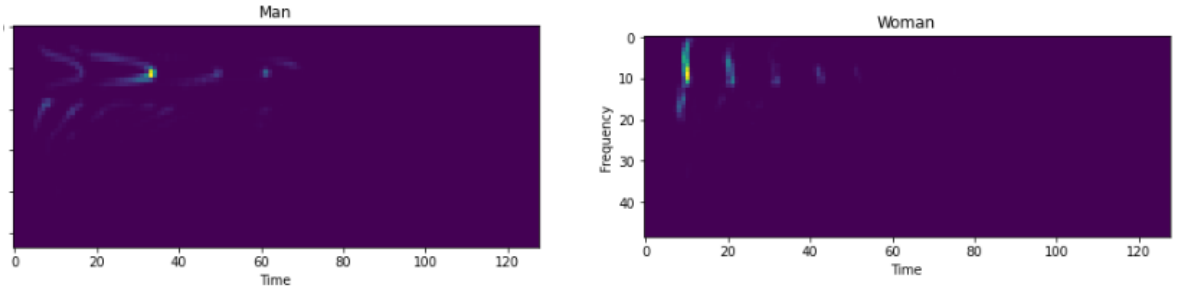
```
def slice_into_frames(amplitudes, window_length, hop_length):
    return librosa.core.spectrum.util.frame(
        np.pad(amplitudes, int(window_length // 2), mode='reflect'),
        frame_length=window_length, hop_length=hop_length)
    # выход: [window_length, num_windows]
```

```
def get_STFT(amplitudes, window_length, hop_length):
    """ Compute short-time Fourier Transform """
    # разбиваем амплитуды на пересекающиеся фреймы [window_length, num_frames]
    frames = slice_into_frames(amplitudes, window_length, hop_length)

    # получаем веса для Фурье, float[window_length]
    fft_weights = librosa.core.spectrum.get_window('hann', window_length, fftbins=True)

    # применяем преобразование Фурье
    stft = np.fft.rfft(frames * fft_weights[:, None], axis=0)
    return stft
```

```
def get_melspectrogram(amplitudes, sample_rate=22050, n_mels=128,
                       window_length=2048, hop_length=512, fmin=1, fmax=8192):
    """
    Implement mel-spectrogram as described above.
    :param amplitudes: float [num_amplitudes]
    :param sample rate: число отсчетов каждую секунду
    :param n_mels: число каналов спектрограммы
    :param window_length: параметр размера окна для Фурье
    :param hop_length: размер пересечения
    :param f_min: мин частота
    :param f_max: макс частота
    :returns: мел-scaled спектрограмма [n_mels, duration]
    """
    # Шаг 1
    stft = get_STFT(amplitudes, window_length, hop_length)
    assert stft.shape == (window_length // 2 + 1, len(amplitudes) // 512 + 1)

    # Шаг 2
    spectrogram = np.abs(stft ** 2)

    # Шаг 3
    mel_basis = librosa.filters.mel(sample_rate, n_fft=window_length,
                                    n_mels=n_mels, fmin=fmin, fmax=fmax)
    # ^-- matrix [n_mels, window_length / 2 + 1]

    mel_spectrogram = np.dot(mel_basis, spectrogram)
    assert mel_spectrogram.shape == (n_mels, len(amplitudes) // 512 + 1)

    return mel_spectrogram
```

```
amplitudes1, _  = librosa.load('sample_m.mp3', 16000)
amplitudes2, _  = librosa.load('sample_f.mp3',16000)
```

