УНИВЕРСИТЕТ ИТМО

В.А. Волохов, Г.М. Лаврентьева,
С.А. Новосёлов, Ю.Н. Матвеев

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО КУРСУ «РАСПОЗНАВАНИЕ ДИКТОРА»

Санкт-Петербург

2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

**В.А. Волохов, Г.М. Лаврентьева,
С.А. Новосёлов, Ю.Н. Матвеев**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО КУРСУ «РАСПОЗНАВАНИЕ ДИКТОРА»

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 09.04.02 «Информационные системы и технологии» в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ высшего образования корпоративной магистратуры

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург

2022

Волохов В.А., Лаврентьева Г.М., Новосёлов С.А., Матвеев Ю.Н. Методические указания к выполнению лабораторных работ по курсу «Распознавание диктора». – СПб: Университет ИТМО, 2022. – 86 с.

Рецензент: Приоров Андрей Леонидович, доктор технических наук, доцент, профессор кафедры цифровых технологий и машинного обучения физического факультета ФГБОУ ВО «Ярославский государственный университет им. П.Г. Демидова».

Содержится набор лабораторных работ по ряду разделов курса «Распознавание диктора». Приводятся основные теоретические сведения, необходимые для выполнения каждой работы, содержание, порядок выполнения и требования для сдачи лабораторного задания, контрольные вопросы, списки рекомендуемой литературы, а также примеры программных кодов, написанных с использованием языка Python. Предназначено для студентов магистратуры, обучающихся по направлению 09.04.02 «Информационные системы и технологии» и осваивающих образовательную программу по профилю подготовки «Речевые технологии и машинное обучение», а также научных руководителей магистрантов.

ВВЕДЕНИЕ

Настоящее учебно-методическое пособие содержит описания для выполнения лабораторных работ по курсу «Распознавание диктора» для студентов вузов, обучающихся по направлениям подготовки / специальностям, связанным с обработкой речевых сигналов.

Голос человека является биометрической модальностью, содержащей индивидуальные особенности диктора, которые можно использовать для его распознавания (верификации / идентификации). Необходимо отметить, что распознавание диктора по голосу и распознавание речи – это разные задачи! Распознавание речи направлено на распознавание слов в речевом сигнале, преобразование речевого сигнала в текст, что не является биометрией. Процесс распознавания диктора основан на использовании признаков, которые подвержены влиянию как физической структуры голосового тракта личности, так и её поведенческих характеристик.

Распознавание диктора по голосу предполагает под собой решение двух основных задач: получение биометрических образцов голоса от конечных пользователей и дальнейшее использование этих биометрических образцов. При этом можно выделить следующие режимы работы системы распознавания диктора:

1. Верификация (аутентификация) и идентификация;
2. Распознавание на закрытом и открытом множествах;
3. Текстозависимое и текстонезависимое распознавание.

В случае *верификации* диктор выдает себя за определенную личность, установление идентичности в данном случае выполняется с помощью оценки схожести между голосами в двух фонограммах (звукозаписях), что определяет этот режим сравнения как «один к одному». Выходом при этом является ответ «да» или «нет» об аутентичности голосов в двух фонограммах. В случае *идентификации* требуется отнести неизвестного диктора к одному из известных. Сравнение выполняется по принципу «один ко многим», а выходом является метка известного диктора, к которому отнесен неизвестный.

Когда все дикторы внутри заданного множества фонограмм являются известными, говорят о *распознавании на закрытом множестве*. Альтернативно, если анализируемая тестовая фонограмма связана с диктором, который не принадлежит заранее определенной группе дикторов, говорят о *распознавании на открытом множестве*.

В случае, когда диктору требуется произнести некоторую заранее определенную фразу, необходимую для принятия решения биометрической системой, говорят о *текстозависимом распознавания*, иначе – о *текстонезависимом*.

Общая схема (конвейер) системы голосовой биометрии представлена на рисунке 1. На вход системы подается некоторая фонограмма, которая

претерпевает предварительную обработку в виде шумоочистки, определения областей речевой активности, выделения акустических признаков, например, мел-частотных кепстральных коэффициентов и т.п. В случае необходимости в общий конвейер может быть включен блок диаризации (не рассматривается в рамках настоящего пособия), позволяющий разделить речевой сигнал на сегменты, в каждом из которых присутствует голос только одного диктора. При этом сегменты, связанные с одним и тем же диктором, маркируются на выходе блока диаризации общей меткой. Блок построения дикторской модели выполняет формирование некоторого дескриптора (эмбеддинга), привязанного к сегментам речи конкретного диктора анализируемой звукозаписи. Сформированные дикторские модели сравниваются с некоторым эталоном / некоторыми эталонами моделей дикторов, сохраненными в определённой базе данных. На основе результатов сравнений биометрическая система принимает итоговое решение.

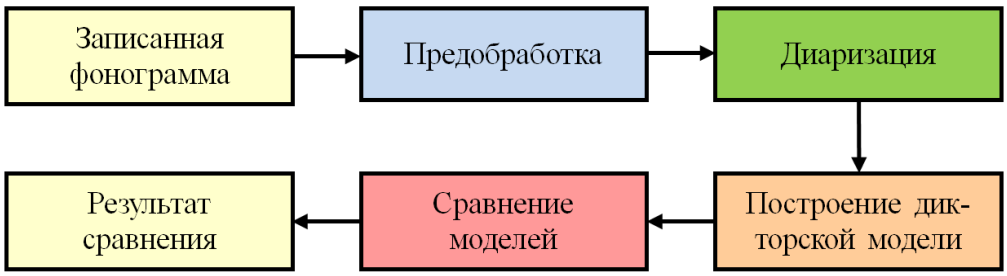


Рисунок 1 – Общая схема системы голосовой биометрии

Учебно-методическое пособие включает в себя пять лабораторных работ, охватывающих основные темы курса «Распознавание диктора». Студент найдет здесь теоретические сведения, необходимые для квалифицированного выполнения каждой работы, содержание, порядок выполнения и требования для сдачи лабораторного задания, контрольные вопросы, списки рекомендуемой литературы, а также примеры программных кодов. При выполнении лабораторных работ предполагается реализация алгоритмов на языке Python 3 для решения различных этапов конвейера системы распознавания диктора. Итогом лабораторного практикума является построенная базовая система распознавания диктора.

Лабораторные занятия проводятся с использованием цифровых вычислительных устройств, содержащих графические процессоры, и затрагивают следующие теоретические разделы:

1. Предобработка речевых сигналов и извлечение информативных признаков (кратковременные энергии мел-частотных полос и мел-частотные кепстральные коэффициенты);
2. Построение и тестирование энергетического детектора речевой активности на основе модели гауссовой смеси;
3. Построение и тестирование блока генерации дикторских моделей, основанного на использовании глубоких ResNet-подобных

нейросетевых архитектур в задаче идентификации диктора на закрытом множестве;

4. Использование критериев принятия решения для автоматического подбора порога принятия решения биометрической системы в задаче верификации диктора на открытом множестве;
5. Применение адаптации на основе s-нормализации и линейной модели калибровки для построения биометрических систем, используемых на практике в задаче верификации диктора на открытом множестве.

Ознакомиться с заданиями для выполнения лабораторных работ можно, скачав GitHub-репозиторий лабораторного практикума по ссылке: https://github.com/itmo-mbss-lab/sr_labs_book.

Задания для выполнения конкретной лабораторной работы находятся в соответствующем директории (**./lab1**, **./lab2** и т.д.) указанного репозитория. Каждый директорий с материалами для выполнения лабораторной работы содержит следующие файлы:

1. Файл **./labN/labN_blank.ipynb** – файл Jupiter Notebook, содержащий теоретическую часть и более конкретные комментарии, что и как должно выполняться и какой функционал должен за это отвечать. Здесь **N** – это номер лабораторной работы;
2. Файл **./labN/exercises_blank.py** представляет Python-файл, содержащий заготовки программных кодов, которые необходимо завершить самостоятельно в ходе выполнения лабораторной работы для запуска всего лабораторного проекта целиком.

Директорий **./common**, находящийся внутри репозитория, содержит общий функционал, который может быть импортирован при выполнении отдельных лабораторных работ.

Учебно-методическое пособие обобщает многолетний опыт авторов и их коллег по разработке систем голосовой биометрии в ходе практической деятельности в Группе компаний ЦРТ, а также преподавания курса «Распознавание диктора» студентам факультета информационных технологий и программирования (ФИТиП) Университета ИТМО. Материал может быть использован при решении задач курсового и дипломного проектирования, а также для выполнения студенческих научных работ.

Благодарности. Выражаем благодарность нашим коллегам за помощь при создании этого пособия. Мы благодарны Пеховскому Т.С., Шулипе А.К., Козлову А.В., Румянцеву Д.А., Котову Т.О., которые оказали значительное влияние на формирование взглядов авторов в данном научном направлении. Эти взгляды формировались также в совместной работе с нашими коллегами Гусевым А.Е., Авдеевой А.С., Виноградовой А.Р., Корсуновым И.С., Волковой М.В., Аусевым Е.В., Байкаловым Р.А., Дарьян В.О., Логуновым А.А., Рязановым М.С. Всем им авторы очень признательны за совместную научно-исследовательскую работу. Отдельно хочется поблагодарить Пояркову Н.В. за создание иллюстрации к обложке пособия.

Лабораторная работа № 1

ИНФОРМАТИВНЫЕ ПРИЗНАКИ РЕЧЕВЫХ СИГНАЛОВ: ИЗВЛЕЧЕНИЕ ПРИЗНАКОВ

Цель работы: изучение процедуры построения информативных акустических признаков для речевых сигналов.

Краткое описание: в рамках настоящей лабораторной работы требуется познакомиться с процедурами предобработки речевых сигналов и извлечения информативных признаков. В работе предлагается научиться извлекать кратковременные энергии мел-частотных полос и мел-частотные кепстральные коэффициенты.

Данные: в качестве данных для выполнения лабораторной работы предлагается использовать тестовую часть базы VoxCeleb1.

Содержание лабораторной работы

1. Подготовка данных для анализа.
2. Выполнение процедуры преэмфазиса.
3. Вычисление акустических признаков разных видов.
4. Выполнение локальных центрирования и масштабирования акустических признаков.
5. Построение распределений первых трёх компонент полученных акустических признаков.

Порядок выполнения и краткая теория

1. Подготовка данных для анализа

Перед выполнением лабораторной работы требуется импортировать некоторые Python-модули.

```
# Import of modules
import os
import sys

sys.path.append(os.path.realpath('..'))

from common import download_dataset, extract_dataset

from math import sqrt, pi
from scipy.fftpack import dct
```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import hist, plot, show, grid, title, \
xlabel, ylabel, legend, axis, imshow

from mpl_toolkits.axes_grid1 import make_axes_locatable

from skimage.morphology import opening, closing
from torchaudio.transforms import Resample
from multiprocessing import Pool
import torchaudio

from exercises_blank import split_meta_line, preemphasis, \
framing, power_spectrum
from exercises_blank import compute_fbank_filters, \
compute_fbanks_features, compute_mfcc, mvn_floating
```

В ходе выполнения лабораторной работы необходимы данные для осуществления процедуры вычисления акустических признаков. Возьмём в качестве этих данных несколько звукозаписей голосов людей мужского и женского пола, сохраненных в формат wav, выбранных из корпуса VoxCeleb1 test set [1]. Данный корпус содержит 4874 звукозаписи (частота дискретизации равна 16 кГц) 40 дикторов мужского и женского пола, разговаривающих преимущественно на английском языке.

В рамках настоящего пункта требуется выполнить загрузку и распаковку звуковых wav-файлов из корпуса VoxCeleb1 test set. Предположим, что здесь и далее в рамках настоящей работы рабочим директорием является **`./lab1`**, находящийся внутри репозитория с лабораторными работами, и все пути до директориев и файлов указаны относительно него. Загрузку и распаковку данных можно выполнить с использованием программного кода, представленного ниже.

```
# Download VoxCeleb1 test set
with open('../data/lists/datasets.txt', 'r') as f:
    lines = f.readlines()

download_dataset(lines, \
                 user='voxceleb1902', \
                 password='nx0bl2v2', \
                 save_path='../data')

# Extract VoxCeleb1 test set
extract_dataset(save_path='../data/voxceleb1_test', \
                fname='../data/vox1_test_wav.zip')
```

2. Выполнение процедуры преэмфазиса

В рамках настоящего пункта предлагается изучить процедуру предобработки речевых сигналов, получившую название преэмфазис [2–4].

Предыскажение или *преэмфазис* (pre-emphasis) – применение *фильтра верхних частот* [5–7] к сигналу до процедуры извлечения акустических признаков, для того чтобы компенсировать тот факт, что обычно вокализованная речь содержит на низких частотах намного больше энергии, чем невокализованная речь на высоких. Обозначим через $x(n)$ обрабатываемый сигнал, а через $y(n)$ результат обработки, тогда процедура преэмфазиса может быть описана с помощью следующего выражения:

$$y(n) = x(n) + \alpha x(n-1).$$

Выражение выше представляет собой *линейное разностное уравнение с постоянными коэффициентами* [5–7] и позволяет описать процедуру работы *нерекурсивного фильтра первого порядка с конечной импульсной характеристикой* [5–8]. Параметр α является единственным параметром рассматриваемого фильтра. В случае, когда параметр α является отрицательным, рассматриваемый фильтр обладает свойствами фильтра верхних частот. Для выполнения процедуры преэмфазиса обычно параметр $\alpha = -0{,}97$.

В рамках настоящего пункта требуется выполнить следующую последовательность действий:

1. Загрузить анализируемый речевой сигнал;
2. Выполнить процедуру преэмфазиса по отношению к загруженному речевому сигналу;
3. Сравнить спектрограммы речевого сигнала до и после процедуры преэмфазиса.

В папке **`./metadata/`** находится текстовый файл **`meta.txt`** со списком файлов, которые предлагается использовать в данной лабораторной работе. Рассматриваемый текстовый файл имеет достаточно простой формат строк, представленный ниже.

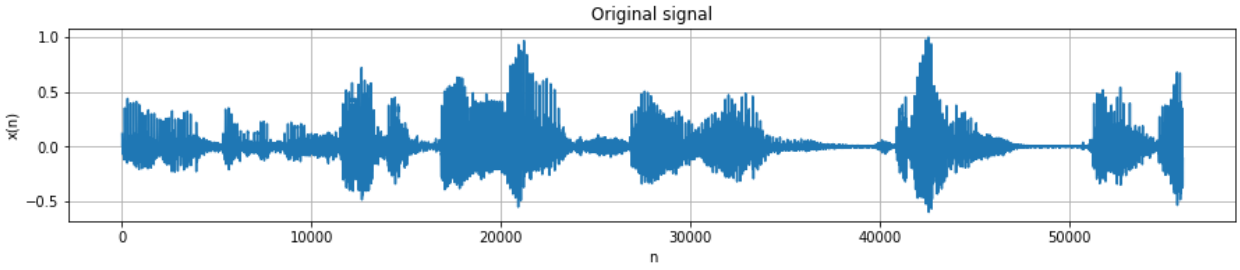
```
Speaker_ID Gender Path
```

Здесь `Speaker_ID` – это идентификационный номер диктора, `Gender` – пол диктора (мужской – `m` или женский – `f`), `Path` – путь до звукозаписи.

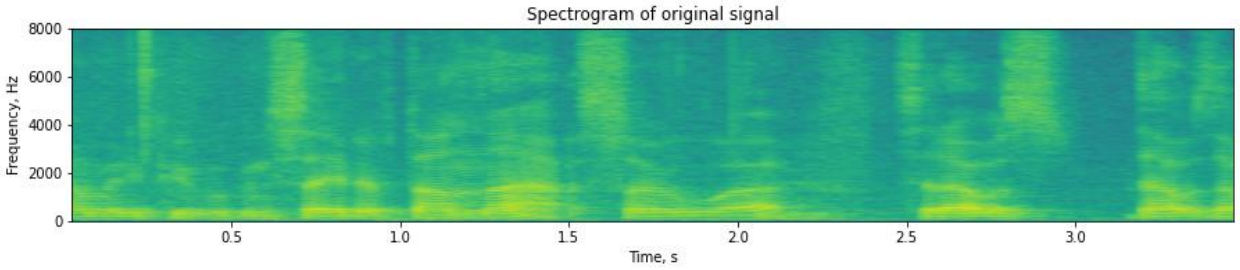
Для выполнения чтения файла **`./metadata/meta.txt`** и загрузки одной из звукозаписей, указанных в нём, можно воспользоваться программным кодом ниже. Для запуска программного кода необходимо завершить функцию `split_meta_line` (парсинг текстового файла) из **`./exercises_blank.py`**.

```
path_to_meta = './metadata/meta.txt'

p = Pool(1)
with open(path_to_meta, 'r') as f:
    list_lines = f.readlines()
```

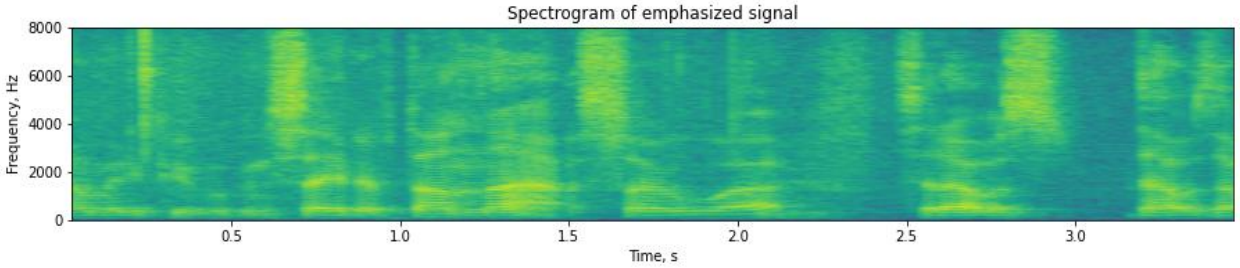
```
speaker_ids, genders, paths = zip(*p.map(split_meta_line, \
                                  list_lines[1:]))

path_to_wav = paths[0]

# Load signal
signal, sample_rate = torchaudio.load(path_to_wav)
signal = signal.numpy().squeeze(axis=0)
signal = signal/np.abs(signal).max()
```



а)



б)



в)

Рисунок 1 – Результаты выполнения программного кода, позволяющего выполнить сравнение спектрограмм речевого сигнала до и после процедуры преэмфазиса: а) осциллограмма исходного сигнала; б) спектрограмма исходного сигнала; в) спектрограмма после применения процедуры преэмфазиса к исходному сигналу

Для построения осциллограммы и спектрограммы первых 3,5 секунд загруженного речевого сигнала, а также спектрограммы сигнала после выполнения процедуры преэмфазиса можно воспользоваться кодом ниже. Для запуска программного кода необходимо завершить функцию `preemphasis` из файла **`./exercises_blank.py`**, позволяющую выполнить преэмфазис анализируемого сигнала.

```
signal = signal[0:int(3.5 * sample_rate)] # keep the first 3.5 s
```

```
emphasized_signal = preemphasis(signal)    # emphasized signal

plt.figure(figsize=(15, 10))
plt.subplots_adjust(wspace=0, hspace=0.5)

plot_a = plt.subplot(311)
plot_a.plot(signal)
plot_a.set_xlabel('n')
plot_a.set_ylabel('x(n)')
plot_a.title.set_text('Original signal')
plot_a.grid()

plot_b = plt.subplot(312)
plot_b.specgram(signal, NFFT=1024, Fs=sample_rate, noverlap=900)
plot_b.set_xlabel('Time, s')
plot_b.set_ylabel('Frequency, Hz')
plot_b.title.set_text('Spectrogram of original signal')

plot_c = plt.subplot(313)
plot_c.specgram(emphasized_signal, NFFT=1024, Fs=sample_rate,
noverlap=900)
plot_c.set_xlabel('Time, s')
plot_c.set_ylabel('Frequency, Hz')
plot_c.title.set_text('Spectrogram of emphasized signal')
plt.show()
```

Результаты запуска кода выше представлены на рисунке 1.

3. Вычисление акустических признаков разных видов

Работа современных систем распознавания диктора, как правило, основана на применении акустических признаков [2–4, 9, 10]. *Акустические признаки* являются формой представления речевого сигнала в частотно-временной области. Возможным примером акустических признаков является *спектрограмма сигнала*, которую можно вычислить с использованием *оконного преобразования Фурье*. Применение спектрограммы на практике не является выгодным, в частности, из-за её большого разрешения по частоте. Поэтому возможным вариантом для построения акустических признаков могут являться: *логарифмы энергий на выходе мел-банка фильтров* (MFBs, mel filter banks), *мел-частотные кепстральные коэффициенты* (MFCCs, mel frequency cepstral coeffitients) и т.п.

В рамках настоящего пункта предлагается вычислить логарифмы энергий на выходе мел-банка фильтров размерности 40 и мел-частотные кепстральные коэффициенты размерности 20. Для вычисления признаков можно воспользоваться следующей последовательностью действий:

1. Представить обрабатываемый речевой сигнал в виде набора фреймов (кадров) с использованием окна Хэмминга (размер окна можно выбрать равным 25 мс, а шаг окна – 10 мс);

2. Вычислить одномерный спектр Фурье и рассчитать на его основе спектр мощности по отношению к каждому из фреймов;
3. Рассчитать мел-банк фильтров;
4. Перемножить квадрат амплитудно-частотной характеристики (АЧХ) каждого фильтра со спектром мощности, который был вычислен для каждого фрейма, и просуммировать коэффициенты получившихся спектров, рассчитав энергии внутри соответствующих полос банка фильтров;
5. Вычислить логарифм от значений энергий на предыдущем шаге. На данном шаге формируется первый тип акустических признаков, то есть логарифмы энергий на выходе мел-банка фильтров;
6. Вычислить дискретное косинусное преобразование от логарифмов значений энергий. На этом этапе формируются мел-частотные кепстральные коэффициенты.

Подробное описание процедуры вычисления акустических признаков, рассматриваемых в рамках настоящего пункта, можно найти в научно-технической литературе [2–4, 9, 10].

Для выполнения заданий настоящего пункта необходимо завершить ряд функций из файла **`./exercises_blank.py`**: `framing` (разбиение сигнала на фреймы), `power_spectrum` (вычисление спектра мощности для сформированных фреймов), `compute_fbank_filters` (синтез мел-банка фильтров), `compute_fbanks_features` (вычисление логарифмов энергий на выходе мел-банка фильтров) и `compute_mfcc` (вычисление мел-частотных кепстральных коэффициентов).

Для разбиения сигнала на фреймы и вычисления спектра мощности для каждого из фреймов можно вызвать соответствующие функции так, как представлено ниже.

