Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.934.8'1 *На правах рукопису*

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема "Інформаційна технологія голосової ідентифікації за біометричними характеристиками людини"
Спеціальність 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
ІПЗм-21 _____ /Анастасія ШЕВЧЕНКО/

Науковий керівник
к.т.н., доц. _____ /Тетяна КОВАЛЮК/

Консультант
з питань нормоконтролю
к.т.н., асистент _____/Анастасія ВЕЧЕРКОВСЬКА/

Допускається до захисту
Завідувач кафедри
д.т.н., проф. _____ /Олексій БИЧКОВ/

Київ – 2022

Рішенням Екзаменаційної комісії
випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії
д.т.н., проф. Андрій БОНДАРЧУК

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій Спеціальність
121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____(О.С.Бичков)
“____”_____20__р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
СТУДЕНТУ
Шевченко Анастасії Дмитрівні
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної магістерської роботи «Інформаційна технологія голосової ідентифікації за біометричними характеристиками людини»

затверджені наказом вищого навчального закладу від „21” грудня 2021 р. №8.

2. Строк здачі студентом закінченої роботи 9 травня 2022 р.

3. Вихідні дані до роботи
Вхідний набір даних зі 43832 файлів від 108 дикторів, метадані про кожного з дикторів (стать, акцент);
мова програмування Python, бібліотеки PyQt5, keras, librosa, parselmouth, scikit-learn;
акустичні характеристики голосів дикторів;
багатошарова повнозв'язна штучна нейронна мережа прямого поширення;
формальні моделі ідентифікації особи.

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
Проблема ідентифікації особи за голосом; проектування та розробка інформаційної технології голосової ідентифікації; тестування та порівняння моделей багатошарових нейронних мереж; тестування моделей гаусових сумішей; реалізація системи ідентифікації особи як програмного застосунку.

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)
Архітектура системи розпізнавання, граф-схема етапів попередньої обробки даних, архітектура штучної нейронної мережі, UML-діаграма класів, відеограми роботи
UML-діаграма класів, відеограми роботи системи, діаграма прецедентів, діаграма шарів програмного застосунку.

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Ковалюк Тетяна Володимирівна	24.12.2021	24.12.2021
2	Ковалюк Тетяна Володимирівна	19.01.2022	19.01.2022
3	Ковалюк Тетяна Володимирівна	27.02.2022	27.02.2022
4	Ковалюк Тетяна Володимирівна	12.03.2022	12.03.2022

7. Дата видачі завдання 24 грудня 2021

Керівник Тетяна КОВАЛЮК
(підпис) (розшифровка підпису)

Завдання прийняв до виконання Анастасія ШЕВЧЕНКО
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів роботи	Термін виконання етапів роботи	Примітка
1. Ознайомлення літературою та інтернет-джерелами на тему ідентифікації диктора	01.02.2022 —10.02.2022	
2. Формалізація постановки задачі, вимог, аналіз об’єкту дослідження, вибір методів для реалізації поставленої задачі.	11.02.2022 — 13.02.2022	
3. Формування та обробка набору даних.	14.02.2022 — 18.02.2022	
4. Проектування архітектури системи.	19.02.2022 — 26.02.2022	
5. Розробка блоків попередньої обробки та витягнення ознак, їх тестування. Витягнення та збереження ознак із вхідного набору даних	26.02.2022 — 15.03.2022	
6. Розробка блоку класифікації, навчання класифікаторів на оброблених вхідних даних. Розробка алгоритму агрегації виходів класифікаторі	15.03.2022 — 27.03.2022	
7. Тестування блоку класифікації	28.03.2022 — 30.03.2022	
8. Розробка програмного застосунку з інтерфейсом користувача, його тестування	31.03.2022 — 12.04.2022	
9. Оформлення пояснювальної записки	28.03.2022 — 17.05.2022	
10. Підготовка презентаційних матеріалів	18.05.2022 — 20.05.2022	

Студент – магістр Анастасія ШЕВЧЕНКО
(підпис) (розшифровка підпису)

Керівник роботи Тетяна КОВАЛЮК
(підпис) (розшифровка підпису)

Анотація

Було виконано магістерську випускну кваліфікаційну роботу на тему «Інформаційна технологія голосової ідентифікації за біометричними характеристиками людини» за спеціальністю 121 — «Інженерія програмного забезпечення». Робота містить 22 рисунки, 2 таблиці, 3 додатки та 32 джерела, складається із 81 сторінку.

Метою роботи є дослідження методів обробки аудіо, проведення експериментальних дослідів по ідентифікації особи за записами її голосу. Об'єктом дослідження є голосова ідентифікація особи. Предметом дослідження є методи обробки звуку, моделі нейронних мереж та гаусових сумішей як класифікатори, що ідентифікують особу за вектором ознак. Результатом є система голосової ідентифікації у вигляді програмного застосунку, яка здатна розпізнавати особу з точністю до 94,8%, запропоновані рішення можуть використовуватися в подальших дослідженнях та розробках у даній сфері.

Ключові слова: машинне навчання, штучні нейронні мережі, моделі гаусових сумішей, ідентифікація диктора, біометрія, розпізнавання голосу.

Summary

A master's degree project: "A biometrics-based voice recognition information technology" was completed**,** speciality 121 — "Software Engineering". The thesis contains 22 figures, 2 tables, 3 appendices and 32 references, consists of 81 pages.

The goal is to research audio processing methods, conduct speaker voice identification experiments. An object of study is person identification by voice. A subject of study is audio processing methods, neural networks and gaussian mixture models as person classifiers. The result is a voice identification system as a software application that's capable to recognize a person with an accuracy of maximum 94,8%, the solutions proposed can be used in further studies and developments in this field.

Keywords: machine learning, artificial neural networks, Gaussian mixture models, speaker identification, biometrics, voice recognition.