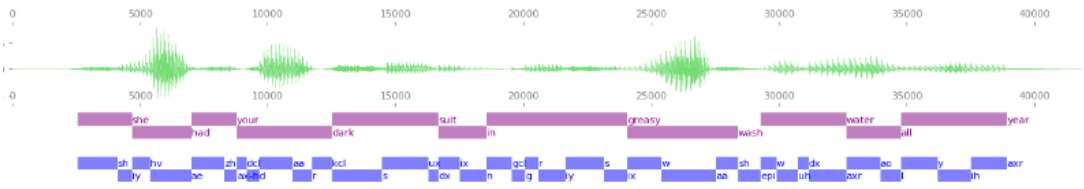
```
plt.figure(figsize=[16, 4])
plt.subplot(1, 2, 1)
plt.title("Man"); plt.xlabel("Time"); plt.ylabel("Frequency")
plt.imshow(get_melspectrogram(amplitudes1).transpose())

plt.subplot(1, 2, 2)
plt.title("Woman"); plt.xlabel("Time"); plt.ylabel("Frequency")
plt.imshow(get_melspectrogram(amplitudes2).transpose());
```




```
ref1 = librosa.feature.melspectrogram(amplitudes2, sr=sample_rate, n_mels=128, fmin=1, fmax=8192)
assert np.allclose(get_melspectrogram(amplitudes2), ref1, rtol=1e-4, atol=1e-4)
```

```
import torchaudio
amplitudes, sr = torchaudio.load('sample_f.mp3')
ref2 = torchaudio.transforms.MelSpectrogram(sample_rate=sample_rate, f_min=1, n_mels=128, f_max=8192)(amplitudes)
```

```
%matplotlib inline
import timit_utils as tu
import timit_utils.audio_utils as au
import timit_utils.drawing_utils as du

DATA_PATH = 'data/lisa/data/timit/raw/TIMIT'
corpus = tu.Corpus(DATA_PATH)
sentence = corpus.train.sentences_by_phone_df('aa').sentence[0]
du.DrawVerticalPanels([du.AudioPanel(sentence.raw_audio, show_x_axis=True),
                       du.WordsPanel(sentence.words_df, sentence.raw_audio.shape[0], show_x_axis=True),
                       du.PhonesPanel(sentence.phones_df, sentence.raw_audio.shape[0])
                      ])
```




```
class timit_dataloader:
    def __init__(self, data_path=_TIMIT_PATH, train_mode=True, age_mode=False):
        self.doc_file_path = os.path.join(data_path, 'DOC', 'SPKRINFO.TXT')
        self.corpus = tu.Corpus(data_path)
        with open(self.doc_file_path) as f:
            self.id_sex_dict = dict([(tmp.split(' ')[0], tmp.split(' ')[2]) for tmp in f.readlines()[39:]])
        with open(self.doc_file_path) as f:
            self.id_age_dict = dict(
                [(tmp.split(' ')[0], 86 - int(tmp.split('  ')[5].split('/')[-1].replace('??', '50'))) \
                 for tmp in f.readlines()[39:]])
        # print(self.id_age_dict)
        if train_mode:
            self.trainset = self.create_dataset('train', age_mode=age_mode)
            self.validset = self.create_dataset('valid', age_mode=age_mode)
        self.testset = self.create_dataset('test', age_mode=age_mode)
```

```
def return_sex(self, id):
    return self.id_sex_dict[id]

def return_age(self, id):
    return self.id_age_dict[id]

def return_data(self):
    return self.trainset, self.validset, self.testset

def return_test(self):
    return self.testset

def create_dataset(self, mode, age_mode=False):
    global people
    assert mode in ['train', 'valid', 'test']
    if mode == 'train':
        people = [self.corpus.train.person_by_index(i) for i in range(350)]
    if mode == 'valid':
        people = [self.corpus.train.person_by_index(i) for i in range(350, 400)]
    if mode == 'test':
        people = [self.corpus.test.person_by_index(i) for i in range(150)]
    spectrograms_and_targets = []
    if age_mode:
        for person in tqdm(people):
            try:
                target = self.return_age(person.name)
                for i in range(len(person.sentences)):
                    spectrograms_and_targets.append(
                        self.preprocess_sample(person.sentence_by_index(i).raw_audio, target, age_mode=True))
            except:
                print(person.name, target)
    else:
        for person in tqdm(people):
            target = self.return_sex(person.name)
            for i in range(len(person.sentences)):
                spectrograms_and_targets.append(
                    self.preprocess_sample(person.sentence_by_index(i).raw_audio, target))