```
frames = framing(emphasized_signal) # framing of the emphasized
                                    # signal

pow_frames = power_spectrum(frames) # power spectrum of framed
                                    # signal
```

Для синтеза мел-банка фильтров и визуализации квадратов АЧХ его фильтров можно воспользоваться программным кодом ниже.

```
fbank = compute_fbank_filters(nfilt=40, \
                              sample_rate=16000, \
                              NFFT=512)

plt.figure(figsize=(15, 5))
plot_a = plt.subplot()
plt.subplots_adjust(wspace=0, hspace=1)

nfilt = fbank.shape[0]
```

```
for k in range(nfilt):
    plot_a.plot(fbank[k,:])
plot_a.set_xlabel('Frequency bins')
plot_a.set_ylabel('Amplitude')
plot_a.title.set_text('FBank')
plot_a.grid()
plt.show()
```

Визуализация рассчитанных квадратов АЧХ синтезированного мел-банка фильтров представлена на рисунке 2.

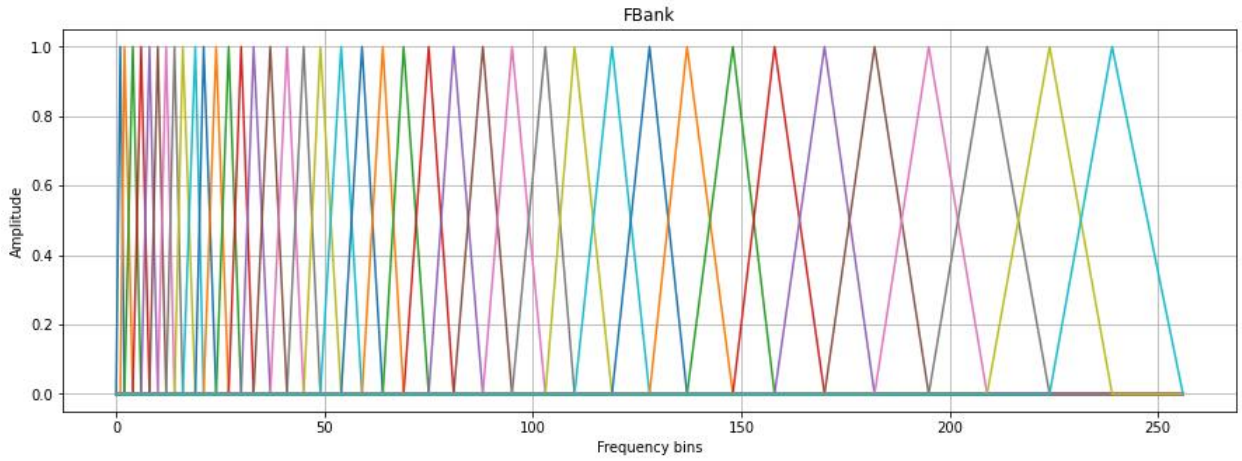


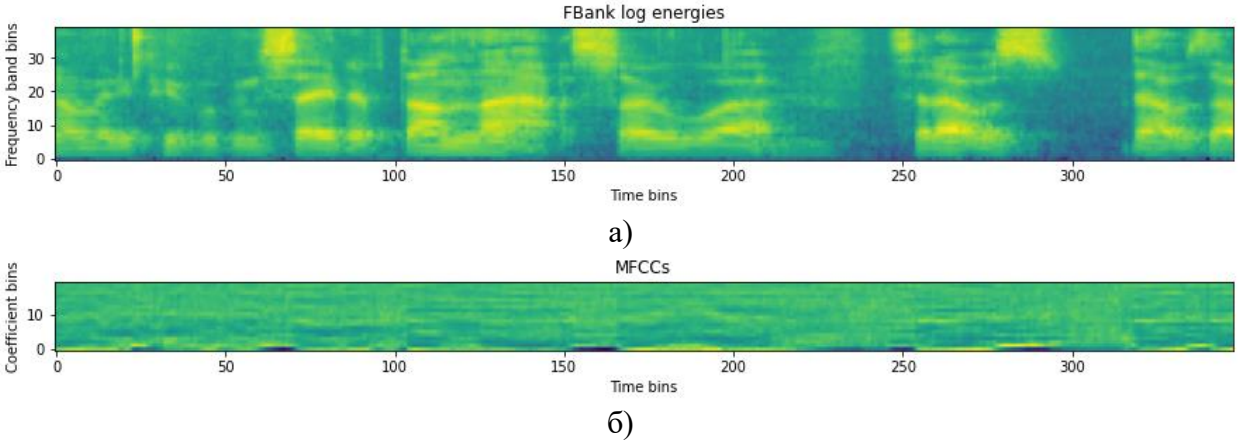
Рисунок 2 – Пример квадратов АЧХ мел-банка фильтров

Для вычисления и визуализации логарифмов энергий на выходе мел-банка фильтров, а также мел-частотных кепстральных коэффициентов можно воспользоваться кодом ниже.

```
# Log mel-filter bank energies
filter_banks_features = compute_fbanks_features(pow_frames, \
                                                fbank)

# Mel frequency cepstral coefficients
mfcc = compute_mfcc(filter_banks_features, num_ceps=20)

plt.figure(figsize=(15, 5))
plt.subplots_adjust(wspace=0, hspace=0.5)

plot_a = plt.subplot(211)
plot_a.imshow(filter_banks_features.T, origin='lower')
plot_a.set_xlabel('Time bins')
plot_a.set_ylabel('Frequency band bins')
plot_a.title.set_text('FBank log energies')

plot_b = plt.subplot(212)
im = plot_b.imshow(mfcc.T, origin='lower')
plot_b.set_xlabel('Time bins')
plot_b.set_ylabel('Coefficient bins')
```

```
plot_b.title.set_text('MFCCs')

plt.show()
```

Пример визуализации вычисленных логарифмов энергий на выходе мел-банка фильтров и также мел-частотных кепстральных коэффициентов для некоторого анализируемого сигнала представлен на рисунке 3.



а)

б)

Рисунок 3 – Результаты выполнения программного кода, позволяющего выполнить вычисление логарифмов энергий на выходе мел-банка фильтров и мел-частотных кепстральных коэффициентов: а) логарифмы энергий на выходе мел-банка фильтров; б) мел-частотные кепстральные коэффициенты

4. Выполнение локальных центрирования и масштабирования акустических признаков

Канал связи может вносить некоторое смещение в захваченный сигнал (микрофон может не иметь равномерной АЧХ, изменения в усилении сигнала приводят к вычислению различных акустических признаков даже для одного и того же куска речи). *Канальный эффект* [3] можно промоделировать с использованием *линейного инвариантного к сдвигу фильтра* (ЛИС) [5–8] и учесть при корректировке значений акустических признаков. Данная процедура получила название *процедуры нормализации* [3].

Идея выполнения процедуры нормализации состоит в вычислении среднего вектора наблюдаемых акустических признаков и центрирования всех векторов акустических признаков произнесения на это среднее. Процедура нормализации может быть выполнена несколькими различными способами:

1. *Локально по произнесению* (вычисление среднего в некоторой окрестности, обычно 300 фреймов, каждого фрейма стека признаков и нормализация на него этого фрейма);
2. *Глобально по произнесению* (вычисление среднего один раз по всему произнесению и нормализация на него всех фреймов этого произнесения);

3. *Глобально по базе данных* (вычисление среднего общего для всех произнесений в тренировочной базе данных и нормализация на него всех фреймов всех произнесений).

Иногда процедура нормализации сопровождается *процедурой масштабирования* [3] акустических признаков. Процедура масштабирования признаков может быть выполнена теми же способами, что и процедура нормализации. Отличие процедуры масштабирования признаков от процедуры нормализации состоит в том, что при её выполнении необходимо вычислить не средний вектор акустических признаков, а вектор среднеквадратического отклонения, на который необходимо поэлементно разделить каждый нормализованный вектор акустических признаков некоторого произнесения.

Предполагая, что набор акустических признаков до выполнения процедур нормализации и масштабирования был задан функцией $|X(k,m)|$, а после выполнения данных процедур – $|X_{norm}(k,m)|$, запишем следующее выражение:

$$|X_{norm}(k,m)| = (X(k,m) - m_X)/\sigma_X,$$

где m_X – это средний вектор акустических признаков, σ_X – это вектор среднеквадратического отклонения акустических признаков, k и m имеют суть частоты и времени (k – номер спектральной составляющей, а m – это номер фрейма).

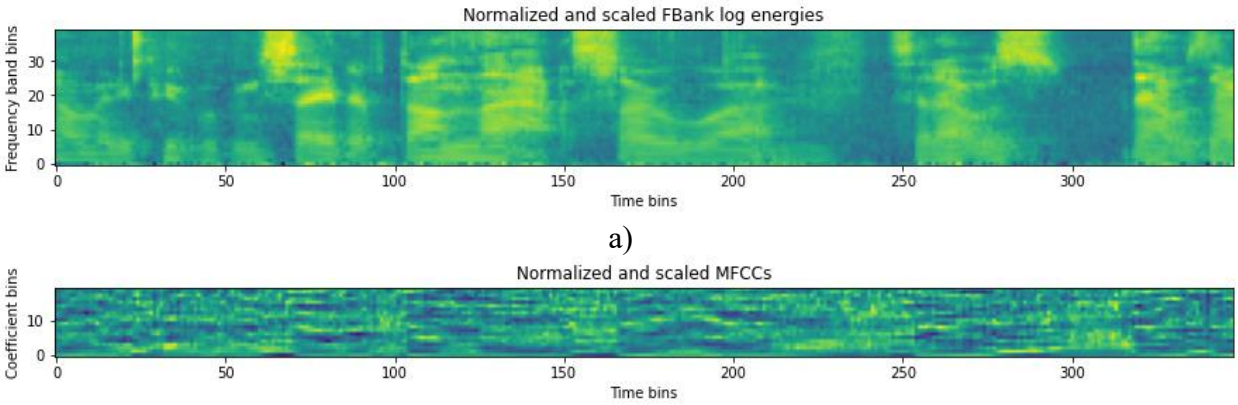
В рамках настоящего пункта требуется выполнить локальные по произнесению процедуры нормализации и масштабирования по отношению к логарифмам энергий на выходе банка фильтров и мел-частотным кепстральным коэффициентам, вычисленным для некоторой звукозаписи.

Для вычисления и визуализации нормализованных и масштабированных логарифмов энергий на выходе мел-банка фильтров, а также нормализованных и масштабированных мел-частотных кепстральных коэффициентов можно воспользоваться кодом ниже.

```
# Normalized and scaled log mel-filter bank energies
filter_banks_features_mvn = \
mvn_floating(filter_banks_features, 150, 150)

# Normalized and scaled mel frequency cepstral coefficients
mfcc_cmvn = mvn_floating(mfcc, 150, 150)

fig = plt.figure(figsize=(15, 5))
plt.subplots_adjust(wspace=0, hspace=0.5)

plot_b = plt.subplot(211)
im_b = plot_b.imshow(filter_banks_features_mvn.T,
origin='lower')
plot_b.set_xlabel('Time bins')
plot_b.set_ylabel('Frequency band bins')
plot_b.title.set_text('Normalized FBank log energies')
```

```
plot_c = plt.subplot(212)
im_c = plot_c.imshow(mfcc_cmvn.T, origin='lower')
plt.colorbar(im_c, cax=cax)
plot_c.set_xlabel('Time bins')
plot_c.set_ylabel('Coefficient bins')
plot_c.title.set_text('Normalized MFCCs')
plt.show()
```

Для успешного запуска программного кода выше необходимо завершить функцию нормализации и масштабирования акустических признаков `mvn_floating` из **`./exercises_blank.py`**. Пример вычисленных нормализованных и масштабированных акустических признаков представлен на рисунке 4.



а)

б)

Рисунок 4 – Результаты выполнения программного кода, позволяющего выполнить нормализацию и масштабирование акустических признаков: а) нормализованные и масштабированные логарифмы энергий на выходе мел-банка фильтров; б) нормализованные и масштабированные мел-частотные кепстральные коэффициенты

5. Построение распределений первых трёх компонент полученных акустических признаков

Для того чтобы проверить правильность расчёта акустических признаков выше, построим гистограммы распределения первых трёх компонент логарифмов энергий на выходе банка фильтров и мел-частотных кепстральных коэффициентов по некоторой базе данных. Рассмотрим в качестве этой базы звукозаписи мужских и женских голосов дикторов, список которых представлен в текстовом файле **`./metadata/meta.txt`**. В указанном файле перечислен список из 20 звукозаписей (10 для дикторов женского пола и 10 для дикторов мужского пола) из базы VoxCeleb1 test set.

Используя звукозаписи, список которых перечислен в **`meta.txt`**, в рамках настоящего пункта необходимо выполнить построение гистограмм распределения первых 3 компонент логарифмов энергий на выходе банка фильтров и мел-частотных кепстральных коэффициентов отдельно для базы мужских и женских голосов. Пример программного кода, позволяющего

выполнить построение требуемых гистограмм для случая логарифмов энергий на выходе мел-банка фильтров, представлен ниже.

```
fbank = compute_fbank_filters(nfilt=40, \
                              sample_rate=16000, \
                              NFFT=512)

def compute_feats(signal):
    # Function to compute log mel-filter bank energies and
    # mel frequency cepstral coefficients

    emphasized_signal = preemphasis(signal)
    frames = framing(emphasized_signal)
    pow_frames = power_spectrum(frames)
    filter_banks_features = \
    compute_fbanks_features(pow_frames, fbank)
    mfcc = compute_mfcc(filter_banks_features, num_ceps=20)

    return filter_banks_features, mfcc

male_fb_features = []
female_fb_features = []
male_mfcc_features = []
female_mfcc_features = []

for (path_to_wav, gender) in zip(paths, genders):
    # Load signal
    signal, sample_rate = torchaudio.load(path_to_wav)
    signal = signal.numpy().squeeze(axis=0)
    signal = signal/np.abs(signal).max()

    # Processing
    filter_banks_mvn, mfcc_cmvn = compute_feats(signal)
    if gender == 'm':
        male_fb_features.append(filter_banks_mvn)
        male_mfcc_features.append(mfcc_cmvn)

    else:
        female_fb_features.append(filter_banks_mvn)
        female_mfcc_features.append(mfcc_cmvn)

male_fb_features = np.concatenate(male_fb_features)
print(male_fb_features.shape)

female_fb_features = np.concatenate(female_fb_features)
print(female_fb_features.shape)

male_mfcc_features = np.concatenate(male_mfcc_features)
print(male_mfcc_features.shape)

female_mfcc_features = np.concatenate(female_mfcc_features)
print(female_mfcc_features.shape)
```

```
comp_number = 1
coeff1_male = male_fb_features[:, comp_number]
coeff1_female = female_fb_features[:, comp_number]

min_coeff1 = min(coeff1_male.min(), coeff1_female.min())
max_coeff1 = min(coeff1_male.max(), coeff1_female.max())

hist(coeff1_male, \
     int(sqrt(len(coeff1_male))), \
     histtype='step', \
     color='green', \
     range=(min_coeff1, max_coeff1), \
     density=1)
hist(coeff1_female, \
     int(sqrt(len(coeff1_female))), \
     histtype='step', \
     color='red', \
     range=(min_coeff1, max_coeff1), \
     density=1)
xlabel('MFBs, 2nd component'); ylabel('Histogram value')
title('Normalized histograms'); grid(); show()
```

Пример визуализации гистограмм распределения логарифмов энергий одной из компонент на выходе банка фильтров для мужских и женских представлен на рисунке 5.

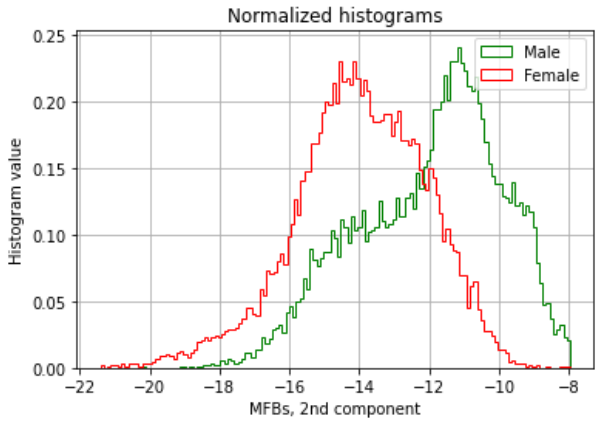


Рисунок 5 – Пример визуализации гистограмм распределения логарифмов энергий одной из компонент на выходе мел-банка фильтров для мужских и женских

Порядок защиты и представление результатов выполнения лабораторной работы

Для защиты лабораторной работы студент должен предоставить преподавателю самостоятельно завершённые программные коды из файла **`./exercises_blank.py`** и продемонстрировать их работоспособность путём поэтапного запуска файла **`./lab1_blank.ipynb`**. Преподаватель оставляет за собой право задать необходимые вопросы, затрагивающие

программную реализацию заданий лабораторной работы. В процессе запуска файла **`./lab1_blank.ipynb`** студенту необходимо:

1. Визуализировать пример осциллограммы и спектрограммы выбранного речевого сигнала и проанализировать их (отметить речевые участки, содержащие вокализованную и невокализованную речь);
2. Визуализировать спектрограммы выбранного речевого сигнала до и после процедуры преэмфазиса. Определить визуально отличия между построенными спектрограммами;
3. Визуализировать квадраты АЧХ мел-банка фильтров. Объяснить полученный результат;
4. Визуализировать логарифмы энергий на выходе мел-банка фильтров и мел-частотные кепстральные коэффициенты для выбранного речевого сигнала до и после выполнения процедур локального центрирования и масштабирования акустических признаков. Пояснить полученный результат;
5. Визуализировать распределения первых трёх компонент полученных акустических признаков. Проанализировать возможность детектирования мужских и женских голосов на основе построенных распределений.

Контрольные вопросы

1. Какие способы представления сигналов существуют?
2. Что такое спектр Фурье (амплитудный и фазовый)?
3. Что такое оконное преобразование Фурье?
4. Что такое спектрограмма?
5. Как выполнить процедуру преэмфазиса?
6. Описать процедуру вычисления акустических признаков разных типов.
7. Для каких целей выполняются процедуры нормализации и масштабирования акустических признаков?

Список литературы

1. Nagrani A., Chung J.S., Zisserman A. VoxCeleb: a large-scale speaker identification dataset // arXiv preprint arXiv:1706.08612 [cs.SD]. 2017.
2. Huang X., Acero A., Hon H. Spoken language processing: a guide to theory, algorithm, and system development. Prentice Hall, 2001.
3. Beigi H. Fundamentals of speaker recognition. Springer, 2011.
4. Černocký H. Voice biometry standard proposal // Interspeech 2015. 2015.
5. Hayes M.H. Schaum’s outlines of digital signal processing. McGraw-Hill, 2011.
6. Брюханов Ю.А. Цифровые цепи и сигналы. – М.: Горячая линия – Телеком, 2017.

7. Хрящёв В.В., Приоров А.Л., Волохов В.А. Основы теории цепей: сборник задач. – Ярославль: ЯрГУ, 2008.
8. Столбов М.Б. Основы анализа и обработки речевых сигналов. – СПб.: НИУ ИТМО, 2021.
9. Ляксо Е.Е., Фролова О.В., Гречаный С.В., Матвеев Ю.Н., Верхоляк О.В., Карпов А.А. Голосовой портрет ребенка с типичным и атипичным развитием. – СПб.: Издательско-полиграфическая ассоциация высших учебных заведений, 2020.
10. Davis S. Mermelstein P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences // IEEE Trans. Acoustics, Speech, and Signal Processing. 1980. V. 28, № 4. P. 357–366.

Лабораторная работа № 2

ОБУЧЕНИЕ ДЕТЕКТОРА РЕЧЕВОЙ АКТИВНОСТИ

Цель работы: изучение процедуры построения энергетического детектора речевой активности на основе модели гауссовой смеси.

Краткое описание: в рамках настоящей лабораторной работы предлагается изучить и реализовать энергетический детектор речевой активности на основе модели гауссовой смеси; реализация детектора предполагает субъективную и объективную оценку его качества по целевым метрикам; тестирование детектора выполняется в идеализированных условиях, а также при наличии шумов и помех.

Данные: в качестве данных для выполнения лабораторной работы предлагается использовать тестовую часть базы VoxCeleb1.

Содержание лабораторной работы

1. Подготовка данных для обучения и тестирования детектора речевой активности.
2. Изучение файлового формата RTTM.
3. Реализация энергетического детектора речевой активности на основе модели гауссовой смеси.
4. Коррекция разметки детектора речевой активности с использованием методов морфологической обработки.
5. Тестирование реализованного детектора речевой активности на «чистых» данных.
6. Тестирование реализованного детектора речевой активности на искажённых данных.

Порядок выполнения и краткая теория

1. Подготовка данных для обучения и тестирования детектора речевой активности

Перед выполнением лабораторной работы требуется импортировать некоторые Python-модули.

```
# Import of modules
import os
import sys

sys.path.append(os.path.realpath('..'))
```

```
from math import sqrt, pi

import numpy as np
from matplotlib.pyplot import figure, plot, show, \
grid, title, xlabel, ylabel, legend, hist

from skimage.morphology import opening, closing
from torchaudio.transforms import Resample
import torchaudio

from common import download_dataset, extract_dataset
from exercises_blank import load_vad_markup, framing, \
frame_energy, norm_energy, gmm_train, eval_frame_post_prob, \
energy_gmm_vad, reverb, awgn
```

В ходе выполнения лабораторной работы необходимы данные для осуществления процедуры обучения и процедуры тестирования реализуемого детектора речевой активности. Возьмём в качестве этих данных несколько звукозаписей голосов людей мужского и женского пола, сохраненных в формат wav, выбранных из корпуса VoxCeleb1 test set [1]. Данный корпус содержит 4874 звукозаписи (частота дискретизации равна 16 кГц) 40 дикторов мужского и женского пола, разговаривающих преимущественно на английском языке.

В рамках настоящего пункта требуется выполнить загрузку и распаковку звуковых wav-файлов из корпуса VoxCeleb1 test set. Предположим, что здесь и далее в рамках настоящей работы рабочим директорием является **`./lab2`**, находящийся внутри репозитория с лабораторными работами, и все пути до директориев и файлов указаны относительно него. Загрузку и распаковку данных можно выполнить с использованием программного кода, представленного ниже.