ЗМІСТ

ВСТУП

Біометричні системи розпізнавання все частіше застосовуються як більш «природні» засоби розпізнавання осіб. Найпростішим для отримання, найчастіше вживаним і поширеним у суспільстві біометричним показником є людська мова. Таким чином, системи розпізнавання дикторів — це технології, які використовують людську мову для розпізнавання, ідентифікації або перевірки особи.

Оскільки розмовна мова є одним із найдоступніших засобів (необхідний лише мікрофон), використовується в різноманітних програмах транзакцій (наприклад, телефонний банкінг) і має потенціал для безпеки шляхом спостереження (з використанням технології підслуховування), не дивно, що розпізнавання мовця є одним із ключових досліджень області обробки сигналів і розпізнавання образів.

Темою даної випускної кваліфікаційної роботи є голосова ідентифікація за біометричними характеристиками людини.

Актуальність роботи підкріплюється потребою в надійних й практичних рішеннях задачі ідентифікації особи за голосом, підвищеним інтересом наукової спільноти до вирішення прикладних задач методами машинного навчання.

Метою випускної кваліфікаційної роботи є дослідження методів обробки аудіофайлів, методів класифікації нейронних мереж, моделей гаусових сумішей, проведення експериментальних дослідів по ідентифікації особи за записами її голосу.

Об’єктом дослідження є голосова ідентифікація особи.

Предметом дослідження є методи обробки звуку з метою витягнення характеристик голосу особи, моделі нейронних мереж та гаусових сумішей як класифікатори, що ідентифікують особу за вектором ознак.

Новизна роботи полягає в застосуванні гібридного блоку класифікації, що складається з двох класифікаторів та агрегатора їх виходів для покращення точності розпізнавання.

РОЗДІЛ 1 ПРОБЛЕМА ІДЕНТИФІКАЦІЇ ОСОБИ ЗА ГОЛОСОМ

1.1. Голосова біометрія та сфери її використання

Біометрією називають сукупність технологій підтвердження та ідентифікації особи, які засновані на перетворенні біологічних, морфологічних та поведінкових характеристик у цифрове представлення [1]. Метою біометрії є розпізнавання особи шляхом оцінки унікальних рис деякої незмінної частини тіла. Перевагами біометричної аутентифікації є зручність використання, ускладнені втрати або забування (адже біометричні характеристики є складовою користувача), ускладнене фальсифікування.

Розрізняють два класи методів біометричної аутентифікації: статичні (за відбитком пальця, райдужною оболонкою очей, геометрією руки) та динамічні (за голосом, за почерком). Статичні засновані на біометричних характеристиках, які присутні з народження, динамічні — на поведінкових [2].

Голосова біометрія належить до динамічних методів. До її переваг належить простота використання та реалізації, загальнодоступність та наявність великої кількості методів побудови шаблонів. Основним недоліком є мінливість голосу через вплив таких факторів, як вік, настрій, здоров’я.

За допомогою інформації, що міститься у мовленнєвому сигналі, голосова біометрія вирішує одну з найбільш актуальних проблем сучасних мовленнєвих технологій — завдання розпізнавання диктора.

Вирішення цього завдання може знайти застосування в криміналістиці, антитерористичному моніторингу, контр-розвідці, радіо-розвідці, комп’ютерній безпеці, медицині, торгівлі, забезпеченні безпеки доступу до фізичних об'єктів, інформаційних та фінансових ресурсів. [3]

Розпізнавання диктора поділяється на два напрямки: ідентифікацію та верифікацію. При верифікації користувач пред’являє свій ідентифікатор, і системі треба підтвердити чи відкинути його. До того ж в більшості випадків у підтвердженні

ідентифікаторів зацікавлений сам користувач, тому він не вносить в мовленнєві паролі варіації, які були відсутні в період навчання.

При ідентифікації диктор не вказує свій ідентифікатор і система розпізнавання має встановити, чи належить мовленнєвий сигнал голосу диктора, який був серед тих, які пройшли навчання.

1.2. Властивості людського голосу

Людський голос — це складний біологічний сигнал, який утворюється із взаємодії скорочення та вібрації голосових зв'язок з емісією легеневого повітря і його потоком через резонансні структури. [4]

Якщо розглядати будову мовленнєвого апарату як акустичної системи, то його зручно представити у вигляді трьох блоків:

- генератор: повітряний резервуар (легені), м'язова система, канал виведення (трахея та гортанна трубка);
- вібратори: голосові зв'язки;
- резонатори: горлянка, ротова на носова порожнини, які утворюють артикуляційну систему.

Частотним діапазоном мови людини вважається інтервал з 500 до 2000 Гц. Саме такий діапазон використовується для передачі мови в телефонних системах. Найбільш вживаним діапазоном є частоти від 280 Гц до 3,3 кГц [4]. Основна частота коливань голосових зв'язок знаходиться в межах від 50 до 250 Гц для чоловіків та від 120 до 500 Гц для жінок. Однак, хоч і основна частота коливань голосових зв'язок знаходиться за межами нижньої межі згаданих вище діапазонів мови людини, сприйняття мови не спотворюється: вухо компенсує недостатню гармоніку основного тону на основі гармонік кратних частот.

1.3. Інформаційні системи голосової біометрії

Перша система голосової ідентифікації була запатентована італійським дослідницьким центром CSELT у 1984 р. [5]. Відтоді сфера розвивалася такими

компаніями як Nuance, VoiceVault, Nok Nok Labs, Sensory Inc, BioLink, VoiceTrust та багатьма іншими [6].

Системи голосової ідентифікації поділяються на такі класи: дикторозалежні, дикторонезалежні, текстозалежні, текстонезалежні (рис. 1.1).

Рис. 1.1. Класифікація систем голосової ідентифікації

Дикторозалежні орієнтуються на ознаки мовлення однієї конкретної особи чи обмеженої групи осіб, тому вони здатні ідентифікувати лише цього зазначеного диктора (або групу). Зміна диктора вимагає повторне налаштування системи.

Дикторонезалежні не залежать від мовленнєвих ознак певної особи: вони самі витягують потрібні ознаки мовлення та проводять порівняння із еталоном, тому вони можуть ідентифікувати будь-яку особу.