    X, y = map(np.stack, zip(*spectrograms_and_targets))
    X = X.transpose([0, 2, 1])  # to [batch, time, channels]
    return X, y
```

```
@staticmethod
def spec_to_image(spec, eps=1e-6):
    mean = spec.mean()
    std = spec.std()
    spec_norm = (spec - mean) / (std + eps)
    spec_min, spec_max = spec_norm.min(), spec_norm.max()
    spec_scaled = 255 * (spec_norm - spec_min) / (spec_max - spec_min)
    spec_scaled = spec_scaled.astype(np.uint8)
    return spec_scaled

@staticmethod
def clasterize_by_age(age):
    if age < 25:
        return 0
    if 25 < age < 40:
        return 0.5
    if age > 40:
        return 1

def preprocess_sample(self, amplitudes, target, age_mode=False, sr=16000, max_length=150):
    spectrogram = librosa.feature.melspectrogram(amplitudes, sr=sr, n_mels=128, fmin=1, fmax=8192)[:, :max_length
    spectrogram = np.pad(spectrogram, [[0, 0], [0, max(0, max_length - spectrogram.shape[1])]], mode='constant')
    if age_mode:
        # target = self.clasterize_by_age(target)
        target = target/80
    else:
        target = 0 if target == 'F' else 1
    # print(np.array(self.spec_to_image(np.float32(spectrogram))).shape)
    return self.spec_to_image(np.float32(spectrogram)), target

def preprocess_sample_inference(self, amplitudes, sr=16000, max_length=150, device='cpu'):
    spectrogram = librosa.feature.melspectrogram(amplitudes, sr=sr, n_mels=128, fmin=1, fmax=8192)[:, :max_length
    spectrogram = np.pad(spectrogram, [[0, 0], [0, max(0, max_length - spectrogram.shape[1])]], mode='constant')
    spectrogram = np.array([self.spec_to_image(np.float32(spectrogram))]).transpose([0, 2, 1])

    return t.tensor(spectrogram, dtype=t.float).to(device, non_blocking=True)
```

```
class dataloader:
    def __init__(self, spectrograms, targets):
        self.data = list(zip(spectrograms, targets))

    def next_batch(self, batch_size, device):
        indices = np.random.randint(len(self.data), size=batch_size)

        input = [self.data[i] for i in indices]

        source = [line[0] for line in input]
        target = [line[1] for line in input]

        return self.torch_batch(source, target, device)

    @staticmethod
    def torch_batch(source, target, device):
        return tuple(
            [
                t.tensor(val, dtype=t.float).to(device, non_blocking=True)
                for val in [source, target]
            ]
        )
```

```
class Model(nn.Module):
    def __init__(self, window_sizes=(3, 4, 5)):
        super(Model, self).__init__()

        self.convs = nn.ModuleList([
            nn.Conv2d(1, 128, [window_size, 128], padding=(window_size - 1, 0))
            for window_size in window_sizes
        ])

        self.fc = nn.Linear(128 * len(window_sizes), 1)
```

```
def forward(self, x):
    # Apply a convolution + max pool layer for each window size
    x = torch.unsqueeze(x, 1)  # [B, C, T, F] Add a channel dim.
    xs = []
    for conv in self.convs:
        x2 = F.relu(conv(x))  # [B, F, T, 1]
        x2 = torch.squeeze(x2, -1)  # [B, F, T]
        x2 = F.max_pool1d(x2, x2.size(2))  # [B, F, 1]
        xs.append(x2)
    x = torch.cat(xs, 2)  # [B, F, window]

    # FC
    x = x.view(x.size(0), -1)  # [B, F * window]
    logits = self.fc(x)  # [B, class]
    probs = torch.sigmoid(logits).view(-1)
    return probs

def loss(self, probs, targets):
    return nn.BCELoss()(probs.float(), targets.float())
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'using {device} mode')
patience = 500
best_loss = 1000
cnt = 0
```