```
# Download VoxCeleb1 test set
with open('../data/lists/datasets.txt', 'r') as f:
    lines = f.readlines()

download_dataset(lines, \
                 user='voxceleb1902', \
                 password='nx0bl2v2', \
                 save_path='../data')

# Extract VoxCeleb1 test set
extract_dataset(save_path='../data/voxceleb1_test', \
                fname='../data/vox1_test_wav.zip')
```

2. Изучение файлового формата RTTM

Файлы расширенной транскрипции с метками времени (RTTM, rich transcription time marked) представляют собой текстовые файлы, в каждой

строке которых содержатся разделённые пробелом метаданные привязанных к звукозаписи объектов. Метаданные аннотируют отдельные элементы звукозаписи. Каждая строка rttm-файла представляет аннотацию одного примера объекта. Типы объектов, привязанных к звукозаписи, варьируются по отношению к решаемой задачи. Так, например, при решении задачи диаризации разметка rttm-файла может содержать в каждой строке метаданные, связанные с одним из нескольких дикторов, разговаривающих в звукозаписи. Объектами в этом случае являются дикторы. В случае решения задачи построения детектора речевой активности разметка rttm-файла содержит в каждой строке метаданные, связанные с речевыми сегментами, присутствующими в звукозаписи. Объектом в этом случае является дикторская речь в целом.

Каждая строка rttm-файла содержит 10 полей. Из присутствующих в rttm-файле полей при выполнении настоящей лабораторной работы будут использоваться следующие: *поле 1*, тип объекта (`SPEAKER`, речь диктора), *поле 2*, имя файла без указания пути до него и расширения, *поле 3*, номер канала, с которым связан речевой сегмент, например, «1» или «2», *поле 4*, время начала речевого сегмента в секундах, *поле 5*, длительность речевого сегмента в секундах. *Поле 6*, *поле 7*, *поле 8*, *поле 9* и *поле 10* будут считаться неопределёнными (`NA`). Возможный пример части rttm-файла для некоторой звукозаписи представлен ниже.

```
SPEAKER 00006 1 3.186 1.733 <NA> <NA> <NA> <NA> <NA>
SPEAKER 00006 1 5.253 1.043 <NA> <NA> <NA> <NA> <NA>
SPEAKER 00006 1 6.376 2.007 <NA> <NA> <NA> <NA> <NA>
SPEAKER 00006 1 8.796 0.676 <NA> <NA> <NA> <NA> <NA>
```

В рамках настоящего пункта предлагается проанализировать содержимое директория **`./ground_truth/rttm/`**, в котором представлены разметки идеального детектора речевой активности в rttm-формате, полученные путём ручной разметки для некоторых звукозаписей из базы данных VoxCeleb1 test set. Настоящие эталонные разметки предполагается использовать для оценки качества реального детектора речевой активности ниже в рамках рассматриваемой лабораторной работы. Отметим, что эталонные разметки звукозаписей на «речь» и «не речь» были получены с использованием Wave Assistant – бесплатное программное обеспечение, позволяющее выполнять обработку речевых сигналов. Разметка детектора речевой активности в Wave Assistant может быть сохранена во внутренний seg-формат, из которого, путём несложной конвертации, разметка может быть представлена в rttm-формате.

Идеальные разметки из директория **`./ground_truth/rttm/`** получены для 20 звукозаписей (10 голосов дикторов женского пола и 10 голос дикторов мужского пола). Имена файлов идеальной разметки имеют следующий вид: **`id10271_1gtz-CUIygI_00006.rttm`**. В соответствии

со структурой данных базы VoxCeleb1 test set из примера выше это означает, что идеальная разметка сформирована для звукозаписи **00006.wav**, выбранной из сессии **1gtz-CUIygI** и соответствующей диктору с идентификационным номером **id10271**. Если скачивание и распаковка данных базы VoxCeleb1 test set выполнены корректно в соответствии с пунктом 1 выше, то звукозапись, которой соответствует файл **./ground_truth/rttm/id10271_1gtz-CUIygI_00006.rttm** может быть найдена по следующему пути: **../data/voxceleb1_test/wav/id10271/1gtz-CUIygI/00006.wav**.

Последовательность действий, которую необходимо выполнить при рассмотрении настоящего пункта, является следующей:

1. Загрузить и визуализировать осциллограмму любой звукозаписи, идеальная разметка детектора речевой активности которой находится в директории **./ground_truth/rttm/**;
2. Прочитать с использованием функционала базового ядра Python rttm-файл для загруженной звукозаписи;
3. Наложить разметку детектора речевой активности на осциллограмму визуализированной звукозаписи.

Ниже приведён пример программного кода, реализующий этапы, указанные выше. Для запуска программного кода необходимо самостоятельно завершить функцию `load_vad_markup` из файла **./exercises_blank.py**, позволяющую прочитать rttm-файл и сгенерировать разметку детектора речевой активности в отсчётах во временной области.

```
# Path to files
path_to_wav  = os.path.join('../data/voxceleb1_test/wav', \
               'id10271', '1gtz-CUIygI/00006.wav')
path_to_rttm = os.path.join('./ground_truth/rttm', \
               'id10271_1gtz-CUIygI_00006.rttm')

# Load signal
signal, fs = torchaudio.load(path_to_wav)
signal = signal.numpy().squeeze(axis=0)
signal = signal/np.abs(signal).max()

# Load ideal VAD's markup
vad_markup_ideal = load_vad_markup(path_to_rttm, signal, fs)

# Plot signal
figure(figsize=(16, 4))
plot(signal, color='green')
plot(vad_markup_ideal, color='red')
xlabel('$n$'); ylabel('$x(n)$')
title('Waveform and markup of ideal VAD'); grid()
legend(['Waveform', 'Ideal VAD']); show()
```

На рисунке 1 представлен результат выполнения программного кода выше в виде огибающей сигнала во временной области и наложенной на неё идеальной разметки детектора речевой активности, представленной в виде прямоугольных импульсов. Единичный уровень прямоугольных импульсов маркирует речевые сегменты, а нулевой уровень – неречевые.

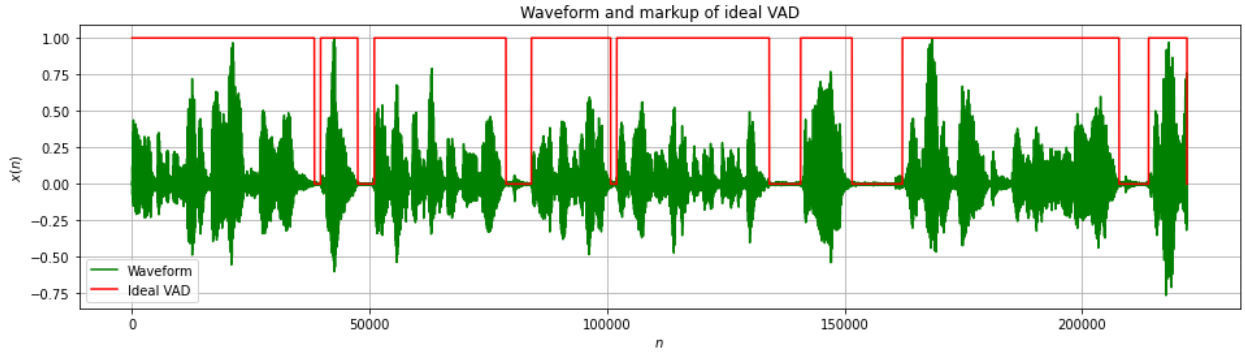


Рисунок 1 – Осциллограмма речевого сигнала и соответствующая ему идеальная разметка детектора речевой активности

3. Реализация энергетического детектора речевой активности на основе модели гауссовой смеси

Детектор речевой активности (VAD, voice activity detector) [2] предназначен для определения присутствия или отсутствия речи в звукозаписи. Входом детектора является обрабатываемый сигнал, а выходом – разметка на «речь» и «не речь». Представить разметку детектора речевой активности можно различными способами, например, с использованием rttm-файла, путём визуализации в виде последовательности прямоугольных импульсов переменной длительности, наложенных на осциллограмму обрабатываемого сигнала во временной области и т.п. В последнем случае речевые участки обычно маркируются уровнем «1», а неречевые участки – уровнем «0».

На данный момент времени существует множество различных реализаций детекторов речевой активности, в том числе современных [3], основанных на использовании теории *глубоких нейронных сетей* [4]. Необходимо отметить, что использование детекторов речевой активности является важным при решении задачи распознавания диктора, поскольку качество дикторских моделей, формируемых на основе речевых сигналов без длительных пауз, получается значительно лучше. В рамках настоящей лабораторной работы предлагается реализовать *энергетический детектор речевой активности на основе модели гауссовой смеси* [5]. Алгоритм реализации рассматриваемого детектора и, фактически, последовательность действий в рамках настоящего пункта, представлены ниже:

1. Загрузить речевой сигнал, разметку на «речь» и «не речь» которого необходимо получить;

2. Разбить обрабатываемый речевой сигнал с использованием некоторой оконной функции, например, прямоугольной, на перекрывающиеся сегменты (фреймы или кадры);
3. Вычислить для каждого из сегментов энергию сигнала. Предположив, что длина оконной функции составляет L отсчётов, энергия сигнала в окне может быть вычислена с использованием следующего выражения:

$$E(i) = \sum_{n=0}^{L-1} |x(n+i)w(n)|^2,$$

где $x(n)$ – обрабатываемый речевой сигнал, $w(n)$ – оконная функция, i – шаг окна;
4. Выполнить нормализацию и масштабирование энергий сегментов на среднее значение и среднеквадратическое значение энергии, вычисленные по всем сегментам звукозаписи (*нормализация и масштабирование глобально по произнесению*), получив признаки для обучения модели гауссовой смеси. Вычислить среднее и среднеквадратическое значения энергии по всем сегментам обрабатываемого сигнала можно с использованием следующих выражений:

$$m_E = \frac{1}{M}\sum_{i=0}^{M-1} E(i),\ \sigma_E = \sqrt{\frac{1}{M}\sum_{i=0}^{M-1}(E(i) - m_E)^2},$$

где M – количество сегментов, на которые был разбит сигнал. Нормализованные и масштабированные энергии сегментов можно определить следующим образом:

$$E_{norm}(i) = (E(i) - m_E)/\sigma_E;$$

5. Обучить модель гауссовой смеси для случая трёх одномерных гауссианов с использованием подготовленных выше признаков. В рассматриваемой задаче предполагается, что два из этих гауссианов порождают признаки, соответствующие речевым сегментам, а третий гауссиан порождает признаки, соответствующие неречевым сегментам. Обучение модели гауссовых смесей можно выполнить с использованием *EM-алгоритма* [6], который, по своей сути, является вероятностным обобщением метода кластеризации на основе *k-средних* [6]. Предположим, что *модель одномерной гауссовой смеси* [5, 6] из трёх гауссианов на множестве нормированных и масштабированных энергий E_{norm} описывается с использованием следующей плотности вероятности:

$$W_{E_{norm}}(e) = \sum_{j=0}^{2} w_j W_j(e),\ \sum_{j=0}^{2} w_j = 1,$$

где $W_j(e)$ – функция правдоподобия -й компоненты смеси, а w_j – её априорная вероятность. Каждая компонента смеси может быть описана гауссовой плотностью вероятности:

$$W_j(e) = N(e;\, m_j, \sigma_j) = (\sigma_j\sqrt{2\pi})^{-1} \cdot e^{-\frac{(e-m_j)^2}{2\sigma_j^2}},$$

где m_j и σ_j – это математическое ожидание и среднеквадратическое отклонение соответствующих компонент гауссовой смеси, а $j = 0, 1, 2$.

E-шаг [5, 6] алгоритма обучения модели гауссовой смеси можно описать следующим образом:

$$g_{ij} = w_j N(E_{norm}(i);\, m_j, \sigma_j) / \sum_{s=0}^{2} w_s N(E_{norm}(i);\, m_s, \sigma_s).$$

M-шаг [5, 6] алгоритма обучения модели гауссовой смеси можно описать с использованием следующих выражений:

$$w_j = \frac{1}{M}\sum_{i=0}^{M-1} g_{ij},$$

$$m_j = \frac{1}{Mw_j}\sum_{i=0}^{M-1} g_{ij} E_{norm}(i),$$

$$\sigma_j = \sqrt{\frac{1}{Mw_j}\sum_{i=0}^{M-1} g_{ij}(E_{norm}(i) - m_j)^2},$$

где M – количество сегментов, на которые был разбит сигнал;

6. С использованием обученной модели оценить принадлежность сегмента к «не речи». В предположении, что значения энергий неречевых сегментов порождаются гауссианом с индексом $j = 0$, оценим апостериорные вероятности того, что объект $E_{norm}(i)$ получен из 0-й компоненты смеси:

$$\hat{g}_{i0} = \hat{w}_0 N(E_{norm}(i);\, \hat{m}_0, \hat{\sigma}_0) / \sum_{s=0}^{2} \hat{w}_s N(E_{norm}(i);\, \hat{m}_s, \hat{\sigma}_s),$$

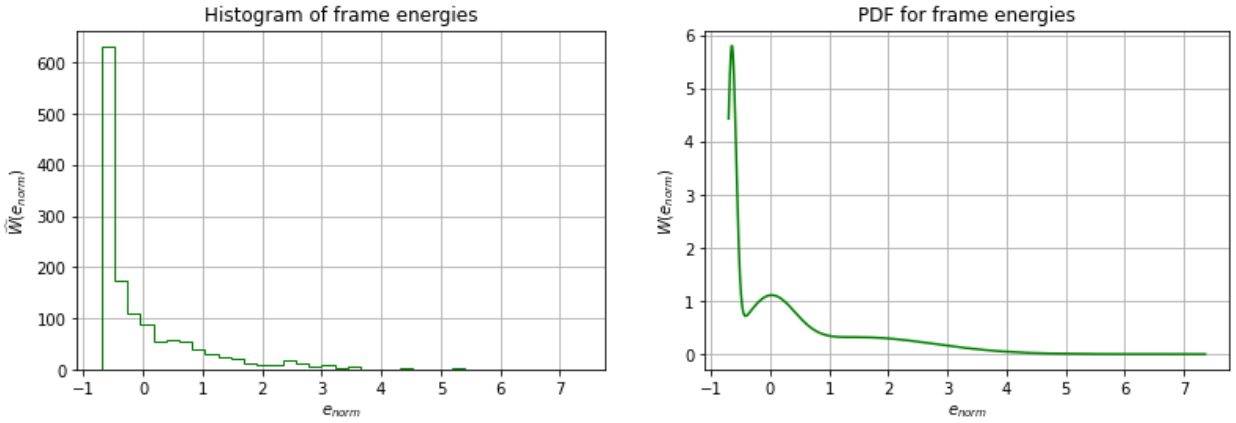
где $\hat{w}_j$, $\hat{m}_j$ и $\hat{\sigma}_j$ являются обученными параметрами модели гауссовой смеси. Чем выше $\hat{g}_{i0}$ для выбранного значения $E_{norm}(i)$, тем вероятнее, что соответствующий фрейм относится к неречевому;

7. Подобрать порог принятия решения для детектора речевой активности;
8. Сформировать пофреймовую разметку детектора речевой активности на «речь» и «не речь», маркируя речевые фреймы символом «1», а неречевые – символом «0». Выполнить данную процедуру можно путём сравнения величины $\hat{g}_{i0}$ для выбранного значения $E_{norm}(i)$ с заданным порогом детектора. Если величина $\hat{g}_{i0}$ находится ниже порога, то сегмент предполагается речевым, в противном случае – неречевым. В случае необходимости пофреймовая разметка детектора речевой активности может быть преобразована в разметку по отсчётам сигнала. Последнее позволит, например, визуально сравнить реальную и эталонную разметки детектора путём их наложения на осциллограмму обрабатываемой звукозаписи.

```
# Squared signal
squared_signal = signal**2

# Frame signal with overlap
window = 320 # window size in samples
shift  = 160 # window shift in samples

frames = framing(squared_signal, window=window, shift=shift)

# Sum frames to get energy
E = frame_energy(frames)

# Normalize the energy
E_norm = norm_energy(E)

# Plot histograms for frame energies
hist(E_norm, int(sqrt(len(E_norm))), \
     histtype='step', color='green')
xlabel('$e_{norm}$')
ylabel('$\widehat{W}(e_{norm})$')
title('Histogram of frame energies'); grid(); show()

# Gaussian probability density function (PDF)
gauss_pdf = lambda value, m, sigma: 1/((abs(sigma) + \
            1e-10)*sqrt(2*pi))*np.exp(-(value - \
            m)**2/(2*(abs(sigma) + 1e-10)**2))

# Train parameters of gaussian mixture models
w, m, sigma = gmm_train(E_norm, gauss_pdf, n_realignment=10)

GMM_pdf = np.zeros(len(E_norm))
for j in range(len(m)):
    GMM_pdf = GMM_pdf + gauss_pdf(sorted(E_norm), \
              m[j], sigma[j])

# Plot PDF for frame energies
plot(sorted(E_norm), GMM_pdf, color='green')
xlabel('$e_{norm}$'); ylabel('$W(e_{norm})$')
title('PDF for frame energies'); grid(); show()

# Estimate a posterior probability that frame isn't speech
g0 = eval_frame_post_prob(E_norm, gauss_pdf, w, m, sigma)

# Compute real VAD's markup
vad_thr = 0.3 # threshold of voice activity detector

vad_frame_markup_real = \
(g0 < vad_thr).astype('float32') # frame VAD's markup

vad_markup_real = \
np.zeros(len(signal)).astype('float32') # sample VAD's markup
```

```
for idx in range(len(vad_frame_markup_real)):
    vad_markup_real[idx*shift:shift+idx*shift] = \
    vad_frame_markup_real[idx]

vad_markup_real[len(vad_frame_markup_real)*shift - \
len(vad_markup_ideal):] = vad_frame_markup_real[-1]

# Plot signal
figure(figsize=(16, 4))
plot(signal, color='green')
plot(vad_markup_real, color='blue')
xlabel('$n$'); ylabel('$x(n)$')
title('Waveform and markup of real VAD'); grid()
legend(['Waveform', 'Real VAD']); show()
```

Выполнить задания настоящего пункта можно с использованием программного кода, представленного выше. Для запуска кода необходимо самостоятельно завершить функции `framing` (разбиение сигнала на фреймы), `frame_energy` (вычисление энергии для каждого фрейма), `norm_energy` (нормализация и масштабирование энергий фреймов) и `gmm_train` (обучение параметров модели гауссовой смеси) из **`./exercises_blank.py`**. Результаты выполнения программного кода выше представлены на рисунке 2 и рисунке 3. На рисунке 2 представлена гистограмма распределения пофреймовых энергий, а также аппроксимация распределения с помощью обученной модели гауссовой смеси.



а) б)

Рисунок 2 – Результаты выполнения программного кода, позволяющего рассчитать реальную разметку детектора речевой активности некоторой звукозаписи: а) гистограмма распределения пофреймовых энергий; б) аппроксимация распределения с помощью обученной модели гауссовой смеси

На рисунке 3 представлена огибающая сигнала во временной области и наложенная на неё реальная разметка детектора речевой активности.

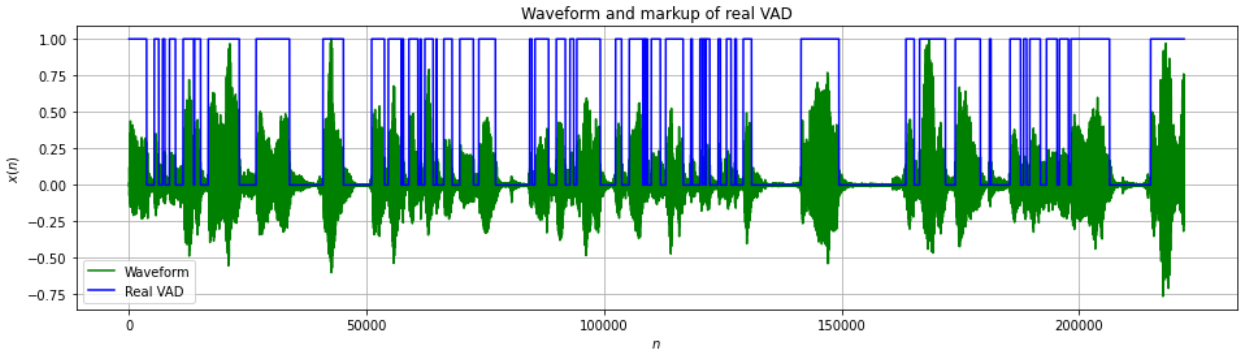


Рисунок 3 – Осциллограмма речевого сигнала и соответствующая ему реальная разметка детектора речевой активности

4. Коррекция разметки детектора речевой активности с использованием методов морфологической обработки

Анализ разметки, полученной с использованием детектора речевой активности выше, показывает, что она сильно отличается от эталонной разметки высокой частотой нарезки звукозаписи на речевые сегменты. Попытаться сделать реальную разметку детектора менее частой, в случае необходимости, можно с использованием методов *морфологической фильтрации* [7], применяемых, например, в ранних системах обработки цифровых изображений, в первую очередь *бинарных* [7], состоящих только из пикселей двух уровней яркости. Фактически, разметку детектора речевой активности можно рассматривать в качестве одной строки или столбца бинарного цифрового изображения. Поэтому с использованием некоторой одномерной бинарной маски *морфологического фильтра*, называемой *структурирующим элементом*, разметка детектора речевой активности может быть отфильтрована.

Ключевыми процедурами морфологической обработки являются следующие:

1. *Наращивание* – процедура морфологической фильтрации, позволяющая увеличить область цифрового изображения;
2. *Эрозия* – процедура морфологической фильтрации, позволяющая уменьшить область цифрового изображения;
3. *Замыкание* – процедура морфологической фильтрации, представляющая последовательное использование наращивания и эрозии;
4. *Размыкание* – процедура морфологической фильтрации, представляющая последовательное использование эрозии и наращивания.

В рамках настоящего пункта предлагается провести эксперименты с процедурами замыкания и размыкания применительно к полученной ранее разметке детектора речевой активности. Пример программного кода,

позволяющего выполнить подобные эксперименты, представлен ниже. На рисунке 4 представлен результат коррекции разметки рассматриваемого детектора речевой активности с помощью методов морфологической фильтрации.

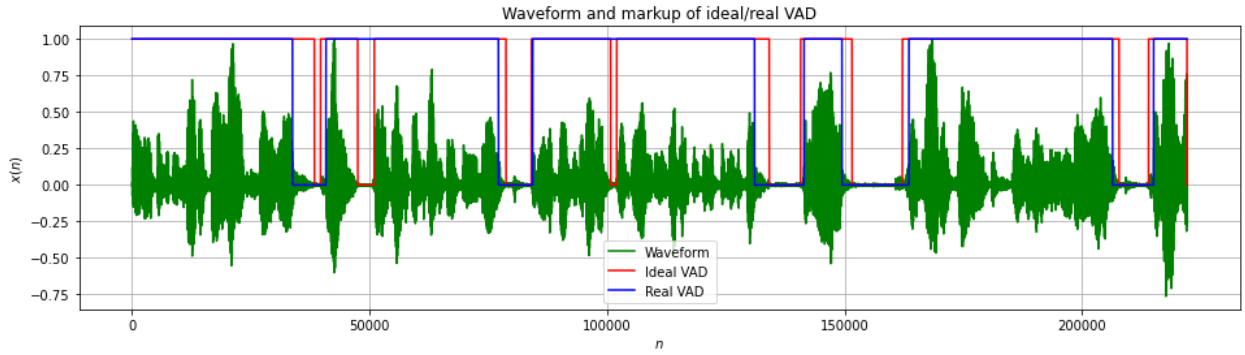


Рисунок 4 – Осциллограмма речевого сигнала, идеальная разметка детектора речевой активности и реальная разметка после коррекции с использованием методов морфологической фильтрации