Текстозалежні потребують, щоб диктор, який проходить ідентифікацію, вимовив певне ключове слово або фразу (пароль), яку повинна вимовити особа, що проходить ідентифікацію.

Текстонезалежні системи не прив’язуються до паролів, для неї мають значення лише індивідуальні ознаки голосу людини, які можливо вилучити із будь-якої фрази, вимовленої диктором.

Процес обробки голосу у системах голосової ідентифікації складається з таких етапів: попередня обробка сигналу, витягнення ознак, розпізнавання. Попередня обробка обов’язково включає зчитування аудіо з пристроїв введення та приведення

сигналу у цифровий вигляд. Опціональними, але бажаними є нормалізація та знешумлення сигналу, маркування ділянок мови. Витягнення ознак застосовує методи обробки сигналів для виділення інформативних характеристик, які повинні описувати модель мови диктора. Вони входять у вектор ознак. Етап розпізнавання може різнитися між різними системами в залежності від обраних методів, але в більшості він є порівнянням вхідного вектору ознак з еталонними. В залежності від оцінки схожості вектору до еталонної моделі приймається рішення щодо установлення особи.

1.3.1. Цифрове подання звуку

Звуковий сигнал — це тривимірний сигнал, в якому три осі представляють час, амплітуду та частоту (рис. 1.2).

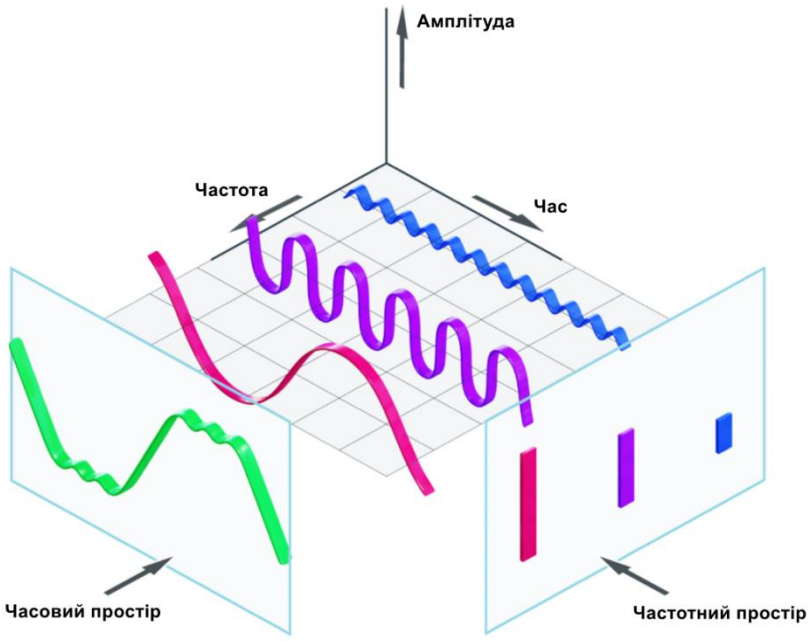

Рис. 1.2. Тривимірний звуковий сигнал

Цифрова система обробки звукового сигналу передбачає подання аналогового мовного сигналу в цифровому вигляді. Процес вилучення з сигналу чисельних значень називається квантуванням (рис. 1.3). У ряді випадків для визначення

квантування використовується поняття бітрейт (bit rate) — число бітів, що обробляються за одну секунду.

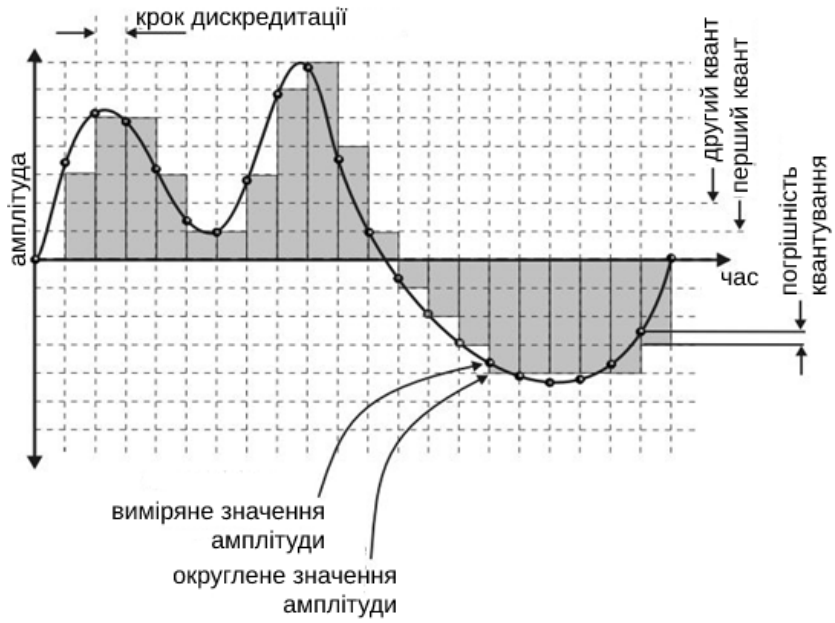

Рис. 1.3. Процес квантування сигналу

Звук складається з коливань, які при цифруванні набувають ступінчастий вигляд. Це обумовлено тим, що комп’ютер може відтворювати в будь-який короткий проміжок часу звук певної амплітуди (гучності), і цей момент не нескінченно короткий. Тривалість цього проміжку і визначає частота дискретизації. Сукупність амплітуди і короткого проміжку часу носить назву семпл. Амплітуда виражається числом, яке може займати в файлі 8, 16, 24, 32 біт (або більше).

1.3.2. Взаємозв’язок властивостей людського голосу та характеристик звукового сигналу

Мовленнєвий сигнал має складну структуру і є нестійким одразу за багатьма параметрами: тривалість фонем, темп, висота голосу. Велику роль грають індивідуальні фізіологічні особливості, активна артикуляція, емоційний стан людини. Складним завданням є вибір набору ознак, що дозволяють повно та компактно описати мовленнєвий сигнал.