ıg cuda mode

```
model = Model()
if device == torch.device('cuda'):
    model.cuda()
else:
    model.cpu()
model.train()
```

```
Model(
  (convs): ModuleList(
    (0): Conv2d(1, 128, kernel_size=(3, 128), stride=(1, 1), padding=(2, 0))
    (1): Conv2d(1, 128, kernel_size=(4, 128), stride=(1, 1), padding=(3, 0))
    (2): Conv2d(1, 128, kernel_size=(5, 128), stride=(1, 1), padding=(4, 0))
  )
  (fc): Linear(in_features=384, out_features=1, bias=True)
)
```

```
!unzip 1025
```

```
_timit_dataloader = timit_dataloader()
train, valid, test = _timit_dataloader.return_data()

trainset = dataloader(*train)
validset = dataloader(*valid)
testset = dataloader(*test)
BATCH_SIZE = 64

optimizer = Adam(
    [p for p in model.parameters() if p.requires_grad], betas=(0.9, 0.999), eps=1e-5
)
```

```
100%|██████████| 350/350 [00:44<00:00,  7.94it/s]
100%|██████████| 50/50 [00:06<00:00,  8.27it/s]
100%|██████████| 150/150 [00:18<00:00,  8.33it/s]
```

```
import torch as t
from tqdm import tqdm_notebook
for i in tqdm_notebook(range(1000)):

    optimizer.zero_grad()

    input, target = trainset.next_batch(BATCH_SIZE, device=device)
    out = model(input)
    loss = model.loss(out, target)
    optimizer.step()
```

```
    if i % 50 == 0:
        model.eval()

        with torch.no_grad():
            optimizer.zero_grad()

            input, target = validset.next_batch(BATCH_SIZE, device=device)
            out = model(input)
            valid_loss = model.loss(out, target)
            out, target = out.cpu().detach().numpy(), target.cpu().detach().numpy()
            # print(out, target)
            out = [1. if tmp > 0.5 else 0 for tmp in out]
            print(f'accuracy_score:{accuracy_score(out, target)}')
            print("i {}, valid {}".format(i, valid_loss.item()))
            print("_________")

        model.train()

    if i % 50 == 0 and best_loss > valid_loss.item():
        best_loss = valid_loss.item()
        cnt = 0
    else:
        cnt += 1

    if cnt > patience:
        break
print('training finished')
```

```
_________
accuracy_score:0.625
i 700, valid 2.0543198585510254
_________
accuracy_score:0.625
i 750, valid 1.9029508829116821
_________
accuracy_score:0.671875
i 800, valid 1.6124660968780518
_________
accuracy_score:0.65625
i 850, valid 1.8747615814208984
_________
accuracy_score:0.734375
i 900, valid 1.095263123512268
_________
accuracy_score:0.71875
i 950, valid 1.5927929878234863
_________
training finished
```

```
# Code for recording audio from the browser
from IPython.display import Javascript
from google.colab import output
from base64 import b64decode
import IPython
import uuid
from google.colab import output


class InvokeButton(object):
  def __init__(self, title, callback):
    self._title = title
    self._callback = callback

  def _repr_html_(self):
    from google.colab import output
    callback_id = 'button-' + str(uuid.uuid4())
    output.register_callback(callback_id, self._callback)

    template = """<button id="{callback_id}" style="cursor:pointer;background-color:#EEEEEE;border-color:#E0E0E0;pad
        <script>
          document.querySelector("#{callback_id}").onclick = (e) => {{
            google.colab.kernel.invokeFunction('{callback_id}', [], {{}})
            e.preventDefault();
          }};
        </script>"""
    html = template.format(title=self._title, callback_id=callback_id)
    return html

RECORD = """
const sleep  = time => new Promise(resolve => setTimeout(resolve, time))
const b2text = blob => new Promise(resolve => {
  const reader = new FileReader()
  reader.onloadend = e => resolve(e.srcElement.result)
  reader.readAsDataURL(blob)
})
var record = time => new Promise(async resolve => {
  stream = await navigator.mediaDevices.getUserMedia({ audio: true })
  recorder = new MediaRecorder(stream)
  chunks = []
  recorder.ondataavailable = e => chunks.push(e.data)
  recorder.start()
  await sleep(time)
  recorder.onstop = async ()=>{
    blob = new Blob(chunks)
    text = await b2text(blob)
    resolve(text)
  }
  recorder.stop()
 })
 """
```

```
def record(sec=3):
  display(Javascript(RECORD))
  s = output.eval_js('record(%d)' % (sec*1000))
  b = b64decode(s.split(',')[1])
  with open('audio.wav','wb+') as f:
    f.write(b)
  return 'audio.wav'
```

```
model.eval()

def predict(wavfile):
    waveform, _ = librosa.load(wavfile, sr=16000)

    input = _timit_dataloader.preprocess_sample_inference(waveform)
    with torch.no_grad():
        out = model(torch.tensor(input, dtype=torch.float).to(device))
        out = out.cpu().detach().numpy()
    print(out)
    out = 'female' if out < 0.5 else 'male'
    return out
```

```
predict('tmp.wav')
```