```
# Morphology Filters
mask_size = 6000 # size of filter's mask

# Close filter
vad_markup_real_filt = closing(vad_markup_real, \
                       np.ones(mask_size))

# Open filter
vad_markup_real_filt = opening(vad_markup_real_filt, \
                       np.ones(mask_size))

# Plot signal
figure(figsize=(16, 4))
plot(signal, color='green')
plot(vad_markup_ideal, color='red')
plot(vad_markup_real_filt, color='blue')
xlabel('$n$'); ylabel('$x(n)$')
title('Waveform and markup of ideal/real VAD'); grid()
legend(['Waveform', 'Ideal VAD', 'Real VAD']); show()
```

5. Тестирование реализованного детектора речевой активности на «чистых» данных

Визуальная оценка качества работы детектора речевой активности является наглядной, но не даёт «полного» понимания того, насколько хорошо работает детектор речевой активности в рамках некоторого кейса (аутистические условия, звукозаписывающее оборудование, используемые кодеки и т.п.) и не позволяет эффективно сравнить работу одного детектора с другим. Для решения указанной проблемы требуется использовать *объективные метрики оценки качества*. Поскольку результатом детектора

речевой активности является разделение сигнала на участки двух типов, «речь» и «не речь», то для численной оценки качества, в данном случае, подойдут метрики, используемые в *бинарной теории принятия решений* [8]. К ним можно отнести *вероятности ошибок первого* (FRR, false rejection rate, или FNR, false negative rate) и *второго рода* (FAR, false acceptance rate, или FPR, false positive rate), *точность* (P, precision) и *полнота выборки* (R, recall), *равный уровень ошибок* (EER, equal error rate) и т.п.

В рамках настоящего пункта требуется вычислить вероятности ошибок первого и второго рода, а также точность и полноту выборки для фиксированного значения порога детектора речевой активности. При вычислении метрик предположим, что *нулевой гипотезой* является наличие речи, а *альтернативной* – отсутствие речи в некотором сегменте обрабатываемого сигнала. В качестве данных для оценки качества детектора речевой активности воспользуемся 20 звукозаписями, разметка идеального детектора которых представлена в директории **`./ground_truth/rttm/`**.

Ниже приведён программный код, позволяющий выполнить эксперименты в рамках настоящего пункта.

```
# Parameters of voice activity detector
window = 320
shift  = 160
n_realignment = 10
vad_thr = 0.3
mask_size_morph_filt = 6000

# Gaussian probability density function
gauss_pdf = lambda value, m, sigma: 1/((abs(sigma) + \
            1e-10)*sqrt(2*pi))*np.exp(-(value - \
            m)**2/(2*(abs(sigma) + 1e-10)**2))

# Path to files
path_to_wav  = '../data/voxceleb1_test/wav'
path_to_rttm = './ground_truth/rttm'

TP, FP, FN, TN = 0.0, 0.0, 0.0, 0.0

for rttm_file in os.listdir(path_to_rttm):
    # Load signal
    signal, fs = \
    torchaudio.load(os.path.join(path_to_wav, rttm_file[:7], \
    rttm_file[8:19], '.'.join([rttm_file[20:25], 'wav'])))

    signal = signal.numpy().squeeze(axis=0)
    signal = signal/np.abs(signal).max()

    # Load ideal VAD's markup
    vad_markup_ideal = \
    load_vad_markup(os.path.join(path_to_rttm, \
    rttm_file), signal, fs)
```

```
    # Compute real VAD's markup
    vad_markup_real = energy_gmm_vad(signal, \
                                     window, \
                                     shift, \
                                     gauss_pdf, \
                                     n_realignment, \
                                     vad_thr, \
                                     mask_size_morph_filt)

    TP = TP + np.sum(vad_markup_ideal*vad_markup_real)
    FP = FP + np.sum((1 - vad_markup_ideal)*vad_markup_real)
    FN = FN + np.sum(vad_markup_ideal*(1 - vad_markup_real))
    TN = TN + np.sum((1 - vad_markup_ideal)*(1 - \
         vad_markup_real))

FNR = FN/(FN + TP) # false negative rate
FPR = FP/(FP + TN) # false positive rate
P   = TP/(TP + FP) # precision
R   = TP/(TP + FN) # recall

print('Threshold value:       {0:.3f}'.format(vad_thr))
print('False negative rate:   {0:.3f}'.format(FNR))
print('False positive rate:   {0:.3f}'.format(FPR))
print('Precision:             {0:.3f}'.format(P))
print('Recall:                {0:.3f}'.format(R))
```

6. Тестирование реализованного детектора речевой активности на искажённых данных

Процедуры формирования и передачи речевого сигнала могут сопровождаться воздействием шумов и помех, приводящих к искажению сигнала. В качестве примеров искажающих факторов, влияющих на ухудшение качества речевого сигнала, можно привести импульсный отклик помещения (реверберация), фоновый шум голосов группы нецелевых дикторов, звук телевизора или радиоприёмника и т.п. Разработка конвейера системы голосовой биометрии требует учёта воздействия искажающих факторов на качество работы отдельных блоков и всей системы в целом. Поскольку отдельные этапы современных систем голосовой биометрии реализуются с использованием глубоких нейронных сетей, требующих наличия больших объёмов данных для их обучения, то возможным вариантом учёта искажений, воздействующих на систему, может являться расширение тренировочной выборки для обучения моделей за счёт методов аугментации [9, 10]. *Аугментация* – методика создания дополнительных обучающих примеров из имеющихся данных. Как правило, при решении задачи аугментации данных в речевой обработке используются дополнительные базы шумов и помех. В качестве примеров можно привести базы SLR17 [9] – MUSAN (корпус музыкальных, речевых и шумовых звукозаписей) и SLR28 [10] – RIR noises (набор данных реальных

и симулированных импульсных откликов комнат, а также изотропных и точечных шумов).

Рассмотрим в рамках настоящего пункта техники аугментации, моделирующие воздействие *реверберации* [10] и *аддитивного белого гауссовского шума* (АБГШ) [8] на речевой сигнал, и проанализируем их влияние на качество работы детектора речевой активности. В рамках настоящего пункта требуется:

1. Промоделировать эффект реверберации для некоторого записанного импульсного отклика помещения с использованием *линейного инвариантного к сдвигу фильтра* [8]. При моделировании можно воспользоваться импульсным откликом Лозаннского собора (Лозанна, Швейцария), который измерили И. Докманич и его коллеги из Федеральной политехнической школы Лозанны, записав звук лопнувшего в соборе воздушного шарика;
2. Промоделировать воздействие аддитивного шума с использованием модели АБГШ для различных уровней мощности шума;
3. Выполнить тестирование детектора речевой активности при наличии реверберации или АБГШ с использованием описанных выше численных метрик оценки качества.

Выполнить задания настоящего пункта можно с использованием программного кода ниже. Для выполнения кода необходимо завершить функции `reverb` (наложение эффекта реверберации на звукозапись) и `awgn` (добавление АБГШ к звукозаписи) из файла **`./exercises_blank.py`**.

```
# Parameters of voice activity detector
window = 320
shift  = 160
n_realignment = 10
vad_thr = 0.3
mask_size_morph_filt = 6000

# Gaussian probability density function
gauss_pdf = lambda value, m, sigma: 1/((abs(sigma) + \
            1e-10)*sqrt(2*pi))*np.exp(-(value - \
            m)**2/(2*(abs(sigma) + 1e-10)**2))

# Path to files
path_to_wav  = '../data/voxceleb1_test/wav'
path_to_rttm = './ground_truth/rttm'

aug_mode = 'awgn' # 'reverb' or 'awgn'

TP, FP, FN, TN = 0.0, 0.0, 0.0, 0.0

for rttm_file in os.listdir(path_to_rttm):
    # Load signal
    signal, fs = \
```

```
    torchaudio.load(os.path.join(path_to_wav, \
    rttm_file[:7], rttm_file[8:19], \
    '.'.join([rttm_file[20:25], 'wav'])))

    signal = signal.numpy().squeeze(axis=0)
    signal = signal/np.abs(signal).max()

    if aug_mode == 'reverb':
        impulse_response, ir_fs = \
        torchaudio.load('./cathIR.wav')

        resample = Resample(orig_freq=ir_fs, \
                   new_freq=fs, \
                   resampling_method='sinc_interpolation')

        impulse_response = resample(impulse_response)
        impulse_response = \
        impulse_response.numpy().squeeze(axis=0)
        impulse_response = \
        impulse_response/np.abs(impulse_response).max()

        signal = reverb(signal, impulse_response)

    else:
        sigma_noise = 0.15
        signal = awgn(signal, sigma_noise)

    # Load ideal VAD's markup
    vad_markup_ideal = \
    load_vad_markup(os.path.join(path_to_rttm, \
                    rttm_file), signal, fs)

    # Compute real VAD's markup
    vad_markup_real = energy_gmm_vad(signal, \
                                     window, \
                                     shift, \
                                     gauss_pdf, \
                                     n_realignment, \
                                     vad_thr, \
                                     mask_size_morph_filt)

    TP = TP + np.sum(vad_markup_ideal*vad_markup_real)
    FP = FP + np.sum((1 - vad_markup_ideal)*vad_markup_real)
    FN = FN + np.sum(vad_markup_ideal*(1 - vad_markup_real))
    TN = TN + np.sum((1 - vad_markup_ideal)*(1 - \
         vad_markup_real))

FNR = FN/(FN + TP) # false negative rate
FPR = FP/(FP + TN) # false positive rate
P   = TP/(TP + FP) # precision
R   = TP/(TP + FN) # recall
```

```
print('Threshold value:       {0:.3f}'.format(vad_thr))
print('False negative rate:   {0:.3f}'.format(FNR))
print('False positive rate:   {0:.3f}'.format(FPR))
print('Precision:             {0:.3f}'.format(P))
print('Recall:                {0:.3f}'.format(R))
```

Порядок защиты и представление результатов выполнения лабораторной работы

Для защиты лабораторной работы студент должен предоставить преподавателю самостоятельно завершённые программные коды из файла **./exercises_blank.py** и продемонстрировать их работоспособность путём поэтапного запуска файла **./lab2_blank.ipynb**. Преподаватель оставляет за собой право задать необходимые вопросы, затрагивающие программную реализацию заданий лабораторной работы. В процессе запуска файла **./lab2_blank.ipynb** студенту необходимо:

1. Визуализировать осциллограмму и идеальную разметку детектора речевой активности для выбранного сигнала, описать полученный результат;
2. Визуализировать гистограмму распределения пофреймовых энергий и её аппроксимацию для выбранного речевого сигнала. Пояснить на основе полученной гистограммы то, какие значения пофреймовых энергий соответствуют речевым и неречевым участкам сигнала;
3. Визуализировать разметку полученного реального детектора речевой активности для выбранного сигнала до и после применения методов морфологической обработки. Сравнить полученные разметки между собой, а также с идеальной разметкой;
4. Оценить качество полученного реального детектора речевой активности на «чистых» и искажённых данных с помощью предлагаемых объективных метрик оценки качества. Проанализировать насколько сильно изменяется качество работы реального детектора речевой активности для различных типов искажений по отношению к случаю, когда детектором обрабатываются «чистые» данные.

Контрольные вопросы

1. Как устроен rttm-файл и для каких целей он используется?
2. Как вычислить энергию сигнала?
3. Что такое модель гауссовой смеси?
4. Что такое детектор речевой активности?
5. Как устроен энергетический детектор речевой активности на основе модели гауссовой смеси?

6. Основные достоинства и недостатки энергетического детектора речевой активности.
7. В чём идея использования методов морфологической обработки в коррекции реальной разметки детектора речевой активности?
8. Какие численные метрики оценки качества можно использовать для описания результата работы детектора речевой активности?
9. Что такое аугментация данных и для каких целей она используется?
10. Как можно построить более совершенный, чем энергетический, детектор речевой активности?

Список литературы

1. Nagrani A., Chung J.S., Zisserman A. VoxCeleb: a large-scale speaker identification dataset // arXiv preprint arXiv:1706.08612 [cs.SD]. 2017.
2. Beigi H. Fundamentals of speaker recognition. Springer, 2011.
3. Gusev A., Volokhov V., Andzhukaev T., Novoselov S., Lavrentyeva G., Volkova M., Gazizullina A., Shulipa A., Gorlanov A., Avdeeva A., Ivanov A., Kozlov A., Pekhovsky T., Matveev Y. Deep speaker embeddings for far-field speaker recognition on short utterances // The Speaker and Language Recognition Workshop (Odyssey 2020). 2020. P. 179–186.
4. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб.: Питер, 2018.
5. Černocký H. Voice biometry standard proposal // Interspeech 2015. 2015.
6. Bishop C.M. Pattern recognition and machine learning. Springer, 2006.
7. Шапиро Л., Стокман Дж. Компьютерное зрение. – М.: БИНОМ. Лаборатория знаний, 2015.
8. Hsu H.P. Schaum's Ouline of Theory and Problems of Probability, Random Variables, and Random Processes. McGraw-Hill, 1997.
9. Snyder D., Chen G., Povey D. MUSAN: a music, speech, and noise corpus // arXiv preprint arXiv:1510.08484 [cs.SD]. 2015.
10. Ko T., Peddinti V., Povey D., Seltzer M.L., Khudanpur S. A study on data augmentation of reverberant speech for robust speech recognition // 2017 IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP). 2017. P. 5220–5224.

Лабораторная работа № 3

ПОСТРОЕНИЕ ДИКТОРСКИХ МОДЕЛЕЙ И ИХ СРАВНЕНИЕ

Цель работы: изучение процедуры построения дикторских моделей с использованием глубоких нейросетевых архитектур.

Краткое описание: в рамках настоящей лабораторной работы предлагается изучить и реализовать схему построения дикторских моделей с использованием глубокой нейросетевой архитектуры, построенной на основе ResNet-блоков. Процедуры обучения и тестирования предлагается рассмотреть по отношению к задаче идентификации на закрытом множестве, то есть для ситуации, когда дикторские классы являются строго заданными. Тестирование полученной системы предполагает использование доли правильных ответов в качестве целевой метрики оценки качества.

Данные: в качестве данных для выполнения лабораторной работы предлагается использовать базу VoxCeleb1.

Содержание лабораторной работы

1. Подготовка данных для обучения и тестирования блока построения дикторских моделей.
2. Обучение параметров блока построения дикторских моделей без учёта процедуры аугментации данных.
3. Обучение параметров блока построения дикторских моделей с учётом процедуры аугментации данных.
4. Тестирование блока построения дикторских моделей.

Порядок выполнения и краткая теория

1. Подготовка данных для обучения и тестирования блока построения дикторских моделей

Перед выполнением лабораторной работы требуется импортировать некоторые Python-модули.

```
# Import of modules
import os
import sys

sys.path.append(os.path.realpath('..'))

import torch
```

```
import torch.nn as nn
from torch.utils.data import DataLoader

from common import download_dataset, concatenate, \
extract_dataset, part_extract, download_protocol, split_musan
from exercises_blank import train_dataset_loader, \
test_dataset_loader, ResNet, MainModel, \
train_network, test_network
from ResNetBlocks import BasicBlock
from LossFunction import AAMSoftmaxLoss
from Optimizer import SGDOptimizer
from Scheduler import OneCycleLRScheduler
from load_save_pth import saveParameters, loadParameters
```

В ходе выполнения лабораторной работы необходимы данные для выполнения процедуры обучения и процедуры тестирования нейросетевого блока генерации дикторских моделей. Возьмём в качестве этих данных звукозаписи, сохраненные в формат wav, из корпуса VoxCeleb1 dev set [1]. Данный корпус содержит 148642 звукозаписи (частота дискретизации равна 16 кГц) для 1211 дикторов женского и мужского пола, разговаривающих преимущественно на английском языке.

В рамках настоящего пункта требуется выполнить загрузку и распаковку звуковых wav-файлов из корпуса VoxCeleb1 dev set, а также загрузку списка файлов, позволяющего разделить корпус VoxCeleb1 dev set на три подмножества (тренировочное, валидационное и тестовое) [1]. Предположим, что здесь и далее в рамках настоящей работы рабочим директорием является **./lab3**, находящийся внутри репозитория с лабораторными работами, и все пути до директориев и файлов указаны относительно него. Загрузку и распаковку данных можно выполнить с использованием программного кода, представленного ниже.

```
# Download VoxCeleb1 dev set
with open('../data/lists/datasets.txt', 'r') as f:
    lines = f.readlines()

download_dataset(lines, \
                 user='voxceleb1902', \
                 password='nx0bl2v2', \
                 save_path='../data')

# Concatenate archives for VoxCeleb1 dev set
with open('../data/lists/concat_arch.txt', 'r') as f:
    lines = f.readlines()

concatenate(lines, save_path='../data')

# Extract VoxCeleb1 dev set
extract_dataset(save_path='../data/voxceleb1_dev', \
                fname='../data/vox1_dev_wav.zip')
```

```
# Download VoxCeleb1 identification protocol
with open('../data/lists/protocols.txt', 'r') as f:
    lines = f.readlines()

download_protocol(lines, save_path='../data/voxceleb1_test')
```

2. Обучение параметров блока построения дикторских моделей без учёта процедуры аугментации данных

Построение современных *дикторских моделей*, как правило, выполняется с использованием нейросетевых архитектур [2], многие из которых позаимствованы из области обработки цифровых изображений. Одними из наиболее распространенных нейросетевых архитектур, используемых для построения дикторских моделей, являются ResNet-подобные архитектуры [2]. В рамках настоящего пункта предлагается выполнить адаптацию нейросетевой архитектуры ResNet34 для решения задачи генерации дикторских моделей (дикторских эмбеддингов) [3, 4]. *Дикторский эмбеддинг* – это высокоуровневый вектор признаков, состоящий, например, из 128, 256 и т.п. значений, содержащий особенности голоса конкретного человека.

При решении задачи распознавания диктора можно выделить эталонные и тестовые дикторские эмбеддинги [4]. *Эталонные эмбеддинги* формируются на этапе регистрации дикторской модели определённого человека и находятся в некотором хранилище данных. *Тестовые эмбеддинги* формируются на этапе непосредственного использования системы голосовой биометрии на практике, когда некоторый пользователь пытается получить доступ к соответствующим ресурсам. Система голосовой биометрии сравнивает по определённой метрике эталонные и тестовые эмбеддинги, формируя оценку сравнения, которая, после её обработки блоком принятия решения, позволяет сделать вывод о том, эмбеддинги одинаковых или разных дикторов сравниваются между собой.

Необходимо отметить, что построение дикторских моделей, как правило, требует наличия *акустических признаков* [5], вычисленных для звукозаписей тренировочной, валидационной и тестовой баз данных. В качестве примера подобных признаков в рамках настоящей лабораторной работы воспользуемся *логарифмами энергий на выходе мел-банка фильтров*. Важно отметить, что акустические признаки подвергаются некоторым процедурам предобработки перед их непосредственной передачей в блок построения дикторских моделей. В качестве этих процедур можно выделить: нормализация и масштабирование признаков, сохранение только речевых фреймов на основе разметки детектора речевой активности и т.п.

После того, как акустические признаки подготовлены, они могут быть переданы в блок построения дикторских моделей. Как правило, структура современных дикторских моделей соответствует структуре *x-векторных архитектур* [6]. Эти архитектуры состоят из четырёх ключевых уровней:

1. *Фреймовый уровень.* Предназначен для формирования локальных представлений голоса конкретного человека. На этом уровне как раз и применяются нейросетевые архитектуры на базе свёрточных нейронных сетей, например, ResNet, позволяющих с использованием каскадной схемы из множества фильтров с локальной маской захватить некоторый локальный контекст шаблона голоса человека. Выходом фреймового уровня является набор *высокоуровневых представлений* (*карт признаков*), содержащих локальные особенности голоса человека;
2. *Уровень статистического пулинга* позволяет сформировать промежуточный вектор признаков фиксированной длины. Эта длина является одинаковой для звукозаписи любой длительности. В ходе работы блока статистического пулинга происходит удаление временно*й* размерности, присутствующей в картах признаков. Это достигается путём выполнения процедуры усреднения карт признаков вдоль оси времени. Выходом уровня статистического пулинга являются *вектор среднего* и *вектор среднеквадратического отклонения*, вычисленные на основе карт признаков. Эти вектора конкатенируются в общий вектор и передаются для дальнейшей обработки на сегментном уровне;
3. *Сегментный уровень.* Предназначен для трансформации промежуточного вектора, как правило, высокой размерности, в компактный вектор признаков, представляющий собой дикторский эмбеддинг. Необходимо отметить, что на сегментном уровне расположены один или несколько полносвязных нейросетевых слоёв, а обработка данных выполняется по отношению ко всей звукозаписи, а не только по отношению к некоторому её локальному контексту, как на фреймовом уровне;
4. *Уровень выходного слоя.* Представляет полносвязный слой с softmax-функциями активации. Количество активаций равно числу дикторов в тренировочной выборке. На вход выходного слоя подаётся дикторский эмбеддинг, а на выходе формируется набор апостериорных вероятностей, определяющих принадлежность эмбеддинга к одному из дикторских классов в тренировочной выборке. Необходимо отметить, что, как правило, в современных нейросетевых системах построения дикторских моделей выходной слой используется только на этапе обучения параметров и на этапе тестирования не используется (на этапе тестирования используются только три первых указанных уровня).

Обучение модели генерации дикторских эмбеддингов выполняется путём решения задачи *классификации* [2, 7] или, выражаясь терминами из области биометрии, *идентификации на закрытом множестве* (количество дикторских меток является строго фиксированным) [4]. В качестве используемой стоимостной функции выступает *категориальная кросс-*

энтропия [2, 7]. Обучение выполняется с помощью мини-батчей, содержащих короткие фрагменты наборов акустических признаков (длительность несколько секунд) различных дикторов из тренировочной базы данных. Обучение на коротких фрагментах позволяет избежать сильного переобучения нейросетевой модели. При выполнении процедуры обучения требуется подобрать набор гиперпараметров, выбрать политику обучения и метод численной оптимизации.

Для выполнения настоящего пункта необходимо осуществить следующую последовательность действий:

1. Сгенерировать списки тренировочных, валидационных и тестовых данных на основе идентификационного протокола базы VoxCeleb1 (см. файл **`../data/voxceleb1_test/iden_split.txt`**). При генерации списков требуется исключить из них звукозаписи дикторов, которые входят в базу VoxCeleb1 test set. Это позволит выполнить тестирование обученных блоков генерации дикторских моделей на протоколе VoxCeleb1-O cleaned, который составлен по отношению к данным из VoxCeleb1 test set, в лабораторной работе № 4;
2. Инициализировать обучаемую дикторскую модель, выбрав любой возможный вариант её архитектуры, предлагаемый в рамках лабораторной работы. При реализации блока статистического пулинга предлагается выбрать либо его классический вариант [6], либо более продвинутую версию, основанную на использовании *механизмов внимания* [8]. Использование последней версии статистического пулинга позволяет реализовать детектор речевой активности прямо внутри блока построения дикторских моделей;
3. Инициализировать загрузчики тренировочной и валидационной выборки;
4. Инициализировать оптимизатор и планировщик для выполнения процедуры обучения;
5. Описать процедуру валидации / тестирования блока построения дикторских моделей;
6. Описать процедуру обучения и запустить её, контролируя значения стоимостной функции и доли правильных ответов на тренировочном множестве, а также долю правильных ответов на валидационном множестве.

Ниже представлен полный список гиперпараметров, которые необходимо настроить перед обучением модели блока генерации дикторских эмбеддингов.