Відмінності між мовцями викликані трьома джерелами варіацій:

1) відмінностями між голосовими зв’язками та розміром й формою голосового тракту (природні відмінності);

2) відмінностями в стилі мовлення (набуті відмінності);

3) відмінностями в типовому виборі слів.

Оскільки останнє джерело варіацій важко конкретизувати і, отже, важко дослідити, розпізнавання мовця зосереджується на перших двох джерелах варіацій між мовцями: природних відмінностях та набутих відмінностях (стилі мовлення). [7]

Природні відмінності — це відмінності в голосовому тракті і голосових зв’язках. Варіація розмірів та форми голосового тракту значною мірою відбивається на спектральних особливостях мовленнєвого сигналу. Наприклад, формантні частоти, резонансна частота голосового тракту відрізняються між різними особами через ці варіації. З іншого боку, відмінності в структурі голосових зв’язок відображаються впливають на частоту основного тону та інтенсивність сигналу.

Набуті відмінності є результатом різного використання артикуляторів. Ці відмінності здебільш відображаються у часовому просторі. Прикладами тривалих дикторозалежних характеристик є темп мовлення і тривалість. На більшість характеристик, які залежать від часу, мовець вплинути може, але до певної міри впливу піддаються і органічні характеристики. Отже, спосіб розпізнавання дикторів полягає в поєднанні спектральних і тривалих характеристик є надійнішим за розпізнавання з використанням лише спектральних ознак сигналу.

Окрему групу представляють параметри, що належать до вокалізованих звуків. У часовому просторі вокалізовані звуки характеризуються явно вираженою квазіперіодичністю сигналу, пов’язаною з коливаннями голосових зв’язок. Частота коливань голосових зв’язок при вимові дзвінких звуків називається частотою основного тону і є однією з індивідуальних особливостей людини, яка залежить від довжини голосових зв’язок, їх маси та натягу. Частота основного тону непостійна в процесі мовлення та може помітно змінюватися навіть у межах одного звуку, особливо для наголошених голосних. При цьому довжина і маса голосових зв’язок є вродженими, а зміна частоти від відбувається за рахунок варіювання натягу зв’язок,

на який, крім того, впливає гучність мови (при підвищенні гучності висота тону зазвичай зростає) [8]

Іншою найважливішою характеристикою вокалізованих звуків (а також будови мовленнєвого апарату конкретної людини) є формантні частоти, які є резонансними частотами акустичних порожнин мовленнєвого апарату. Резонансні частоти порожнини простої трубки постійного перерізу розташовуються через рівні за частотою інтервали ΔF [8] (формула (1.1)):

$$\Delta F = (2n - 1) \cdot \frac{c}{4L}, \tag{1.1}$$

де n – ціле позитивне число,

c – швидкість звуку у повітрі (приблизно 350 м/с),

L – довжина трубки, у дорослої людини довжина мовленнєвого тракту (від голосових зв'язок до губ) складає близько 17 см.

Переріз голосового тракту не є постійним, тому резонансні частоти, на відміну від описуваного формулою (1.1) випадку, розподіляються по різним частотним інтервалам. У чоловіків середня відстань між формантами, яка залежить від довжини голосового тракту, становить близько 1000 Гц . У жіночих і дитячих голосів середня відстань між формантами за частотою більша, ніж у чоловічих, тому що в них менша довжина акустичної порожнини. На спектрі сигналу форманти є максимумами в околах деяких частот.

Для опису основних властивостей голосного звуку достатньо визначити частоти першої та другої формант (F1 та F2).

Отже, мовленнєвий сигнал може бути описаний великою кількістю часових, спектральних, енергетичних властивостей. Питання вибору набору параметрів є нетривіальним і становить велику складність та важливість.

1.3.3. Поширені підходи до вирішення задачі та огляд існуючих рішень

Методи розпізнавання дикторів можна розділити на три основні підходи. Перший та найстаріший підхід полягає у використанні довгострокових середніх акустичних характеристик, таких як представлення спектру або висота голосу. Наприклад, в роботі [9] використовуються частота основного тону, часткові коефіцієнти автокореляції та їх статистичні характеристики, отримана точність на обмеженому датасеті — до 91%.

Ідея полягає в тому, щоб усереднювати інші фактори та залишати лише залежні від диктора компоненти. Для спектральних характеристик довгострокове середнє являє собою форму голосового тракту диктора. Цей підхід еквівалентний класифікатору процесів Гаусса і успішно використовується для текстонезалежних завдань ідентифікації мовця у роботі [10]. Однак, процес усереднення відкидає багато інформації, що залежить від мовця, і може вимагати довгих (понад 20 секунд) висловлювань для отримання стабільної довгострокової статистики мовлення.

Другий підхід полягає в моделюванні акустичних характеристик диктора в окремих фонетичних звуках. Порівняння характеристик фонетичних звуків у тестовому висловлюванні з еталонними характеристиками схожих фонетичних звуків вимірює відмінності мовців. Цей підхід може бути досягнутий використанням явної або неявної сегментації мови на фонетичні звукові класи перед навчанням моделі. До таких методів належать приховані марківські моделі, метод опорних векторів, факторний аналіз, імовірнісний лінійний дискримінантний аналіз.

Так, у роботі [11] було запропоновано кругові супрасегментні приховані марківські моделі третього порядку з використанням мел-кепстральних коефіцієнтів (MFCC) у якості вектору ознак та отримано точність розпізнавання у 99%, але вона знижувалася до 86% при поданні на вхід зашумлених даних.

Модель гаусових сумішей підпадає під підхід неявної сегментації до розпізнавання мовця. Він надає імовірнісну модель основних звуків голосу людини, але на відміну від прихованих марківських моделей не накладає жодних обмежень

між звуковими класами. Вона набула популярності у 90-х роках, та, наприклад, у роботі [12] досягла точності у 95% на невеликому датасеті.