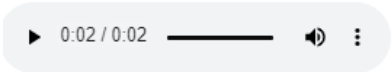
```
[0.9705649]
<ipython-input-47-ca83bda2d51f>:8: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clo
  out = model(torch.tensor(input, dtype=torch.float).to(device))
'male'
```

```
def classify():
  print("Now recording for 3 seconds, say what you will...")
  record()
  os.system('ffmpeg -i audio.wav -ar 16000 -y audio.wav')
  print(f"Audio recording complete, guess it is {predict('audio.wav')}")

InvokeButton('Start recording', classify)
```

Start recording

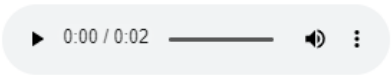
```
Now recording for 3 seconds, say what you will...
[0.96050864]
Audio recording complete, guess it is male
<ipython-input-47-ca83bda2d51f>:8: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clor
  out = model(torch.tensor(input, dtype=torch.float).to(device))
```

```
IPython.display.Audio('audio.wav')
```




```
def spec_to_image(spec, eps=1e-6):
    mean = spec.mean()
    std = spec.std()
    spec_norm = (spec - mean) / (std + eps)
    spec_min, spec_max = spec_norm.min(), spec_norm.max()
    spec_scaled = 255 * (spec_norm - spec_min) / (spec_max - spec_min)
    spec_scaled = spec_scaled.astype(np.uint8)
    return spec_scaled

def preprocess_sample_inference(amplitudes, sr=16000, max_length=150, device='cuda'):
    spectrogram = librosa.feature.melspectrogram(amplitudes, sr=sr, n_mels=128, fmin=1, fmax=8192)[:, :max_length]
    spectrogram = np.pad(spectrogram, [[0, 0], [0, max(0, max_length - spectrogram.shape[1])]], mode='constant')
    spectrogram = np.array([spec_to_image(np.float32(spectrogram))]).transpose([0, 2, 1])

    return torch.tensor(spectrogram, dtype=torch.float).to(device, non_blocking=True)
def predict(wavfile):
    waveform, _ = librosa.load(wavfile, sr=16000)

    input = preprocess_sample_inference(waveform)
    with torch.no_grad():
        out = model(input).cpu().detach().numpy()
    out = 1- out
    print(out)
    out = 'female' if out > 0.5 else 'male'
    return out
```

```
IPython.display.Audio('audio.wav')
```




```
predict('audio.wav')
```

```
[0.03949136]
'male'
```

Выводы

В данной работе были рассмотрены назначение и структура WAV-файлов, а также описание осциллограммы и спектрограммы аудиосигналов, исследован принцип работы преобразования Фурье и его применимость к предобработке звуковых данных, изучена мел-шкала и способ получения наиболее эффективных аудиопризнаков – мел-кепстральных коэффициентов, разработан модуль на основе технологии нейронных сетей, реализующий классификацию пола человека по записи его голоса.

Результаты практической части показывают, что использование преобразования Фурье для предобработки аудиосигналов и мел-кепстральных коэффициентов для извлечения признаков из звуковых данных обеспечивают высокое качество разработанного нейросетевого модуля-классификатора для решения задачи классификации пола человека по записи его голоса. Данные методы также могут быть эффективно применены для решения иных задач в области обработки естественного языка.

Список использованных источников

1. MP3, AAC, WAV, FLAC: рассказываем обо всех форматах аудиофайлов [Электронный ресурс]. – Электрон. дан. – URL: https://www.audiomania.ru/content/art-7314.html
2. Wave File Format - формат звукового файла WAV [Электронный ресурс]. – Электрон. дан. – URL: http://microsin.net/programming/pc/wav-format.html
3. Формат звуковых файлов WAV [Электронный ресурс]. – Электрон. дан. – URL: https://radioprog.ru/post/1025
4. Структура WAV файла [Электронный ресурс]. – Электрон. дан. – URL: https://audiocoding.ru/articles/2008-05-22-wav-file-structure/
5. Столбов М.Б., Основы анализа и обработки речевых сигналов – СПб.: НИУ ИТМО, 2021 – 101 с.
6. Рисуем звук [Электронный ресурс]. – Электрон. дан. – URL: https://arman-engineering.ru/risuem-zvuk-chast-1/
7. Практическое применение преобразования Фурье для анализа сигналов [Электронный ресурс]. – Электрон. дан. – URL: https://itnan.ru/post.php?c=1&p=269991
8. Что такое преобразование Фурье? [Электронный ресурс]. – Электрон. дан. – URL: https://radioprog.ru/post/877
9. S. Davis and P. Mermelstein Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. In IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 28 No. 4, 1980