```
# Select hyperparameters

# Acoustic features
n_mels = 40 # number of mel filters in bank filters
log_input = True # logarithm of features by level
```

```
# Neural network architecture
layers = [3, 4, 6, 3] # number of ResNet blocks in different
                      # level of frame level
activation = nn.ReLU  # activation function used in ResNet
                      # blocks
num_filters = [32, 64, 128, 256] # number of filters of ResNet
                                 # blocks in different level of
                                 # frame level
encoder_type = 'SP' # type of statistic pooling layer
                    # ('SP' - classical statistic pooling layer
                    # and 'ASP' - attentive statistic pooling)
nOut = 512 # embedding size

# Loss function for angular losses
margin = 0.35 # margin parameter
scale  = 32.0 # scale parameter

# Train dataloader
max_frames_train = 200 # number of frame to train
train_path = '../data/voxceleb1_dev/wav' # path to train wav
                                         # files
batch_size_train = 128 # batch size to train
pin_memory = False     # pin memory
num_workers_train = 5  # number of workers to train
shuffle = True         # shuffling of training examples

# Validation dataloader
max_frames_val = 1000 # number of frame to validate
val_path = '../data/voxceleb1_dev/wav' # path to val wav files
batch_size_val = 128 # batch size to validate
num_workers_val = 5  # number of workers to validate

# Test dataloader
max_frames_test = 1000 # number of frame to test
test_path = '../data/voxceleb1_dev/wav' # path to test wav files
batch_size_test = 128 # batch size to test
num_workers_test = 5  # number of workers to test

# Optimizer
lr = 2.5          # learning rate value
weight_decay = 0 # weight decay value

# Scheduler
val_interval = 5 # frequency of validation step
max_epoch = 40   # number of epochs

# Augmentation
musan_path = '../data/musan_split' # path to splitted
                                   # SLR17 dataset
rir_path = \
'../data/RIRS_NOISES/simulated_rirs' # path to SLR28
                                     # dataset
```

Программный код для генерации списков тренировочных, валидационных и тестовых данных представлен ниже.

```
# Generate data lists
train_list = []
val_list   = []
test_list  = []

with open('../data/voxceleb1_test/iden_split.txt', 'r') as f:
    lines = f.readlines()

black_list = \
os.listdir('../data/voxceleb1_test/wav') # exclude speaker IDs
                                         # from VoxCeleb1 test
                                         # set
num_train_spk = [] # number of train speakers

for line in lines:
    line   = line.strip().split(' ')
    spk_id = line[1].split('/')[0]

    if not (spk_id in black_list):
        num_train_spk.append(spk_id)

    else:
        continue

    # Train list
    if (line[0] == '1'):
        train_list.append(' '.join([spk_id, line[1]]))

    # Validation list
    elif (line[0] == '2'):
        val_list.append(' '.join([spk_id, line[1]]))

    # Test list
    elif (line[0] == '3'):
        test_list.append(' '.join([spk_id, line[1]]))

num_train_spk = len(set(num_train_spk))
```

С использованием программного кода ниже можно выполнить инициализацию модели, загрузчиков тренировочных и валидационных данных, а также оптимизатора и планировщика.

```
# Initialize model
model = ResNet(BasicBlock, \
               layers=layers, \
               activation=activation, \
               num_filters=num_filters, \
               nOut=nOut, \
```

```
                encoder_type=encoder_type, \
                n_mels=n_mels, \
                log_input=log_input)

trainfunc = AAMSoftmaxLoss(nOut=nOut, \
                           nClasses=num_train_spk, \
                           margin=margin, \
                           scale=scale)

main_model = MainModel(model, trainfunc).cuda()

# Initialize train dataloader (without augmentation)
train_dataset = \
train_dataset_loader(train_list=train_list, \
                     max_frames=max_frames_train, \
                     train_path=train_path)
train_loader = \
DataLoader(train_dataset, \
           batch_size=batch_size_train, \
           pin_memory=pin_memory, \
           num_workers=num_workers_train, \
           shuffle=shuffle)

# Initialize validation dataloader
val_dataset = \
test_dataset_loader(test_list=val_list, \
                    max_frames=max_frames_val, \
                    test_path=val_path)
val_loader = DataLoader(val_dataset, \
                        batch_size=batch_size_val, \
                        num_workers=num_workers_val)

# Initialize optimizer and scheduler
optimizer = SGDOptimizer(main_model.parameters(), \
                         lr=lr, \
                         weight_decay=weight_decay)
scheduler = \
OneCycleLRScheduler(optimizer, \
                    pct_start=0.30, \
                    cycle_momentum=False, \
                    max_lr=lr, \
                    div_factor=20, \
                    final_div_factor=10000, \
                    total_steps= \
                    max_epoch*len(train_loader))
```

Для запуска процедуры обучения модели экстрактора дикторских эмбеддингов можно воспользоваться примером программного кода ниже.

```
start_epoch = 0
checkpoint_flag = False
```

```
if checkpoint_flag:
    start_epoch = loadParameters(main_model, \
                  optimizer, \
                  scheduler, \
                  path='../data/lab3_models/checkpoint.pth')
    start_epoch = start_epoch + 1

# Train model
for num_epoch in range(start_epoch, max_epoch):
    train_loss, train_top1 = \
    train_network(train_loader, \
                  main_model, \
                  optimizer, \
                  scheduler, \
                  num_epoch, \
                  verbose=True)

    if (num_epoch + 1)%val_interval == 0:
        _, val_top1 = test_network(val_loader, \
                                   main_model)

        saveParameters(main_model, \
                       optimizer, \
                       scheduler, \
                       num_epoch, \
                       path='../data/lab3_models')
```

Отметим, что перед выполнением запуска процедуры обучения модели необходимо завершить программные коды из класса `ResNet` (реализация прямого прохода сигнала в нейросетевой модели), функции `train_network` (процедура обучения модели) и `test_network` (процедура тестирования модели) в файле **`./exercises_blank.py`**.

3. Обучение параметров блока построения дикторских моделей с учётом процедуры аугментации данных

Отметим, что процедура построения современных дикторских моделей основана на обучении глубоких нейронных сетей, требующих наличия больших объёмов данных для настройки их параметров. Поэтому возможным вариантом увеличения тренировочной выборки может являться использование методов аугментации статистических данных. Применение реалистичных методов аугментации к тренировочному множеству позволяет создать набор обучающих данных, с использованием которого можно повысить робастность блока генерации дикторских моделей при наличии шумов и помех.

Для выполнения процедуры аугментации данных в настоящей лабораторной работе предлагается использовать базы SLR17 [9] – MUSAN (корпус музыкальных, речевых и шумовых звукозаписей) и SLR28 [10] – RIR

noises (набор данных реальных и симулированных импульсных откликов комнат, а также изотропных и точечных шумов). Необходимо отметить, что перед применением методов аугментации, основанных на использовании баз подобных SLR17 и SLR28, к имеющимся данным, требуется удостовериться, что частоты дискретизации искажающих баз и оригинальных данных являются одинаковыми. Применительно к рассматриваемой лабораторной работе частоты дискретизации всех используемых звукозаписей должны быть равны 16 кГц.

Как известно, можно выделить *два режима выполнения аугментации данных*: *онлайн* (применяется в ходе процедуры обучения модели) и *оффлайн* (применяется до процедуры обучения модели). В рамках настоящей лабораторной работы предлагается использовать онлайн аугментацию в силу не очень большого набора тренировочных данных и большей гибкости экспериментов, чем в случае онлайн аугментации. Необходимо отметить, что применение онлайн аугментации на практике замедляет процедуру обучения в сравнении с оффлайн аугментацией, так как наложение искажений, извлечение акустических признаков и их возможная предобработка требуют определённого машинного времени.

В рамках настоящего пункта предлагается выполнить следующие действия:

1. Загрузить и извлечь данные из базы SLR17. Частота дискретизации данных в рассматриваемой базе равна 16 кГц по умолчанию. Поскольку звукозаписи рассматриваемой базы являются достаточно длинными, рекомендуется предварительно разбить эту базу на более маленькие фрагменты, например, длительностью 5 секунд с шагом 3 секунды, сохранив их на диск;
2. Загрузить и извлечь данные из базы SLR28. Частота дискретизации данных в рассматриваемой базе равна 16 кГц по умолчанию;
3. Модернизировать загрузчик тренировочных данных из класса `train_dataset_loader` в файле **`./exercises_blank.py`** под возможность случайного наложения (искажаем исходные звукозаписи) и отсутствия наложения (не искажаем исходные звукозаписи) одного из четырёх типов искажений (реверберация, музыкальный шум, фоновый шум голосов нескольких дикторов, неструктурированный шум), описанных внутри класса `AugmentWAV`, находящегося в следующем файле: **`../common/DatasetLoader.py`**;
4. Используя процедуру обучения из предыдущего пункта с идентичными настройками, выполнить тренировку параметров блока генерации дикторских моделей на исходных данных при наличии их аугментированных копий. При выполнении инициализации загрузчика тренировочных данных необходимо выставить формальный параметр `augment` в значение `True`, а также указать пути до баз искажений.

4. Тестирование блока построения дикторских моделей

Из научно-технической литературы известно, что применение алгоритмов машинного обучения на практике требует использования трёх наборов данных [7]: *тренировочное множество* (используется для обучения параметров модели), *валидационное множество* (используется для настройки гиперпараметров), *тестовое множество* (используется для итогового тестирования).

В рамках настоящего пункта предлагается выполнить итоговое тестирования блоков генерации дикторских моделей, обученных без аугментации и с аугментацией тренировочных данных, сравнив их между собой. При проведении процедуры тестирования рекомендуется выбрать различное количество фреймов для тестовых звукозаписей, чтобы грубо понять то, как длительность фонограммы влияет на качество распознавания диктора.

В качестве целевой метрики предлагается использовать *долю правильных ответов* (accuracy) [7], то есть количество верно классифицированных объектов по отношению к общему количеству объектов тестового множества. Как и при проведении процедуры обучения и валидации, рассматриваемая процедура тестирования предполагает решение задачи идентификации диктора на закрытом множестве.

Программный код для проведения итоговой процедуры тестирования модели представлен ниже. Для проведения тестирования необходимо инициализировать загрузчик тестовых данных, загрузить параметры обученной модели в её инициализированный экземпляр класса, выполнить вычисление целевой метрики оценки качества.

```
# Initialize test dataloader
test_dataset = \
test_dataset_loader(test_list=test_list, \
                    max_frames=max_frames_test, \
                    test_path=test_path)
test_loader = DataLoader(test_dataset, \
                         batch_size=batch_size_test, \
                         num_workers=num_workers_test)

# Load model
num_epoch = loadParameters(main_model, \
            optimizer, \
            scheduler, \
            path='../data/lab3_models/checkpoint.pth')

# Test model
_, test_top1 = test_network(test_loader, main_model)

print("Accuracy (test set) {:2.3f}%".format(test_top1))
```

Порядок защиты и представление результатов выполнения лабораторной работы

Для защиты лабораторной работы студент должен предоставить преподавателю самостоятельно завершённые программные коды из файла **`./exercises_blank.py`** и продемонстрировать их работоспособность путём поэтапного запуска файла **`./lab3_blank.ipynb`**. Преподаватель оставляет за собой право задать необходимые вопросы, затрагивающие программную реализацию заданий лабораторной работы. В процессе запуска файла **`./lab3_blank.ipynb`** студенту необходимо:

1. Предоставить результаты оценки качества на тренировочном и валидационном множествах для обученных моделей блока генерации дикторских эмбеддингов в случае наличия и отсутствия процедуры аугментации тренировочных данных. Сравнить полученные результаты и прокомментировать отличия между ними;
2. Предоставить результаты оценки качества на тестовом множестве для обученных моделей блока генерации дикторских эмбеддингов в случае наличия и отсутствия процедуры аугментации тренировочных данных. Сравнить полученные результаты и прокомментировать отличия между ними.

Контрольные вопросы

1. Что такое верификация и идентификация диктора?
2. Что такое распознавание диктора на закрытом и открытом множестве?
3. Что такое текстозависимое и текстонезависимое распознавание диктора?
4. Описать схему обучения блока генерации дикторских моделей на основе нейронных сетей.
5. Описать основные компоненты, из которых состоит нейросетевой блок генерации дикторских моделей (фреймовый уровень, слой статистического пулинга, сегментный уровень и выходной слой).
6. Что такое дикторский эмбеддинг и на каком уровне блока построения дикторских моделей он генерируется?
7. Как устроены нейросетевые архитектуры на основе ResNet-блоков?
8. Что такое полносвязная нейронная сеть прямого распространения?
9. Как устроена стоимостная функция для обучения нейросетевого блока генерации дикторских моделей?
10. Что такое аугментация данных?

Список литературы

1. Nagrani A., Chung J.S., Zisserman A. VoxCeleb: a large-scale speaker identification dataset // arXiv preprint arXiv:1706.08612 [cs.SD]. 2017.
2. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб.: Питер, 2018.

3. Gusev A., Volokhov V., Andzhukaev T., Novoselov S., Lavrentyeva G., Volkova M., Gazizullina A., Shulipa A., Gorlanov A., Avdeeva A., Ivanov A., Kozlov A., Pekhovsky T., Matveev Y. Deep speaker embeddings for far-field speaker recognition on short utterances // The Speaker and Language Recognition Workshop (Odyssey 2020). 2020. P. 179–186.
4. Bai Z., Zhang X.-L., Chen J. Speaker recognition based on deep learning: an overview // arXiv preprint arXiv:2012.00931 [eess.AS]. 2021.
5. Beigi H. Fundamentals of speaker recognition. Springer, 2011.
6. Snyder D., Garcia-Romero D., Sell G., Povey D., Khudanpur S. X-vectors: robust DNN embeddings for speaker recognition // 2018 IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP). 2018. P. 5329–5333.
7. Marsland S. Machine learning: an algorithmic perspective. Chapman and Hall, 2009.
8. Okabe K., Koshinaka T., Shinoda K. Attentive statistics pooling for deep speaker embedding // arXiv preprint arXiv:1803.10963 [eess.AS]. 2018.
9. Snyder D., Chen G., Povey D. MUSAN: a music, speech, and noise corpus // arXiv preprint arXiv:1510.08484 [cs.SD]. 2015.
10. Ko T., Peddinti V., Povey D., Seltzer M.L., Khudanpur S. A study on data augmentation of reverberant speech for robust speech recognition // 2017 IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP). 2017. P. 5220–5224.

Лабораторная работа № 4

КРИТЕРИИ ПРИНЯТИЯ РЕШЕНИЯ И МЕТРИКИ КАЧЕСТВА

Цель работы: изучение основных критериев принятия решения и метрик качества, рассматриваемых при решении задачи голосовой биометрии.

Краткое описание: в рамках настоящей лабораторной работы предлагается изучить и реализовать классические критерии принятия решения (критерий максимального правдоподобия, критерий максимума апостериорной вероятности, критерий Неймана-Пирсона, критерий Байеса и минимаксный критерий), позволяющие автоматически подобрать порог принятия решения для биометрической системы, а также оценить качество полученных биометрических решений по целевым метрикам в задаче верификации диктора на открытом множестве.

Данные: в качестве данных для выполнения лабораторной работы предлагается использовать тестовую часть базы VoxCeleb1.

Содержание лабораторной работы

1. Подготовка тестовых данных и протокола для проведения тестирования системы верификации диктора.
2. Вычисление дикторских моделей для эталонных и тестовых звукозаписей используемого протокола.
3. Формирование оценок сравнения эталонных и тестовых дикторских моделей.
4. Построение таргет- и импостор-гистограмм оценок сравнения моделей.
5. Вычисление порогового значения, вероятности ошибки принятия решения, вероятностей ошибок первого и второго рода при использовании критерия максимального правдоподобия, критерия максимума апостериорной вероятности, критерия Неймана-Пирсона, критерия Байеса и минимаксного критерия.
6. Вычисление значения равного уровня ошибок.

Порядок выполнения и краткая теория

1. Подготовка тестовых данных и протокола для проведения тестирования системы верификации диктора

Перед выполнением лабораторной работы требуется импортировать некоторые Python-модули.

```
# Import of modules
import os
import sys

sys.path.append(os.path.realpath('..'))

from math import sqrt, pi

import numpy as np
from matplotlib.pyplot import hist, plot, show, grid, title, \
xlabel, ylabel, legend, axis, imshow
import torch
from torch.utils.data import DataLoader
import itertools

from common import download_dataset, extract_dataset, \
download_protocol
from common import test_dataset_loader
from common import get_eer
from common import extract_features, compute_scores
from LoadModel import load_model
from ResNetSE34V2 import MainModel
from exercises_blank import tar_imp_hists, llr, map_test, \
neyman_pearson_test, bayes_test, minmax_test
```

В ходе выполнения лабораторной работы необходимы тестовые данные, а также связанный с ними протокол тестирования. *Тестовые данные* представляют собой звукозаписи голосов людей мужского и женского пола, сохраненных в формат wav, выбранных из корпуса VoxCeleb1 test set [1]. Данный корпус содержит 4874 звукозаписи (частота дискретизации равна 16 кГц) 40 дикторов мужского и женского пола, разговаривающих преимущественно на английском языке. Отметим, что рассматриваемые тестовые данные были исключены, в смысле меток дикторов и звукозаписей, из тренировочных и валидационных данных, используемых при обучении и проверке качества экстрактора дикторских моделей в лабораторной работе № 3. Последнее связано с тем, что в рамках настоящей лабораторной работы предлагается выполнить тестирование обученного ранее экстрактора дикторских моделей применительно к задаче верификации диктора на открытом множестве [2]. *Верификационный протокол тестирования* VoxCeleb1-O cleaned [3] в настоящей лабораторной работе организован в виде текстового файла, в каждой строчке которого содержится одна таргет- или импостор-попытка [4] сравнения эталонной и тестовой звукозаписей. Голоса эталона и теста из *таргет-попытки* (метка «1») соответствуют одному и тому же диктору, а голоса эталона и теста из *импостор-попытки* (метка «0») соответствуют разным дикторам.

В рамках настоящего пункта требуется выполнить загрузку и распаковку звуковых wav-файлов из корпуса VoxCeleb1 test set, а также загрузку верификационного протокола тестирования VoxCeleb1-O cleaned.

Предположим, что здесь и далее в рамках настоящей работы рабочим директорием является **./lab4**, находящийся внутри репозитория с лабораторными работами, и все пути до директориев и файлов указаны относительно него. Загрузку и распаковку данных можно выполнить с использованием программного кода, представленного ниже.