Третій та останній підхід до розпізнавання дикторів — це використання дискримінаційних штучних нейронних мереж (ШНМ). Замість того, щоб тренувати окремі моделі для представлення окремих дикторів, дискримінаційні ШНМ навчаються моделювати функцію прийняття рішень, яка найкраще розрізняє мовців з відомого набору. Різні види мереж, такі як ШНМ з часовою затримкою [13], локально-з'єднані ШНМ [14], згорткові ШНМ [14, 15] та мережі довгої короткострокової пам'яті [16], застосовувалися до завдань верифікації чи ідентифікації особи. Як правило, ШНМ вимагають меншої кількості параметрів, ніж незалежні моделі дикторів, та мають високу продуктивність динаміків. Основним недоліком багатьох методів ШНМ є те, що всю мережу потрібно перенавчати, коли до системи додається новий диктор.

У більшості згаданих робіт у якості ознак мови використовувалися MFCC. Варто відзначити, що окрім них із звукового сигналу можна отримати й інші ознаки, які так чи інакше описують голос мовця.

1.4. Постановка задачі

Метою роботи є дослідження інформаційних методів обробки людської мови та розробка системи ідентифікації дикторів за записами їх мовлення в аудіофайлах.

У рамках роботи необхідно:

- сформувати перелік необхідних та значущих характеристик аудіосигналу для подальшої обробки;
- сформувати вхідний набір даних (датасет);
- провести попередню обробку даних;
- розробити та застосувати алгоритми витягнення характеристик із даних для формування векторів ознак;
- розробити архітектуру класифікатора за сформованими векторами ознак та навчити його, оцінити точність;
- розробити програмний застосунок для демонстрації роботи класифікатора.

Розроблена система повинна демонструвати точність вище 90% на тесто-вій вибірці при розпізнаванні наперед обраних класів.

До програмного застосунку поставлені наступні функціональні вимоги:

1. Коректне розпізнавання диктора у більшості випадків.
2. Застосунок має надавати можливість користувачеві відкрити необхідний файл з мовою мовця (лише у форматі WAVE).
3. Можливість відтворення відкритого аудіо і показ повідомлення про помилку, якщо аудіо не було відкрито перед спробою відтворити.
4. Можливість знешумлення відкритого аудіо і показ повідомлення про помилку, якщо аудіо не було відкрито. Відмова зменшувати шум в аудіо, яке вже було знешумлене.
5. Можливість запуску процесу розпізнавання особи і показ повідомлення про помилку, якщо аудіо не було завантажено. Коли особа ідентифікована, програма має показати користувачеві розпізнаний клас (ім’я диктора).
6. Якщо під час будь-якого з перелічених процесів виникла помилка, застосунок не може закритися або завершити роботу. Він повинен повідомити про це користувача та продовжити обробку ввідних запитів.
7. Можливість редагування налаштувань користувача.

До нефункціональних вимог відносяться:

1. Зручний графічний інтерфейс.
2. Процес розпізнавання не повинен перевищувати 1 хвилину. Інші дії користувача, як відкриття та прослуховування аудіофайлу, повинні мати час відповіді менше 5 секунд.

1.5. Висновки

У даному розділі було розглянуто і проаналізовано технології голосової ідентифікації особи, їх методи, особливості та обмеження, поточний стан досліджень з теми. Охарактеризовано людський голос як з біологічної точки зору, так й з технічної, описано основні характеристики голосу та їх взаємозв’язок із властивостями звукового сигналу.

Метою було поставлено дослідження інформаційних методів обробки людської мови та створення застосунку для ідентифікації особи за голосом. Формалізовано функціональні та нефункціональні вимоги до системи та окреслено базові етапи розробки.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ГОЛОСОВОЇ ІДЕНТИФІКАЦІЇ

2.1. Архітектура системи розпізнавання

Розроблювана технологія голосової ідентифікації являє собою «чорний ящик», який приймає на вхід WAVE-файл, а на виході надає клас (ім'я диктора), до якого було віднесено вхідні дані. Система складається з трьох блоків: попередньої обробки, витягнення ознак та класифікації (рис. 2.1).

Рис. 2.1. Архітектура системи розпізнавання

Блок попередньої обробки відповідає за приведення вхідних даних до придатнішого для подальшої роботи вигляду. Блок витягнення ознак шляхом методів обробки сигналу розраховує інформативні характеристики голосу диктора. Блок класифікації за цими характеристиками вирішує, якому диктору належить вхідне аудіо. Варто відзначити, що цей блок складається з трьох складових: двох класифікаторів та одного агрегатора результатів класифікації. Таке рішення було прийнято для покращення точності розпізнавання.

Детальніше про кожен з блоків йдеться в підрозділах 2.4—2.6.

2.2. Вибір засобів та інструментів розробки

Мовою програмування обрано Python. Це високорівнева мова, яка підтримує безліч бібліотек і фреймворків, в тому числі інструменти для обробки аудіо та машинного навчання. Для використання було обрано такі бібліотеки як PyQt5, Numpy, Keras, Parselmouth, Scipy, Scikit-learn, Pandas та Librosa.

Для розробки алгоритмів обробки аудіо, витягнення ознак та навчання класифікаторів було обрано хмаровий інтерактивний редактор коду Kaggle Kernels на сайті kaggle.com [17]. Він надає безкоштовний доступ до потужних CPU та GPU, завдяки чому процес обробки великого обсягу файлів на навчання класифікаторів значно пришвидшується.

Для розробки програмного віконного застосунку було обрано інтерактивне середовище розробки PyCharm Community та редактор віконного інтерфейсу Qt5 Designer.

2.3. Вхідні дані для розроблюваної системи та їх аналіз

Якість системи ідентифікації сильно залежить від навчального набору даних — датасету. При замалому обсязі стає складніше виділити спільні характеристики з сигналів, записаних у різних умовах, система перенавчається на обмежених екземплярах. Більші датасети з великим міжкласовим розмаїттям (об’єкти одного класу достатньо відрізняються між собою) дозволяють отримати більше інформативних ознак, але потребують значних обчислювальних затрат.