```
# Download VoxCeleb1 test set
with open('../data/lists/datasets.txt', 'r') as f:
    lines = f.readlines()

download_dataset(lines, \
                 user='voxceleb1902', \
                 password='nx0bl2v2', \
                 save_path='../data')

# Extract VoxCeleb1 test set
extract_dataset(save_path='../data/voxceleb1_test', \
                fname='../data/vox1_test_wav.zip')

# Download VoxCeleb1-O cleaned protocol
with open('../data/lists/protocols.txt', 'r') as f:
    lines = f.readlines()

download_protocol(lines, save_path='../data/voxceleb1_test')
```

2. Вычисление дикторских моделей для эталонных и тестовых звукозаписей используемого протокола

Эталонные и тестовые звукозаписи таргет- и импостор-попыток из тестового протокола неудобно сравнивать напрямую. Рассматриваемые в лабораторной работе данные содержат звукозаписи с частотой дискретизации 16 кГц. Последнее означает, что в 1 секунде записи содержится 16000 отсчётов цифрового сигнала, а в двух секундах – 32000 и т.п. Сравнивать между собой длинные вектора не самая удачная идея, так как это является затратным в смысле вычислительной сложности. Дополнительно необходимо отметить, что «сырой» звуковой сигнал содержит не только голос диктора, а помехи и шумы, воздействие акустики помещения и т.п. Поэтому на практике при решении задачи распознавания диктора для эталонной и тестовой звукозаписей вычисляются *дикторские модели* [2, 4], содержащие внутри себя основные характеристики голоса конкретного диктора. Построить дикторскую модель можно различными способами, например, с использованием специально обученной искусственной нейронной сети [2], на вход которой передаётся набор акустических признаков, вычисленных для определенной звукозаписи, а на выходе формируется вектор малой размерности, например, длины 128, 256 и т.п. Этот вектор, иногда называемый *эмбеддингом* [2], и является дикторской моделью.

Ниже представлена последовательность действий, которую необходимо выполнить при рассмотрении настоящего пункта:

1. Описать и инициализировать модель для генерации дикторских эмбеддингов;
2. Выполнить загрузку обученных весов в инициализированную модель. В качестве модели и обученных весов для неё рекомендуется использовать модель и обученные веса из лабораторной работы № 3;
3. Вычислить эмбеддинги для протокольных эталонных и тестовых звукозаписей.

Ниже представлен программный код, инициализирующий модель и загружающий в неё обученные веса.

```
# Model initialization
model = MainModel()

# Load trained model
with open('../data/lists/models.txt', 'r') as f:
    lines = f.readlines()

model = load_model(model, lines, save_path='../data/models')
model.eval()
model = model.cuda()
```

Ниже представлен программный код, позволяющий загрузить верификационный протокол тестирования, инициализировать загрузчик тестовых данных, а также выполнить вычисление дикторских моделей для каждой тестовой звукозаписи.

```
# Load test protocol
with open('../data/voxceleb1_test/veri_test2.txt', 'r') as f:
    lines = f.readlines()

# Get a list of unique file names
files = \
list(itertools.chain(*[x.strip().split()[-2:] for x in lines]))
setfiles = list(set(files))
setfiles.sort()

# Define test data loader
test_dataset = test_dataset_loader(setfiles, \
               test_path='../data/voxceleb1_test/wav/', \
               eval_frames=200, \
               num_eval=5)
test_loader  = torch.utils.data.DataLoader(test_dataset, \
                                           batch_size=1, \
                                           shuffle=False, \
                                           num_workers=10, \
                                           drop_last=False, \
```

```
                                                sampler=None)

# Extract features for every waveform
feats = extract_features(model, test_loader)
```

3. Формирование оценок сравнения эталонных и тестовых дикторских моделей

Для того чтобы автоматически принять решение о принадлежности пары сравнения «эталон-тест» к таргет- или импостор-попытке, необходимо сформировать оценку сравнения (score) [2, 4] между эталонной и тестовой дикторскими моделями. В простейшем случае подобные сравнения можно выполнить с использованием *евклидовой метрики*, *косинусной метрики* и т.п.

В рамках настоящего пункта необходимо выполнить следующее:

1. Осуществить формирование оценок сравнения эталонных и тестовых дикторских моделей с использованием любой подходящей метрики;
2. Сформировать списки оценок сравнения таргет- / импостор-меток и имён «эталон-тест» в триальных сравнениях.

Ниже представлен вызов функции `compute_scores`, реализующей шаги выше и определённой в файле **`../common/scoring.py`**.

```
# Compute scores between enroll and test speaker models
all_scores, \
all_labels, \
all_trials = compute_scores(feats, lines)
```

4. Построение таргет- и импостор-гистограмм оценок сравнения моделей

После вычисления оценок сравнения для таргет- и импостор-попыток из протокола тестирования можно провести первоначальную оценку качества работы системы распознавания диктора путём визуального анализа гистограмм распределения таргет- и импостор-оценок. Величина наложения гистограмм друг на друга определяет это качества. Чем сильнее друг на друга наложены гистограммы, тем качество работы системы распознавания диктора ниже и, наоборот, чем меньше друг на друга наложены гистограммы, тем качество работы системы распознавания диктора выше.

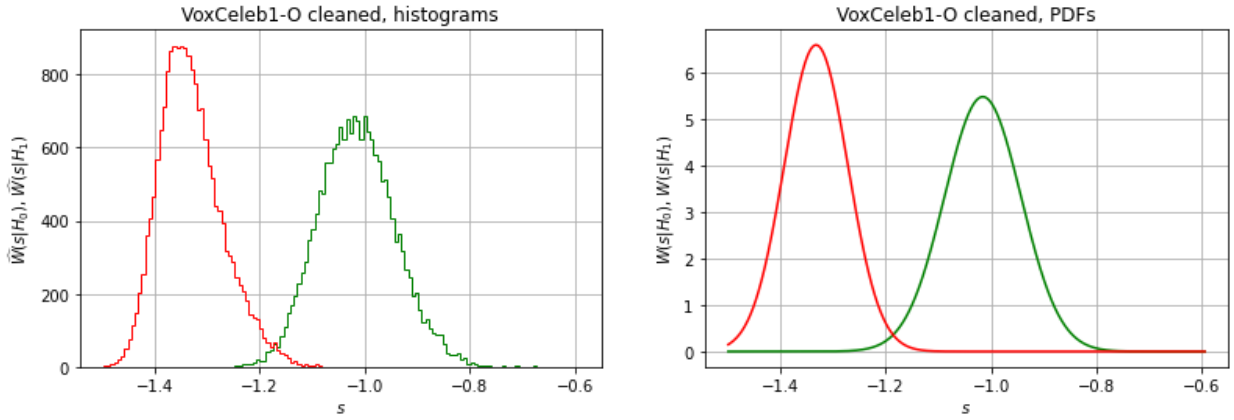
В рамках настоящего пункта необходимо выполнить следующее:

1. Разделить оценки сравнения на два набора: таргет-сравнения и импостор-сравнения.
2. Построить гистограммы для таргет-оценок и импостор-оценок.

Ниже представлен программный код, позволяющий выполнить шаги выше. Для запуска кода необходимо завершить функцию `tar_imp_hists` (вычисление таргет- и импостор-оценок) из **`./exercises_blank.py`**.

```
# Compute target and impostor scores
tar_scores, imp_scores = tar_imp_hists(all_scores, all_labels)

# Plot histograms for target and impostor scores
min_scores = np.concatenate((tar_scores, imp_scores)).min()
max_scores = np.concatenate((tar_scores, imp_scores)).max()

hist(tar_scores, \
     int(sqrt(len(tar_scores))), \
     histtype='step', \
     color='green', \
     range=(min_scores, max_scores))
hist(imp_scores, \
     int(sqrt(len(imp_scores))), \
     histtype='step', \
     color='red', \
     range=(min_scores, max_scores))
xlabel('$s$')
ylabel('$\widehat{W}(s|H_0)$, $\widehat{W}(s|H_1)$')
title('VoxCeleb1-O cleaned, histograms'); grid(); show()
```

Результат выполнения программного кода выше представлен на рисунке 1а.



а) б)

Рисунок 1 – Гистограммы распределения таргет- (справа) и импостор-оценок (слева) представлены на рисунке 1а. Аппроксимации гистограмм, полученные с использованием гауссовских распределений, представлены на рисунке 1б. Гистограммы и их аппроксимации получены применительно к протоколу VoxCeleb1-O cleaned для исследуемой нейросетевой модели генерации дикторских эмбеддингов

5. Вычисление порогового значения, вероятности ошибки принятия решения, вероятностей ошибок первого и второго рода

Выражаясь терминами *бинарной теории принятия решений* [5–7], предположение о наличии таргет-пары при решении задачи распознавания диктора связывается с *нулевой гипотезой* – H_0 [5–7], а о наличии импостор-

пары связывается с *альтернативной гипотезой* – H_1 [5–7]. Когда задача распознавания диктора рассматривается в качестве верификации, в системе возможно возникновение следующих ситуаций:

1. Пара сравнения «эталон-тест» является таргет-парой, решение системы принимается в пользу таргет-пары – событие (D_0, H_0). Подобную ситуацию называют *правильным положительным решением* (true positive) [5];
2. Пара сравнения «эталон-тест» является таргет-парой, решение системы принимается в пользу импостор-пары – событие (D_1, H_0). Подобную ситуацию называют *ложным отрицательным решением* (false negative) [5]. Говорят, что в этом случае возникает *ошибка первого рода*;
3. Пара сравнения «эталон-тест» является импостор-парой, решение системы принимается в пользу таргет-пары – событие (D_0, H_1). Подобную ситуацию называют *ложным положительным решением* (false positive) [5]. Говорят, что в этом случае возникает *ошибка второго рода*;
4. Пара сравнения «эталон-тест» является импостор-парой, решение системы принимается в пользу импостор-пары – событие (D_1, H_1). Подобную ситуацию называют *правильным отрицательным решением* (true negative) [5].

Как правило, перед принятием решения значение оценки сравнения эталона и теста представляется в виде натурального *логарифма отношения правдоподобия* (LLR, log-likelihood ratio) [4–7]:

$$\mathrm{LLR} = \ln[W(s|H_0)/W(s|H_1)],$$

здесь s – величина оценки сравнения эталона и теста, а $W(s|H_0)$ и $W(s|H_1)$ – плотности вероятности, описывающие распределение оценок сравнения для таргет- и импостор-попыток. Обычно подобные плотности вероятности рассматриваются в виде одномерных гауссовских плотностей, параметры математического ожидания и среднеквадратического отклонения которых оцениваются с использованием некоторого набора данных так:

$$m_{H_i} = \frac{1}{N_{H_i}}\sum_{i=0}^{N_{H_i}-1} s(i),\ \sigma_{N_{H_i}} = \sqrt{\frac{1}{N_{H_i}}\sum_{i=0}^{N_{H_i}-1}\left(s(i) - m_{N_{H_i}}\right)^2},\ i = 0, 1,$$

здесь N_{H_i} – количество таргет- / импостор-пар в верификационном протоколе, используемом для оценки параметров плотностей вероятности $W(s|H_0)$ и $W(s|H_1)$. В простейшем случае этот протокол может совпадать с тестовым протоколом, но ожидания по качеству работы системы распознавания диктора в этом случае будут завышенными! Поэтому на практике протокол верификации для оценки параметров распределений $W(s|H_0)$ и $W(s|H_1)$ выбирается отличным от протокола тестирования (см. решение задачи калибровки в лабораторной работе № 5).

Для принятия автоматического решения биометрической системой требуется подобрать величину *порога принятия решения* [4–7]. В случае, если значение LLR для сравниваемой пары «эталон-тест» будет выше порога, решение биометрической системой принимается в пользу таргет-пары, и, наоборот, если значение LLR для сравниваемой пары «эталон-тест» будет ниже порога, решение биометрической системой принимается в пользу импостор-пары.

Бинарная теория принятия решений говорит о том, что порог принятия решения может быть выбран несколькими различными способами [5]: критерий максимального правдоподобия, критерий максимума апостериорной вероятности (критерий минимума вероятности ошибки), критерий Неймана-Пирсона, критерий Байеса и минимаксный критерий.

Критерий максимального правдоподобия [5] предполагает, что априорные вероятности $P(H_0)$ и $P(H_1)$ выпадения таргет- и импостор-пар являются одинаковыми. Математически критерий может быть сформулирован следующим образом: если $\mathrm{LLR} > 0$, то решение принимается в пользу таргет-пары, и, наоборот, если $\mathrm{LLR} < 0$, то решение принимается в пользу импостор-пары. Величина порога принятия решения в этом случае равна 0.

Критерий максимума апостериорной вероятности [5] является более общим, чем критерий максимального правдоподобия, и учитывает возможность неравенства априорных вероятностей выпадения таргет- и импостор-пар. Математически критерий может быть сформулирован следующим образом: если $\mathrm{LLR} > \ln[P(H_1)/P(H_0)]$, то решение принимается в пользу таргет-пары, и, наоборот, если $\mathrm{LLR} < \ln[P(H_1)/P(H_0)]$, то решение принимается в пользу импостор-пары. Величина порога принятия решения в этом случае равна $\ln[P(H_1)/P(H_0)]$. Иногда рассматриваемый критерий называют *критерием минимума вероятности ошибки*, поскольку можно математически доказать, что рассматриваемый критерий следует из минимизации выражения для *вероятности ошибки системы принятия решения* [5]:

$$P_e = P(D_1, H_0) + P(D_0, H_1) = P(D_1|H_0)P(H_0) + P(D_0|H_1)P(H_1),$$

где $P(D_1, H_0)$ – вероятность события (D_1, H_0), а $P(D_0, H_1)$ – вероятность события (D_0, H_1). Вероятности $P(D_1|H_0)$ и $P(D_0|H_1)$ называют *вероятностями ошибок первого* (FRR, false rejection rate) [4, 5, 7] и *второго рода* (FAR, false acceptance rate) [4, 5, 7].

Критерий Неймана-Пирсона [5] предполагает, что одна из вероятностей ошибок, например, первого рода, строго задана. Величина порога принятия решения выбирается в точке, для которой вероятность другой ошибки, например, второго рода, будет минимальна.

В основе *критерия Байеса* [5] лежит минимизация *средней стоимости* или *байесовского риска* [5], который математически описывается следующим образом:

$$\bar{C} = C_{00}P(D_0|H_0)P(H_0) + C_{10}P(D_1|H_0)P(H_0) + C_{01}P(D_0|H_1)P(H_1) + C_{11}P(D_1|H_1)P(H_1),$$

где C_{00}, C_{10}, C_{01} и C_{11} – скалярные величины, описывающие стоимости, связанные с событиями (D_0, H_0), (D_1, H_0), (D_0, H_1) и (D_1, H_1) соответственно. Минимизация средней стоимости позволяет следующим образом описать критерий Байеса: если $\text{LLR} > \ln[(C_{01} - C_{11}) \cdot P(H_1)/[(C_{10} - C_{00}) \cdot P(H_0)]]$, то решение принимается в пользу таргет-пары, и, наоборот, если $\text{LLR} < \ln[(C_{01} - C_{11}) \cdot P(H_1)/[(C_{10} - C_{00}) \cdot P(H_0)]]$, то решение принимается в пользу импостор-пары. Величина порога принятия решения в этом случае равна $\ln[(C_{01} - C_{11}) \cdot P(H_1)/[(C_{10} - C_{00}) \cdot P(H_0)]]$.

При рассмотрении практических задач возможна ситуация, при которой априорные вероятности выпадения таргет- и импостор-пар являются неизвестными. В этом случае выбор порога принятия решения может быть выполнен с использованием *минимаксного критерия* [5, 8]. В случае минимаксного критерия выражение для среднего риска максимизируется по $P(H_0)$, а затем для найденного значения $P(H_0)$ минимизируется так, как в критерии Байеса.

В рамках настоящего пункта требуется выполнить следующие задания:

1. Вычислить логарифм отношение правдоподобия для всех оценок сравнения «эталон-тест» из рассматриваемого протокола тестирования. При решении данной задачи необходимо выполнить сортировку значений в порядке возрастания внутри одномерного массива, содержащего все оценки сравнения эталонов и тестов. Дополнительно рекомендуется синхронно с массивом выше сформировать одномерный массив, содержащий идеальную разметку сравнений эталонов и тестов;
2. Вычислить пороговое значение, вероятность ошибки принятия решения, вероятности ошибок первого и второго рода на основе оценок сравнения моделей в случае использования критерия максимального правдоподобия и критерия максимума апостериорной вероятности;
3. Вычислить пороговое значение, вероятность ошибки второго рода для заданной ошибки первого рода на основе оценок сравнения моделей в случае использования критерия Неймана-Пирсона;
4. Вычислить пороговое значение, минимум средней стоимости, вероятности ошибок первого и второго рода на основе оценок сравнения моделей в случае использования критерия Байеса;
5. Вычислить пороговое значение, минимум максимума средней стоимости, вероятности ошибок первого и второго рода, а также вероятность таргет-пары на основе оценок сравнения моделей в случае использования минимаксного критерия.

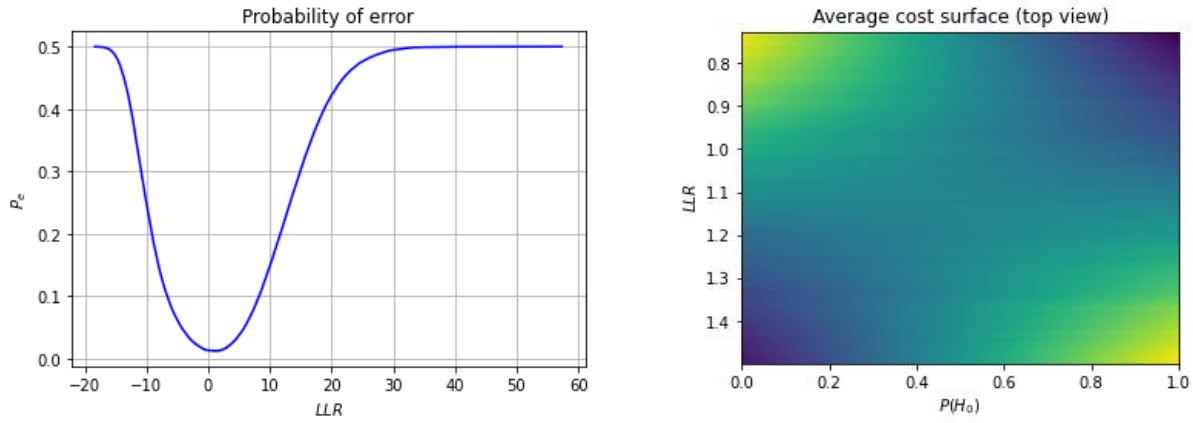
С использованием представленного ниже программного кода можно выполнить расчёт логарифма отношения правдоподобия для всех оценок

сравнения «эталон-тест» из рассматриваемого протокола тестирования, а также построить гауссовские аппроксимации гистограмм распределения этих оценок (см. рисунок 1б). Для выполнения рассматриваемого кода необходимо завершить функцию `llr` (вычисление логарифма отношения правдоподобия) в файле **`./exercises_blank.py`**.

```
# Gaussian probability density function
gauss_pdf = lambda score, mu, sigma: 1/((abs(sigma) + \
            1e-10)*sqrt(2*pi))*np.exp(-(score - \
            mu)**2/(2*(abs(sigma) + 1e-10)**2))

# Compute log-likelihood ratio
ground_truth_sort, \
all_scores_sort, \
tar_gauss_pdf, \
imp_gauss_pdf, \
LLR = llr(all_scores, all_labels, tar_scores, \
      imp_scores, gauss_pdf)

# Plot PDFs for target and impostor scores
plot(all_scores_sort, tar_gauss_pdf, color='green')
plot(all_scores_sort, imp_gauss_pdf, color='red')
xlabel('$s$'); ylabel('$W(s|H_0)$, $W(s|H_1)$');
title('VoxCeleb1-O cleaned, PDFs'); grid(); show()
```

Выполнение программного кода ниже позволяет осуществить изучение критерия максимума апостериорной вероятности, который при значении $P(H_0) = 0{,}5$ преобразуется в критерий максимального правдоподобия. Для выполнения рассматриваемого кода необходимо завершить функцию `map_test` (реализация критерия максимума апостериорной вероятности) в файле **`./exercises_blank.py`**. Дополнительно на рисунке 2а представлен пример зависимости вероятности ошибки от порога принятия решения. Данную зависимость можно получить в ходе исследования критерия максимума апостериорной вероятности для $P(H_0) = 0{,}5$.

```
# Maximum-likelihood test and maximum a posteriori (MAP) test.
# MAP is the same as Minimum Probability of Error Test
P_Htar = 1/2 # prob. of target hypothesis

llr_thr_val, fnr_thr_val, fpr_thr_val, P_err_min = \
map_test(ground_truth_sort, LLR, tar_scores, imp_scores, P_Htar)

print('Threshold value:       {0:.3f}'.format(llr_thr_val))
print('False negative rate:   {0:.3f}'.format(fnr_thr_val))
print('False positive rate:   {0:.3f}'.format(fpr_thr_val))
print('Probability of error: {0:.3f}'.format(P_err_min))
```



а) б)

Рисунок 2 – Пример изменения вероятности ошибки в зависимости от порога принятия решения (рисунок 2а) и пример поверхности средней стоимости (вид сверху), имеющей форму «седла» (рисунок 2б)

Выполнение программного кода ниже позволяет осуществить изучение критерия Неймана-Пирсона. Для выполнения рассматриваемого кода необходимо завершить функцию `neyman_pearson_test` (реализация критерия Неймана-Пирсона) в файле **`./exercises_blank.py`**.

```
# Neyman-Pearson test
fnr     = 1/4 # given prob. of Type I error P(Dimp|Htar),
              #false negative rate
llr_thr_val, fpr_thr_val = \
neyman_pearson_test(ground_truth_sort, LLR, tar_scores, \
                    imp_scores, fnr)

print('Threshold value:       {0:.3f}'.format(llr_thr_val))
print('False negative rate:   {0:.3f}'.format(fnr))
print('False positive rate:   {0:.3f}'.format(fpr_thr_val))
```

Выполнение программного кода ниже позволяет осуществить изучение критерия Байеса. Для выполнения рассматриваемого кода необходимо завершить функцию `bayes_test` (реализация критерия Байеса) в файле **`./exercises_blank.py`**.

```
# Bayes' test
P_Htar = 1/2 # prob. of target hypothesis
C00    = 0   # cost of event (Dtar, Htar)
C10    = 1   # cost of event (Dimp, Htar)
C01    = 1   # cost of event (Dtar, Himp)
C11    = 0   # cost of event (Dimp, Himp)

llr_thr_val, fnr_thr_val, fpr_thr_val, AC_val = \
bayes_test(ground_truth_sort, LLR, tar_scores, imp_scores, \
           P_Htar, C00, C10, C01, C11)
```

```
print('Threshold value:          {0:.3f}'.format(llr_thr_val))
print('False negative rate:      {0:.3f}'.format(fnr_thr_val))
print('False positive rate:      {0:.3f}'.format(fpr_thr_val))
print('Minimum of average cost: {0:.3f}'.format(AC_val))
```

Выполнение программного кода ниже позволяет осуществить изучение минимаксного критерия. Для выполнения рассматриваемого кода необходимо завершить функцию `minmax_test` (реализация минимаксного критерия) в файле **`./exercises_blank.py`**. На рисунке 2б представлен пример имеющей форму «седла» поверхности функции средней стоимости. В центре этой поверхности находится *седловая точка*, определяющая решение минимаксной задачи оптимизации.