Вибірка даних для розроблюваної системи базується на датасеті CSTR VCTK Corpus — 11-гігабайтному корпусі мови 110 дикторів з різними акцентами англійської [18]. Усі записи були передискредитовані до 48 кГц. Спочатку цей корпус був призначений для систем синтезу мовлення з тексту на основі прихованих марківських моделей, особливо для адаптивного до мовлення синтезу мовлення на основі ПММ, який використовує середні голосові моделі, навчені на кількох динаміках та технологіях адаптації динаміків.

Із корпусу для навчання класифікаторів було виділено 43832 аудіо файли від 108 дикторів (рис. 2.2). Такий об’єм у машинному навчанні вважається середнім за обсягом. На кожного диктора приходиться від 172 до 503 файлів.

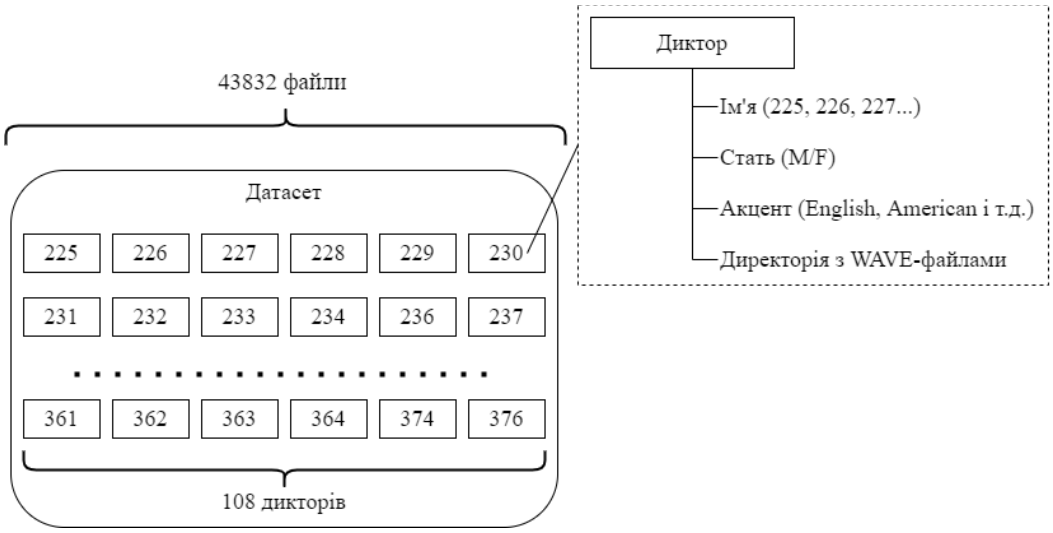

Рис. 2.2. Набір даних для навчання системи

У датасеті також були надані такі метадані, як стать та акцент диктора. Статистика щодо кількості мовців, які належать до певної статі та мають певний акцент представлені на рис. 2.3 та рис. 2.4 відповідно.

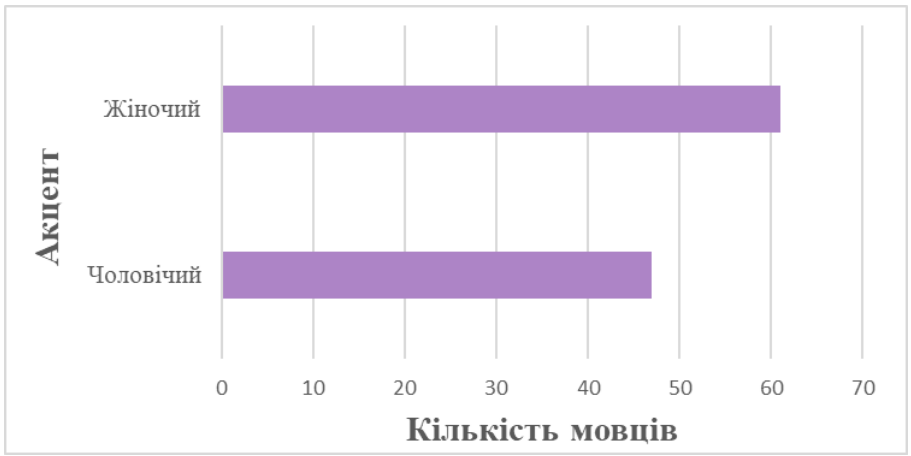

Рис. 2.3. Статистика розподілу обраних дикторів за статтю

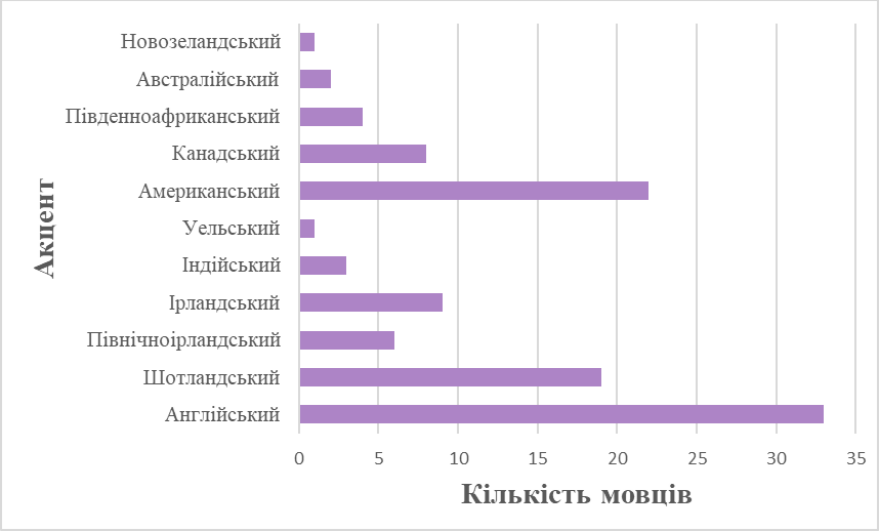

Рис. 2.4. Статистика розподілу обраних дикторів за акцентом

2.4. Попередня обробка даних

Процес попередньої обробки даних представлено на рис. 2.5. Першим кроком після зчитування усіх WAVE-файлів із набору даних була їх передискретизація. Частота дискретизації кожного аудіо файлу була приведена до 8000 — це достатньо, щоб отримати інформативні ознаки та скоротити час, необхідний на підрахунки ознак. Отриманий сигнал нормалізувалися у межах [-1;1].

Рис. 2.5. Етапи попередньої обробки даних

Для ізоляції голосу — усунення непотрібних звуків і шумів, — використовувався алгоритм зменшення шумів, оснований на методі спектрального стробування. Він працює шляхом обчислення спектрограми сигналу та оцінки порогу шуму (або вентиля) для кожної смуги частот цього сигналу/шуму [19]. Цей поріг використовується для обчислення маски, яка блокує шум нижче порогу, який може змінюватися.

Дані нормалізуються ще раз та запускається процес відсікання тиші. У два етапи — з початку сигналу та з його кінця — почергово перебиралися ділянки, де бралася амплітуда та порівнювалась із порогом. Якщо амплітуда була нижче за поріг, то ділянка вважалася тишею і відсікалася. Якщо амплітуда вже перевищувала поріг, то вважалося, що діапазон тиші закінчився, та далі з цього кінця вже нічого не видалялося. Поріг розраховувався за формулою (2.1):

$$thres = \frac{|\max(S)| - |\mathrm{mean}(S)|}{d}, \tag{2.1}$$

де S — це сигнал у часовому просторі (залежність амплітуди від часу),
d— наперед заданий коефіцієнт, який визначає чутливість порогу (чим більше, тим менш чутливим стає поріг); для даного процесу обробки брався $d = 20$.

Після відсікання тиші дані нормалізувалися в останній раз, та на цьому етап попередньої обробки завершується.

2.5. Побудова вектору ознак

2.5.1. Вибір акустичних характеристик

У даній роботі із аудіофайлу витягуються чотири види ознак: частота основного тону (або фундаментальна частота F0), формантні частоти F1 та F2, спектральній центроїд, та мел-кепстральні коефіцієнти (MFCC).

Частота основного тону. Частота коливань голосових зв’язок при проголошенні дзвінких звуків називається частотою основного тону (F0) (рис. 2.6) і є однією з

індивідуальних особливостей людини, що залежить від довжини голосових зв'язок, їхньої маси та натягу.

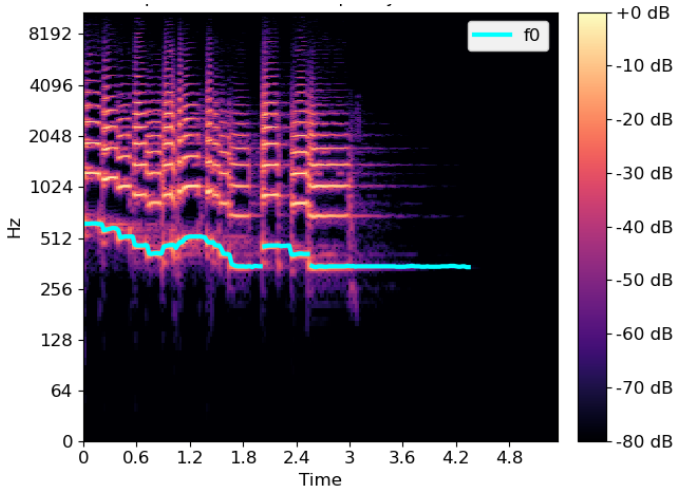

Рис. 2.6. Частота основного тону на спектрограмі сигналу

Формантні частоти. Форманта — це концентрація акустичної енергії навколо певної частоти мовленнєвої хвилі. Існує кілька формант (рис. 2.7), кожна з яких має різну частоту, приблизно по одній у кожній смузі 1000 Гц для середньостатистичних чоловіків, 1100 Гц — для жінок. Справжній діапазон залежить від фактичної довжини голосового тракту. Кожній форманті відповідає резонансний режим голосового тракту [20].

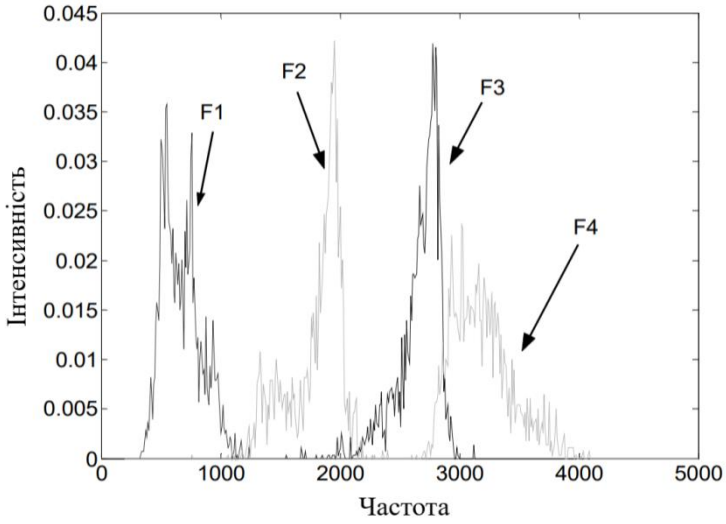

Рис. 2.7. Формантні частоти сигналу

Форманти зазвичай позначаються F1, F2, F3 і т.д. Формантні частоти F1 та F2 широко використовуються у системах розпізнавання мови завдяки їх взаємозв’язку із артикуляційними ознаками, які різняться для різних голосних звуків; а також і в системах розпізнавання дикторів, адже вони залежать від індивідуальної форми мовленнєвого апарату людини.

Спектральний центроїд. Спектральний центроїд вказує, де знаходиться «центр маси» звуку, і обчислюється як середньозважене значення частот, присутніх у звуці (рис.). Якщо частоти в аудіо однакові протягом усього часу, тоді спектральний центроїд буде навколо центру, а якщо в кінці є високі частоти, то центроїд буде ближче до його кінця.