```
# Minimax test
P_Htar_thr = np.linspace(0, 1, num=1000) # set of prob. of
                                         # target hypothesis
C00 = 0 # cost of event (Dtar, Htar)
C10 = 1 # cost of event (Dimp, Htar)
C01 = 1 # cost of event (Dtar, Himp)
C11 = 0 # cost of event (Dimp, Himp)

llr_thr_val, \
fnr_thr_val, \
fpr_thr_val, \
AC_val, \
P_Htar_thr_val = minmax_test(ground_truth_sort, LLR, \
                             tar_scores, imp_scores, \
                             P_Htar_thr, C00, C10, C01, C11)

print('Threshold value:          {0:.3f}'.format(llr_thr_val))
print('False negative rate:      {0:.3f}'.format(fnr_thr_val))
print('False positive rate:      {0:.3f}'.format(fpr_thr_val))
print('Minmax of average cost: {0:.3f}'.format(AC_val))
print('Probability of Htar:      {0:.3f}'.format(P_Htar_thr_val))
```

6. Вычисление значения равного уровня ошибок

Оценивать качество работы системы распознавания диктора визуально по наложению гистограмм для таргет- и импостор-оценок сравнения эталона и теста не очень удобно. Особые сложности возникают тогда, когда количество сравниваемых систем очень большое. Поэтому можно воспользоваться некоторыми численными метриками автоматической оценки качества, например, вероятностью ошибки, как это было выполнено выше. В предположении, что априорные вероятности выпадения таргет- и импостор-пар сравнения эталонов и тестов являются одинаковыми, а порог принятия решения выбран в точке, для которой вероятности ошибок первого и второго рода одинаковые, вероятность ошибки переходит в *метрику равного уровня ошибок* (EER, equal error rate) [4, 7].

Рассмотрим вычисление величины равного уровня ошибок для используемого протокола тестирования VoxCeleb1-O cleaned. Вызов функции для подсчёта равного уровня ошибок `get_eer`, определённой в файле **`../common/perf.py`**, представлен ниже.

```
# Compute Equal Error Rate
EER, _ = get_eer(tar_scores, imp_scores)

print('Equal Error Rate (EER): {0:.3f}%'.format(EER))
```

Порядок защиты и представление результатов выполнения лабораторной работы

Для защиты лабораторной работы студент должен предоставить преподавателю самостоятельно завершённые программные коды из файла **`./exercises_blank.py`** и продемонстрировать их работоспособность путём поэтапного запуска файла **`./lab4_blank.ipynb`**. Преподаватель оставляет за собой право задать необходимые вопросы, затрагивающие программную реализацию заданий лабораторной работы. В процессе запуска файла **`./lab4_blank.ipynb`** студенту необходимо:

1. Визуализировать гистограммы распределения и их аппроксимации для таргет- и импостор-оценок сравнения эталонных и тестовых звукозаписей из рассматриваемого протокола тестирования для выбранной модели блока генерации дикторских эмбеддингов. Выполнить предварительную оценку качества биометрического решения по степени наложения гистограмм;
2. Оценить качество полученного биометрического решения для различных критериев автоматического выбора порога принятия решения. Сравнить полученные оценки между собой;
3. Привести значение метрики равного уровня ошибок для оценки качества рассматриваемого биометрического решения.

Контрольные вопросы

1. Как устроен протокол тестирования системы верификации?
2. Что такое таргет- и импостор-пара?
3. Какие подходы к сравнению дикторских моделей вам известны?
4. Что такое критерий принятия решения и какими они бывают (максимального правдоподобия, максимума апостериорной вероятности, Неймана-Пирсона, байесовский, минимаксный)?
5. Что такое ошибки первого и второго рода и как вычислить их вероятности применительно к задаче распознавания диктора?
6. Что такое равный уровень ошибок?

7. Что такое кривая функционирования приемника (ROC, receiver operating characteristic) и кривая компромисса между ошибками детектирования (DET, detection error tradeoff)?

Список литературы

1. Nagrani A., Chung J.S., Zisserman A. VoxCeleb: a large-scale speaker identification dataset // arXiv preprint arXiv:1706.08612 [cs.SD]. 2017.
2. Bai Z., Zhang X.-L., Chen J. Speaker recognition based on deep learning: an overview // arXiv preprint arXiv:2012.00931 [eess.AS]. 2021.
3. Nagrani A., Chung J.S., Xie W., Zisserman A. Voxceleb: large-scale speaker verification in the wild // Computer Speech & Language. 2020. V. 60. P. 101027.
4. Hansen J.H.L., Hasan T. Speaker recognition by machines and humans: a tutorial review // IEEE Signal Processing Magazine. 2015. V. 32, № 6. P. 74–99.
5. Hsu H.P. Schaum's Ouline of Theory and Problems of Probability, Random Variables, and Random Processes. McGraw-Hill, 1997.
6. Hsu H.P. Schaum's Ouline of Analog and Digital Communications. McGraw-Hill, 2003.
7. Beigi H. Fundamentals of speaker recognition. Springer, 2011.
8. Босс В. Лекции по математике. Т. 7: оптимизация: учебное пособие. Изд. 3-е. – М.: Книжный дом «ЛИБРОКОМ», 2010.

Лабораторная работа № 5

АДАПТАЦИЯ И КАЛИБРОВКА СИСТЕМЫ РАСПОЗНАВАНИЯ ДИКТОРА

Цель работы: изучение методов адаптации и калибровки, используемых на практике при решении задачи голосовой биометрии.

Краткое описание: в рамках настоящей лабораторной работы предлагается изучить и реализовать алгоритмы адаптации и калибровки биометрических систем. В качестве процедуры адаптации предлагается рассмотреть адаптацию на основе s-нормализации. Процедуру калибровки предлагается реализовать с помощью метода логистической регрессии. Тестирование системы голосовой биометрии после выполнения процедур адаптации и калибровки предлагается выполнить на основе метрик равного уровня ошибок и минимума функции стоимости детектирования применительно к задаче верификации на открытом множестве.

Данные: в качестве данных для выполнения лабораторной работы предлагается использовать тестовые части баз VoxCeleb1 и VoxCeleb2.

Содержание лабораторной работы

1. Подготовка тестовых данных и протокола для проведения тестирования системы верификации диктора.
2. Вычисление дикторских моделей для эталонных и тестовых звукозаписей используемого протокола.
3. Формирование оценок сравнения эталонных и тестовых дикторских моделей.
4. Доменная адаптация на основе s-нормализации.
5. Калибровка системы голосовой биометрии.

Порядок выполнения и краткая теория

1. Подготовка тестовых данных и протокола для проведения тестирования системы верификации диктора

Перед выполнением лабораторной работы требуется импортировать некоторые Python-модули.

```
# Import of modules
import os
import sys
```

```
import torch
import numpy as np
import itertools

from math import sqrt, pi
from matplotlib.pyplot import hist, plot, show, grid, title, \
xlabel, ylabel, legend, axis
from torch.utils.data import DataLoader
from functools import partial
import glob

import common
from common import download_dataset, extract_dataset, \
download_protocol, run_voxceleb_convert, part_extract
from common import get_voxceleb_filelist
from common import test_dataset_loader
from common import extract_features, compute_scores_cosine
from common import get_eer, get_dcf
from common import concatenate
from LoadModel import load_model
from ResNetSE34V2 import MainModel

from exercises_blank import plot_histograms_2sets, \
s_norm, get_tar_imp_scores, train_calibration
from exercises_blank import CalibrationLoss, \
LinearCalibrationModel, CalibrationDataset, train_calibration
from Optimizer import SGDOptimizer
from Scheduler import StepLRScheduler
```

В ходе выполнения лабораторной работы необходимы тестовые данные, а также связанный с ними протокол тестирования. *Тестовые данные* представляют собой звукозаписи голосов людей мужского и женского пола, сохраненных в формат wav, выбранных из корпуса VoxCeleb1 test set [1]. Данный корпус содержит 4874 звукозаписи (частота дискретизации равна 16 кГц) 40 дикторов мужского и женского пола, разговаривающих преимущественно на английском языке. Отметим, что рассматриваемые тестовые данные были исключены, в смысле меток дикторов и звукозаписей, из тренировочных и валидационных данных, используемых при обучении и проверке качества экстрактора дикторских моделей в лабораторной работе № 3. Последнее связано с тем, что в рамках настоящей лабораторной работы предлагается выполнить тестирование обученного ранее экстрактора дикторских моделей применительно к задаче верификации диктора на открытом множестве [2] при использовании процедур адаптации [2, 3] и калибровки [4, 5]. *Верификационный протокол тестирования* VoxCeleb1-O cleaned [1] в настоящей лабораторной работе организован в виде текстового файла, в каждой строчке которого содержится одна таргет- или импостор-попытка сравнения эталонной и тестовой звукозаписей. Голоса эталона и теста из *таргет-попытки* (метка «1») соответствуют одному и тому же

диктору, а голоса эталона и теста из *импостор-попытки* (метка «0») соответствуют разным дикторам. Отметим, что в качестве дополнительного множества данных, которое в настоящей лабораторной работе используется как адаптационное и калибровочное множества, рассматривается корпус VoxCeleb2 test set [1]. Данный корпус содержит звукозаписи 36237 звукозаписей 118 дикторов мужского и женского пола, разговаривающих преимущественно на английском языке. Звукозаписи сохранены в формате m4a и, для удобства работы с ними средствами Python, должны быть преобразованы к формату wav. Отметим, что метки дикторов в корпусах VoxCeleb1 и VoxCeleb2 не пересекаются между собой.

В рамках настоящего пункта требуется осуществить следующую последовательность действий:

1. Выполнить загрузку и распаковку звуковых wav-файлов баз VoxCeleb1 test set и VoxCeleb2 test set;
2. Преобразовать базу VoxCeleb2 test set из m4a-формата в wav-формат;
3. Выполнить загрузку протокола тестирования VoxCeleb1-O cleaned;
4. Подготовить тестовые и адаптационные данные путём применения к оригинальным звукозаписям MP3-кодека. В качестве тестовых данных предлагается рассмотреть данные, полученные из базы VoxCeleb1 test set, а в качестве адаптационных – из базы VoxCeleb2 test set.

Предположим, что здесь и далее в рамках настоящей работы рабочим директорием является **`./lab5`**, находящийся внутри репозитория с лабораторными работами, и все пути до директориев и файлов указаны относительно него. Описанную выше последовательность действий можно выполнить с использованием программных кодов, представленных ниже.

Вначале необходимо выполнить загрузку и распаковку файлов баз VoxCeleb1 test set и VoxCeleb2 test set, а также загрузить протокол тестирования VoxCeleb1-O cleaned.

```
# Download VoxCeleb1 test set
with open('../data/lists/datasets.txt', 'r') as f:
    lines = f.readlines()

download_dataset(lines, \
                 user='voxceleb1902', \
                 password='nx0bl2v2', \
                 save_path='../data')

# Download VoxCeleb2 test set
with open('../data/lists/datasets2.txt', 'r') as f:
    lines = f.readlines()

download_dataset(lines, \
                 user='voxceleb1902', \
```

```
                    password='nx0bl2v2', \
                    save_path='../data')

# Extract VoxCeleb1 test set and VoxCeleb2 test set
extract_dataset(save_path='../data/voxceleb1_test', \
                    fname='../data/vox1_test_wav.zip')

extract_dataset(save_path='../data/voxceleb2_test', \
                    fname='../data/vox2_test_aac.zip')

# Download VoxCeleb1-O cleaned protocol
with open('../data/lists/protocols.txt', 'r') as f:
    lines = f.readlines()

download_protocol(lines, save_path='../data/voxceleb1_test')
```

Далее необходимо выполнить преобразование базы VoxCeleb2 test set из m4a-формата в wav-формат с помощью функции `aac_to_wav` из файла **`../common/dataprep.py`**. Для запуска процедуры преобразования в параллельном режиме можно воспользоваться функцией `run_voxceleb_convert` из файла **`../common/dataprep.py`**.

```
def aac_to_wav(infile, outfile):
    # Function to convert from aac to wav

    common.aac_to_wav(infile, outfile)

# Prepare VoxCeleb2: convert from aac to wav format
run_voxceleb_convert(input_path='../data/voxceleb2_test/aac', \
                     result_path='../data/voxceleb2_test/wav', \
                     fun=aac_to_wav)
```

Затем необходимо подготовить тестовые на основе VoxCeleb1 test set и адаптационные на основе VoxCeleb2 test set данные путём применения к оригинальным звукозаписям MP3-кодека. Таким образом, будут получены данные в новом домене для проведения исследований по адаптации. Для обработки звукозаписей с помощью MP3-кодека можно воспользоваться функцией `apply_mp3_codec` из файла **`../common/dataprep.py`**.

```
def convert_to_in_domain(infile, outfile):
    # Function to apply MP3 codec

    common.apply_mp3_codec(infile, outfile)

# Create in-domain test dataset based on VoxCeleb1 test set
run_voxceleb_convert(input_path='../data/voxceleb1_test/wav', \
result_path='../data/voxceleb1_test/wav_mp3_codec', \
fun=convert_to_in_domain, \
threads=10)
```

```
# Create adaptation dataset based on VoxCeleb2 test set
run_voxceleb_convert(input_path='../data/voxceleb2_test/wav', \
result_path='../data/voxceleb2_test/wav_mp3_codec', \
fun=convert_to_in_domain, \
threads=10)
```

2. Вычисление дикторских моделей для эталонных и тестовых звукозаписей используемого протокола

На данном этапе выполнения лабораторной работы существуют две базы данных, которые можно использовать, чтобы протестировать блок построения дикторских моделей из лабораторной работы № 3:

1. Исходная, выходящая за границы целевых условий (out-of-domain), тестовая база VoxCeleb1 test set. Отметим, что в лабораторной работе № 3 экстрактор дикторских моделей был обучен для условий, в которых была записана оригинальная база VoxCeleb1 test set;
2. Целевая (in-domain) тестовая база VoxCeleb1 test set, обработанная MP3-кодеком. В настоящей лабораторной работе при решении задачи адаптации требуется, чтобы обученный ранее экстрактор дикторских моделей качественно работал для новых целевых условий.

Отметим, что с тестовыми базами, определёнными выше, в настоящей лабораторной работе связан одинаковый протокол тестирования VoxCeleb1-O cleaned. Аналогично лабораторной работе № 4, построим дикторские модели для имеющихся тестовых баз с использованием нейросетевого экстрактора дикторских эмбеддингов, обученного на out-of-domain тренировочном множестве (без применения MP3-кодека), и оценим качество полученных дикторских моделей на имеющихся in-domain и out-of-domain тестовых данных. Последовательность действий в рамках настоящего пункта должна быть следующей:

1. Описать и инициализировать модель для генерации дикторских эмбеддингов, и загрузить в неё обученные веса (должны быть получены в рамках лабораторной работы № 3);
2. Вычислить эмбеддинги для эталонных и тестовых звукозаписей исходной тестовой базы (out-of-domain условия) и её версии, трансформированной MP3-кодеком (in-domain условия).

Ниже представлен программный код, инициализирующий модель и загружающий в неё обученные веса.

```
# Model initialization
model = MainModel()

# Load trained model
with open('../data/lists/models.txt', 'r') as f:
    lines = f.readlines()
```

```
model = load_model(model, lines, save_path='../data/models')
model.eval()
model = model.cuda()
```

Ниже представлен программный код, позволяющий загрузить верификационный протокол тестирования, инициализировать загрузчики для двух наборов тестовых данных, а также выполнить вычисление дикторских моделей для этих данных.

```
# Read test protocol
with open('../data/voxceleb1_test/veri_test2.txt', 'r') as f:
    lines = f.readlines()

# Get a list of unique file names
files = \
list(itertools.chain(*[x.strip().split()[-2:] for x in lines]))
setfiles = list(set(files))
setfiles.sort()

# Define out-of-domain test dataloader
test_dataset = test_dataset_loader(setfiles, \
               test_path='../data/voxceleb1_test/wav', \
               eval_frames=500, \
               num_eval=1)
test_loader = torch.utils.data.DataLoader(test_dataset, \
                                          batch_size=1, \
                                          shuffle=False, \
                                          num_workers=8, \
                                          drop_last=False, \
                                          sampler=None)

# Extract features for every out-of-domain waveform
feats_out_of_domain = extract_features(model, test_loader)

# Define in-domain test data loader
test_dataset = test_dataset_loader(setfiles, \
test_path='../data/voxceleb1_test/wav_mp3_codec', \
eval_frames=500, \
num_eval=1)

test_loader = torch.utils.data.DataLoader(test_dataset, \
                                          batch_size=1, \
                                          shuffle=False, \
                                          num_workers=8, \
                                          drop_last=False, \
                                          sampler=None)

# Extract features for every in-domain waveform
feats_in_domain = extract_features(model, test_loader)
```

3. Формирование оценок сравнения эталонных и тестовых дикторских моделей

Для того чтобы автоматически принять решение о принадлежности пары сравнения «эталон-тест» к таргет- или импостор-попытке, необходимо сформировать оценку сравнения (score) [2, 5] между эталонной и тестовой дикторскими моделями. В простейшем случае подобные оценки можно получить с использованием *евклидовой метрики*, *косинусной метрики* и т.п.

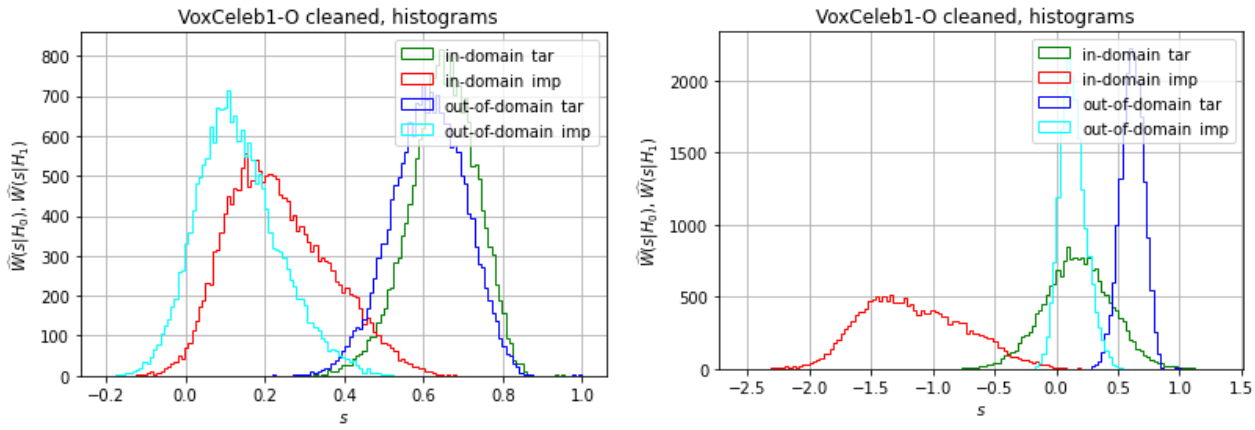
В рамках настоящего пункта необходимо выполнить следующее:

1. Осуществить формирование оценок сравнения эталонных и тестовых дикторских моделей с использованием любой подходящей метрики для out-of-domain и in-domain тестовых баз;
2. Сформировать списки оценок сравнения, таргет- / импостор-меток и имен «эталон-тест» в триальных сравнениях;
3. Получить оценки равного уровня ошибок (EER, equal error rate) [3, 5] и построить графики распределения для out-of-domain и in-domain тестовых баз.

Ниже представлен вызов функции `compute_scores_cosine` (вычисление оценок сравнения), определённой в файле **`../common/scoring.py`** по отношению к out-of-domain и in-domain тестовым данным.

```
# Compute scores between enroll and test speaker models
# for in-domain and out-of-domain data
all_scores_in_domain, \
all_labels_in_domain, \
all_trials_in_domain = \
compute_scores_cosine(feats_in_domain, protocol)

all_scores_out_of_domain, \
all_labels_out_of_domain, \
all_trials_out_of_domain = \
compute_scores_cosine(feats_out_of_domain, protocol)

# Compute target and impostor histogram
plot_histograms_2sets(all_scores_in_domain, \
                      all_labels_in_domain, \
                      all_scores_out_of_domain, \
                      all_labels_out_of_domain)
```

На рисунке 1а показан результат выполнения функции `plot_histograms_2sets` из **`./exercises_blank.py`**. На рисунке 1а представлены таргет- и импостор-гистограммы распределения оценок сравнения «эталон-тест» для случая out-of-domain и in-domain данных из VoxCeleb1 test set, связанных с протоколом тестирования VoxCeleb1-O cleaned, для анализируемого экстрактора дикторских моделей.



а) б)

Рисунок 1 – Таргет- и импостор-гистограммы распределения оценок сравнения «эталон-тест» для случая out-of-domain и in-domain данных из VoxCeleb1 test set, связанных с протоколом тестирования VoxCeleb1-O cleaned, представлены на рисунке 1а (EER = 1,308 % для out-of-domain условий и EER = 3,776 % для in-domain условий). На рисунке 1б представлено откорректированное расположение таргет- и импостор-гистограмм распределения из рисунка 1а для случая in-domain данных (EER = 1,308 % для out-of-domain условий и EER = 3,149 % для in-domain условий)

4. Доменная адаптация на основе s-нормализации

Доменная адаптация (domain adaptation) [2, 3] рассматривается как проблема настройки параметров модели для случая, когда существует большой объём нецелевых данных (out-of-domain data или source data) и малый объём размеченных или неразмеченных целевых данных (in-domain data или target data). В научно-технической литературе по распознаванию диктора часто выделяют следующие адаптационные техники: *вычитание среднего*, *s-нормализация*, *корреляционное выравнивание* и т.п. Важно отметить, что доменная адаптация может выполняться на всех этапах конвейера голосовой биометрии, то есть на этапе предобработки, диаризации, построения дикторской модели и т.п.

В рамках настоящей лабораторной работы рассматривается наиболее часто используемый в научно-технической литературе подход доменной адаптации, основанный на процедуре s-нормализации.

Адаптация с помощью *s-нормализации* [6] предполагает выполнение процедуры нормализации оценок сравнения эталонов и тестов с целью уменьшения вариативности в попытках сравнения, которое ведёт к более лучшей калибровке и более надёжному выбору порога принятия решения для системы голосовой биометрии. При выполнении адаптации на основе s-нормализации необходимо выполнить следующую последовательность действий:

1. Выполнить сравнение эталонной и тестовой дикторских моделей в рамках одной попытки, получив оценку;

2. Ввести в рассмотрение *импосторную когорту*, состоящую из дикторских моделей, которые не пересекаются, в смысле дикторских меток, с эталонной и тестовой моделями в рамках одной попытки;
3. Сравнить эталонную дикторскую модель с каждой моделью из импосторной когорты, получив оценки сравнения; упорядочить оценки сравнения по убыванию схожести моделей; выбрать первые N оценок (самые сложные оценки сравнения) и вычислить для них среднее, μ_e, и среднеквадратическое отклонение, σ_e;
4. Повторить процедуру выше для тестовой дикторской модели, получив значения μ_t и σ_t;
5. Выполнить нормализацию оценки сравнения s для рассматриваемой попытки: $\hat{s} = (s - \mu_e)/(2\sigma_e) + (s - \mu_t)/(2\sigma)_t$.

В рамках настоящего пункта необходимо реализовать этапы адаптации на основе s-нормализации, описанные выше. Предполагается, что данные из базы VoxCeleb2 test set будут использоваться в качестве адаптационного множества. Поскольку основу адаптации с помощью s-нормализации составляет импосторная когорта, то искажённые под целевые условия данные из базы VoxCeleb2 test set в рамках настоящего пункта будут использоваться для построения этой когорты. При проведении процедуры тестирования в одном случае база VoxCeleb1 test set будет рассматривается в оригинальном виде, а в другом – в искажённом с использованием MP3-кодека (предполагается, что выбранное искажение моделирует целевые условия). Итогом выполнения настоящего пункта является оценка качества полученного после адаптации к целевым условиям верификационного решения. Качество предлагается проанализировать с использованием равного уровня ошибок и гистограмм распределения таргет- и импостор-оценок.

Для выполнения процедуры адаптации на основе s-нормализации предлагается воспользоваться кодом ниже. Для выполнения данного кода необходимо завершить функцию `s_norm` из **`./exercises_blank.py`**.

```
setfiles = \
get_voxceleb_filelist(input_path=os.path.join('../data', \
voxceleb2_test', 'wav_mp3_codec'))
setfiles.sort()

# Define adaptation data loader
adapt_dataset = test_dataset_loader(setfiles, \
test_path='../data/voxceleb2_test/wav_mp3_codec', \
eval_frames=1000, \
num_eval=1)
adapt_loader  = torch.utils.data.DataLoader(adapt_dataset, \
                                            batch_size=1, \
                                            shuffle=False, \
                                            num_workers=8, \
                                            drop_last=False, \
```

```
                                                    sampler=None)

# Extract features for every waveform
feats_adapt = extract_features(model, adapt_loader)

# Perform s normalization
all_scores_in_domain, \
all_labels_in_domain, \
all_trials_in_domain = \
s_norm(feats_in_domain, protocol, feats_adapt, N_s=100, eps=0.6)

# Compute target and impostor histogram
plot_histograms_2sets(all_scores_in_domain, \
                      all_labels_in_domain, \
                      all_scores_out_of_domain, \
                      all_labels_out_of_domain)
```

На рисунке 1б представлены скорректированные таргет- и импостор-гистограммы распределения оценок сравнения «эталон-тест» для случая in-domain данных из VoxCeleb1 test set, связанных с протоколом тестирования VoxCeleb1-O cleaned, для анализируемого экстрактора дикторских моделей.

5. Калибровка системы голосовой биометрии

Результатом процедуры сравнения дикторских моделей являются «сырые» оценки сравнения. Основными проблемами при использовании «сырых» оценок являются:

1. Порог принятия решения для выбранной рабочей точки полностью привязан к системе распознавания диктора (если существует N систем, то существует N порогов);
2. Для произвольной выходной оценки сравнения отсутствует информация о вероятности принадлежности к статистическим гипотезам;
3. Сложно учесть влияние параметров качества речевого сигнала на выходные оценки.

Калибровка [4, 5, 7] является решением указанных проблем. Необходимость калибровки в сообществе по распознаванию диктора обусловлена следующим:

1. Представление оценок сравнения в калиброванном виде для конкурсов, например, NIST SRE. Качество калибровки учитывается наравне с качеством верификации;
2. В коммерческих системах распознавания диктора, так как на практике для заданной оценки сравнения важно знать вероятность статистической гипотезы.

В простейшем случае *модель калибровки* для системы распознавания диктора может быть описана в виде линейной функциональной зависимости:

$s_c = a \cdot s + b$. Здесь a и b – это параметры *линейной модели калибровки* [4], s – это «сырое» значение оценки сравнения, а s_c – значение оценки сравнения после выполнения процедуры калибровки.

Для обучения параметров a и b линейной модели калибровки можно воспользоваться стоимостной функцией для *логистической регрессии* [4]. Можно доказать, что при решении задачи верификации апостериорная вероятность принадлежности пары сравнения «эталон-тест» к таргет-паре (нулевая гипотеза, H_0), для заданного значения калиброванной оценки сравнения эталона и теста, может быть выражена следующим образом: $P(H_0 \,|s_c) = 1/\left(1 + e^{-s_c + T_{act}^{\mathrm{LLR}}}\right)$. Здесь T_{act}^{LLR} – это величина оптимального порога принятия решения в пространстве логарифма отношения правдоподобия. Фактически величина T_{act}^{LLR} задаёт желаемую теоретически *рабочую точку биометрической системы*. Обычно в научно-технической литературе по распознаванию диктора величина T_{act}^{LLR} задаётся в следующем виде: $T_{act}^{\mathrm{LLR}} = \ln((1 - P(H_0))/P(H_0))$. Здесь $P(H_0)$ – это априорная вероятность таргет-гипотезы. Аналогичным образом, что и выше, можно записать апостериорную вероятность принадлежности пары сравнения «эталон-тест» к импостор-паре (альтернативная гипотеза, H_1) для заданного значения калиброванной оценки сравнения эталона и теста: $P(H_1 \,|s_c) = 1/(1 + e^{s_c - T_{act}^{\mathrm{LLR}}})$. Предполагая, что априорные вероятность выпадения таргет- и импостор-попыток равны $P(H_0)$ и $P(H_1) = 1 - P(H_0)$, соответственно, можно записать следующее выражение для стоимостной функции, которая позволит путём численной оптимизации оценить оптимальные значения параметров a и b через минимизацию этой функции:

$$J(a,b) = \frac{P(H_0)}{N_0}\sum_{i=1}^{N_0} \ln\left(1 + e^{-(as_i + b) + T_{act}^{\mathrm{LLR}}}\right) + \frac{P(H_1)}{N_1}\sum_{i=1}^{N_1} \ln(1 + e^{(as_i + b) - T_{act}^{\mathrm{LLR}}}).$$

Здесь N_0 и N_1 – это количество таргет- и импостор-пар в *калибровочном протоколе* для верификации, который необходимо подготовить отдельно перед обучением калибровочной модели. Попытки сравнения «эталон-тест» в калибровочном протоколе должны быть схожими, в смысле длительностей эталонной и тестовой звукозаписей, условий записи фонограмм и т.п., с теми попытками, которые будут встречаться при практическом использовании итоговой системы распознавания диктора. Для оценки качества модели калибровки системы голосовой биометрии можно воспользоваться подходом, представленным ниже.

Пусть в настоящей лабораторной работе *байесовский риск* [4, 5] задан в следующем *нормализованном* виде: $\bar{C} = P(D_1|H_0) + P(D_0|H_1)P(H_1)/P(H_0)$. Здесь $P(D_1|H_0)$ – это вероятность ошибки первого рода, а $P(D_0|H_1)$ – вероятность ошибки второго рода. Биометрическая система считается откалиброванной, если практический минимум байесовского риска, $\bar{C}_{min}$, вычисленный в эмпирически подобранном пороге T_{min}^{LLR} по отношению

к некоторому протоколу тестирования для задачи верификации, совпадает с теоретическим минимум байесовского риска, $\bar{C}_{act}$, для точки T_{act}^{LLR} [4, 5].

В рамках настоящего пункта необходимо обучить линейную модель калибровки. Для решения указанной задачи требуется выполнить следующие действия:

1. Подготовить калибровочный протокол для обучения модели калибровки;
2. Задать рабочую точку, которая определит величину T_{act}^{LLR};
3. Обучить калибровочную модель путём минимизации стоимостной функции для логистической регрессии с использованием методов численной оптимизации первого порядка;
4. Посчитать значения $\bar{C}_{min}$ и $\bar{C}_{act}$, а также T_{min}^{LLR} и T_{act}^{LLR} для выбранного протокола тестирования до выполнения и после выполнения процедуры калибровки.

Ниже представлен программный код, позволяющий выполнить обучение модели калибровки.

Вначале необходимо выполнить настройку гиперпараметров процедуры обучения модели калибровки.

```
# Select hyperparameters

# Calibration parameters
P_tar = 0.5 # prob. of target hypothesis
calib_protocol_size = 40000 # number of trials in calibration
                            # protocol

# Train dataloader
batch_size = 1     # batch size to train
pin_memory = True  # pin memory
num_workers = 1    # number of workers to train
shuffle = True     # shuffling of training examples

# Optimizer
weight_decay = 0   # weight decay value
lr = 5             # learning rate value

# Scheduler
step_size = 100    # period of learning rate decay
gamma = 0.1        # multiplicative factor of learning rate decay
max_epoch = 300    # number of epochs
```

Далее необходимо загрузить метаданные для базы VoxCeleb2 test set и инициализировать для неё загрузчик оригинальных данных, для которых требуется построить дикторские модели. Предполагается, что указанная база в рамках настоящего пункта будет использоваться в качестве калибровочного набора данных. С использованием метаданных по базе VoxCeleb2 test set может быть сгенерирован калибровочный протокол для верификации.

```
# Download metadata for calibration set
with open('../data/lists/protocols2.txt', 'r') as f:
    lines = f.readlines()

download_protocol(lines, save_path='../data/voxceleb2_test')

setfiles = \
get_voxceleb_filelist(input_path='../data/voxceleb2_test/wav')
setfiles.sort()

# Define calibration data loader
calib_dataset = test_dataset_loader(setfiles, \
test_path='../data/voxceleb2_test/wav', \
eval_frames=1000, \
num_eval=1)
calib_loader  = torch.utils.data.DataLoader(calib_dataset, \
                                            batch_size=1, \
                                            shuffle=False, \
                                            num_workers=8, \
                                            drop_last=False, \
                                            sampler=None)

# Extract features for every waveform
feats_calib = extract_features(model, calib_loader)

# Create calibration protocol
spk_ids_calib_protocol = [] # speaker IDs in calibration
protocol
for key in feats_calib.keys():
    spk_ids_calib_protocol.append(key.split('/')[0])

spk_ids_calib_protocol = list(set(spk_ids_calib_protocol))

protocol_train = [] # calibration protocol

# Generate target trials
count = 0
while count < int(P_tar*calib_protocol_size):
    spk_id   = np.random.choice(spk_ids_calib_protocol, 1)[0]
    spk_utts = \
    list(filter(lambda x: spk_id in x, feats_adapt.keys()))
    spk_utts = np.random.choice(spk_utts, 2)

    if spk_utts[0].split('/')[1] != spk_utts[1].split('/')[1]:
        protocol_train.append(' '.join(['1', spk_utts[0], \
                                spk_utts[1]]))

        count = count + 1

# Generate impostor trials
count = 0
while count < int((1 - P_tar)*calib_protocol_size):
```

```
    spk_ids  = np.random.choice(spk_ids_calib_protocol, 2)

    if spk_ids[0] != spk_ids[1]:
        spk1_utts = list(filter(lambda x: spk_ids[0] in x, \
                    feats_adapt.keys()))
        spk1_utts = np.random.choice(spk1_utts, 1)[0]
        spk2_utts = list(filter(lambda x: spk_ids[1] in x, \
                    feats_adapt.keys()))
        spk2_utts = np.random.choice(spk2_utts, 1)[0]

        protocol_train.append(' '.join(['0', spk1_utts, \
                              spk2_utts]))

        count = count + 1
```

Для пар сравнений «эталон-тест» из калибровочного протокола могут быть вычислены оценки сравнения эталонов и тестов. Вычисленные оценки с учётом их принадлежности к таргет- и импостор-сравнениям будут использоваться для обучения параметров линейной модели калибровки методом логистической регрессии.

```
# Compute scores between enroll and test for calibration
# protocol
all_scores_calib_train, \
all_labels_calib_train, \
all_trials_calib_train = \
compute_scores_cosine(feats_calib, protocol_train)
```

Для загрузки вычисленных значений оценок сравнения эталонов и тестов в ходе выполнения процедуры обучения необходимо инициализировать загрузчик данных.

```
# Initialize train dataloader
tar_calib_train, imp_calib_train = \
get_tar_imp_scores(all_scores_calib_train, \
                   all_labels_calib_train)

calib_train_set = \
CalibrationDataset(target_scores=tar_calib_train, \
                   impostor_scores=imp_calib_train)

calib_train_loader = DataLoader(calib_train_set, \
                                batch_size=1, \
                                pin_memory=pin_memory, \
                                num_workers=num_workers, \
                                shuffle=True)
```

Для запуска процедуры обучения параметров калибровочной модели необходимо выполнить инициализации модели, оптимизатора и планировщика процедуры обучения, а также определить стоимостную

функцию. Запуск процедуры обучения может быть выполнен с использованием программного кода ниже после завершения определения класса `LinearCalibrationModel` (класс линейной калибровочной модели), класса `CalibrationLoss` (класс стоимостной функции) и функции `train_calibration` (процедура обучения калибровочной модели) из **`./exercises_blank.py`**.

```
# Initialize calibration model
calib_model = LinearCalibrationModel()
criterion = CalibrationLoss(ptar=P_tar)

# Initialize optimizer and scheduler
optimizer = SGDOptimizer(calib_model.parameters(), \
                         lr=lr, \
                         weight_decay=weight_decay)

scheduler = StepLRScheduler(optimizer, \
                            test_interval=step_size, \
                            lr_decay=gamma)

# Train model
train_calibration(calib_train_loader, \
                  calib_model, \
                  criterion, \
                  optimizer, \
                  scheduler, \
                  max_epoch, \
                  verbose=True)
```

Итогом исследований настоящего пункта является применение обученной калибровочной модели к «сырым» оценкам сравнения, полученным применительно к протоколу VoxCeleb1-O cleaned и связанной с ним оригинальной базой VoxCeleb1 test set. С помощью программного кода ниже, помимо вычисления калиброванных оценок сравнения эталонов и тестов, можно построить гистограммы таргет- и импостор-оценок после выполнения процедуры калибровки (рисунок 2), а также вычислить значения EER, $\bar{C}_{min}$ и $\bar{C}_{act}$ до и после калибровки.

```
# Create target and impostor raw scores for original
# VoxCeleb1 test set
tar_test, imp_test = \
get_tar_imp_scores(all_scores_out_of_domain, \
                   all_labels_out_of_domain)

# Compute target and impostor calibrated scores
calib_model.eval()
tar_test_calib = \
calib_model(torch.tensor([tar_test]).transpose(0, 1))
tar_test_calib = tar_test_calib.detach().numpy().squeeze()
```

```
imp_test_calib = \
calib_model(torch.tensor([imp_test]).transpose(0, 1))
imp_test_calib = imp_test_calib.detach().numpy().squeeze()

# Plot histograms for target and impostor scores to calibrated
# scores
min_scores = np.concatenate((tar_test_calib, \
                             imp_test_calib)).min()
max_scores = np.concatenate((tar_test_calib, \
                             imp_test_calib)).max()

hist(tar_test_calib, \
     int(sqrt(len(tar_test_calib))), \
     histtype='step', \
     color='blue', \
     range=(min_scores, max_scores))
hist(imp_test_calib, \
     int(sqrt(len(imp_test_calib))), \
     histtype='step', \
     color='cyan', \
     range=(min_scores, max_scores))
xlabel('$s_c$')
ylabel('$\widehat{W}(s_c|H_0)$, $\widehat{W}(s_c|H_1)$')
title('VoxCeleb1-O cleaned, histograms'); grid(); show()
```

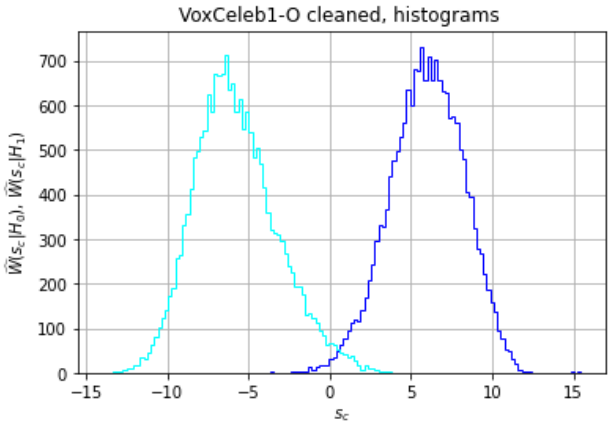


Рисунок 2 – Пример гистограмм распределения таргет- (справа) и импостор-оценок (слева) сравнения после выполнения процедуры калибровки с помощью обученной линейной модели. Значения целевых метрик оценки качества в данном случае получились следующими: EER = 1,308 %, $\bar{C}_{min} = 0{,}0258$ и $\bar{C}_{act} = 0{,}0292$

```
# Compute metrics before calibration
EER, _ = get_eer(tar_test, imp_test)
minDCF, _, actDCF, _ = get_dcf(tar_test, \
                               imp_test, \
                               P_target=P_tar)

print('Values of metrics before calibration')
print("EER: {:.3f}%".format(EER))
print("minDCF: {0:.4f}, actDCF: {1:.4f}".format(minDCF, actDCF))
```

```
# Compute metrics after calibration
EER, _ = get_eer(tar_test_calib, imp_test_calib)
minDCF, _, actDCF, _ = get_dcf(tar_test_calib, \
                               imp_test_calib, \
                               P_target=P_tar)

print('Values of metrics after calibration')
print("EER: {:.3f}%".format(EER))
print("minDCF: {0:.4f}, actDCF: {1:.4f}".format(minDCF, actDCF))
```

Порядок защиты и представление результатов выполнения лабораторной работы

Для защиты лабораторной работы студент должен предоставить преподавателю самостоятельно завершённые программные коды из файла **./exercises_blank.py** и продемонстрировать их работоспособность путём поэтапного запуска файла **./lab5_blank.ipynb**. Преподаватель оставляет за собой право задать необходимые вопросы, затрагивающие программную реализацию заданий лабораторной работы. В процессе запуска файла **./lab5_blank.ipynb** студенту необходимо:

1. Визуализировать таргет- и импостор-гистограммы распределения оценок сравнения эталонных и тестовых звукозаписей для случая out-of-domain и in-domain данных, привязанных к заданному протоколу тестирования, для выбранной модели блока генерации дикторских эмбеддингов. Привести для каждого из случаев значение метрики равного уровня ошибок для оценки качества биометрического решения;
2. Визуализировать таргет- и импостор-гистограммы распределения оценок сравнения эталонных и тестовых звукозаписей для случая in-domain данных, привязанных к заданному протоколу тестирования, после применения процедуры адаптации на основе s-нормализации для выбранной модели блока генерации дикторских эмбеддингов. Привести значение метрики равного уровня ошибок для оценки качества биометрического решения в рассматриваемом случае. Сравнить полученное значение с величиной равного уровня ошибок для случая in-domain данных до применения процедуры s-нормализации;
3. Визуализировать таргет- и импостор-гистограммы распределения оценок сравнения эталонных и тестовых звукозаписей, привязанных к заданному протоколу тестирования, после применения линейной модели калибровки для выбранной модели блока генерации дикторских эмбеддингов. Привести значения целевых метрик оценки качества до и после выполнения процедуры калибровки для рассматриваемого биометрического решения. Проанализировать полученный результат.

Контрольные вопросы

1. Что такое доменная адаптация?
2. Как выполнить процедуру адаптации на основе s-нормализации?
3. Для какой цели решается задача калибровки?
4. Что такое «хорошая» калибровка?
5. Какие модели калибровки систем голосовой биометрии существуют?
6. Как обучить калибровочную модель с использованием метода логистической регрессии?

Список литературы

1. Nagrani A., Chung J.S., Xie W., Zisserman A. Voxceleb: large-scale speaker verification in the wild // Computer Speech & Language. 2020. V. 60. P. 101027.
2. Bai Z., Zhang X.-L., Chen J. Speaker recognition based on deep learning: an overview // arXiv preprint arXiv:2012.00931 [eess.AS]. 2021.
3. Beigi H. Fundamentals of speaker recognition. Springer, 2011.
4. Brümmer N., Villiers E. The BOSARIS toolkit: theory, algorithms and code for surviving the new DCF // arXiv:1304.2865 [stat.AP].
5. Hansen J.H.L., Hasan T. Speaker recognition by machines and humans: a tutorial review // IEEE Signal Processing Magazine. 2015. V. 32, № 6. P. 74–99.
6. Matějka P., Novotný O., Plchot O., Burget L., Sánchez M.D., Černocký J. Analysis of score normalization in multilingual speaker recognition // Interspeech 2017. 2017. P. 1567–1571.
7. Mandasari M.I., Saeidi R., McLaren M., Leeuwen D. Quality measure functions for calibration of speaker recognition systems in various duration conditions // IEEE Trans. Audio Speech and Language Processing. 2013. V. 21, № 11. P. 2425–2438.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Н

О

П

Р

С

У

Ф

Ш

Э

СОДЕРЖАНИЕ

Волохов Владимир Андреевич
Лаврентьева Галина Михайловна
Новосёлов Сергей Александрович
Матвеев Юрий Николаевич

Методические указания к выполнению лабораторных работ по курсу «Распознавание диктора»

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, лит. А