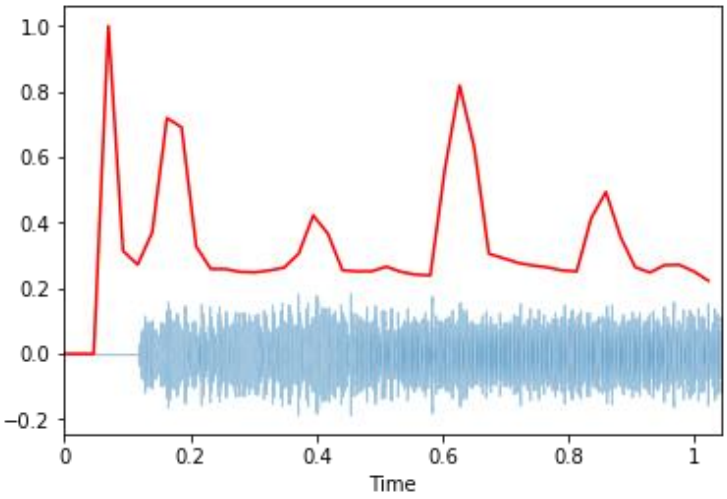

Рис. 2.8. Спектральний центроїд

Мел-кепстральні коефіцієнти. Шкала Мела — це логарифмічне перетворення частоти сигналу. Основна ідея полягає в тому, що звуки на однаковій відстані за шкалою Мела сприймаються як такі, що знаходяться на однаковій відстані від людей. Перехід до шкали Мел є актуальним рішенням у технологій обробки мови, так як це зволяє наблизитися до більш повної імітації сприйняття звуку людським вухом та відсіяти невпливові на рішення дані.

Перехід до Мел дозволяє шляхом дискретного косинусного перетворення та логарифмування спектру обчислити мел-кепстральні коефіцієнти (MFCC), які широко використовуються у розпізнаванні аудіо-сигналів (рис. 2.9). MFCC сигналу

— це невеликий набір ознак, які коротко описують загальну форму спектральної оболонки.

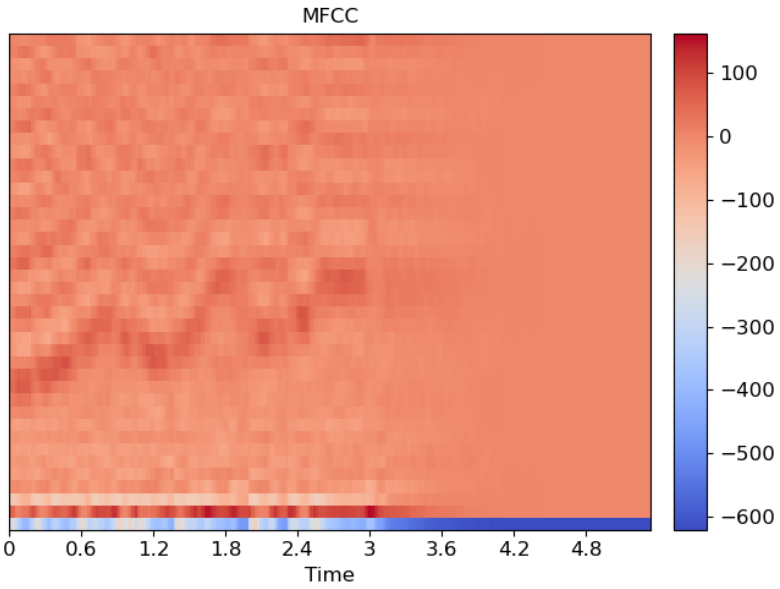

Рис. 2.9. Мел-кепстральні коефіцієнти

2.5.2 Обчислення акустичних характеристик

Для отримання MFCC вхідний сигнал представлявся у частотному просторі у вигляді спектрограми. Вони отримується дискретним перетворенням Фур'є сигналу (формула (2.2) [21]):

$$X(k) = \sum_{t=0}^{T-1} x(t) e^{-i\frac{2\pi}{T}kt}, \tag{2.2}$$

де $X(k)$ — комплексе значення амплітуди та фази вхідного сигналу;

k — індекс частоти;

T — кількість значень сигналу, які були виміряні за період, а також кількість компонент розкладання.

Одиниця мел пов'язана з герцями (f) формулою (2.3) [21]:

$$mel(f) = 1127 \ln\left(1 + \frac{f}{700}\right), \tag{2.3}$$

де f – частота звука;

$mel(f)$ – значення висоти у мелах.

Це перетворення застосовується до спектрограми сигналу для отримання мелспектрограми. Вже з неї і отримуються MFCC.

Алгоритм отримання мел-кепстральних коефіцієнтів складається з п'яти основних кроків:

1) розбиття сигналу на фрейми і застосування віконної функції;

2) отримання модулів коефіцієнтів дискретного перетворення Фур'є;

3) перехід до мел-простору частот;

4) застосування банку мел-фільтрів;

5) застосування дискретного косинусного перетворення.

Для скорочення розмірності і згладжування MFCC застосовується банк фільтрів. Вектор індивідуальних ознак $C_n[j]$ буде складатися з цих мел-кепстральних коефіцієнтів. Вони обчислюються за формулою (2.4) [21]:

$$C_n[j] = \sum_{k=0}^{K-1} \log S_n[k] \cos\left(j\left(k + \frac{1}{2}\right)\frac{\pi}{K}\right), 1 \le j \le K - 1, \tag{2.4}$$

де C_n — вектор індивідуальних ознак;

J — бажана кількість коефіцієнтів;

S_n — масив обчислених спектральних потужностей (квадрат модулів коефіцієнтів дискретного перетворення Фур'є) із застосуванням банку фільтрів.

У якості J — кількості коефіцієнтів, було взято 20. Для отримання одновимірного вектору замість двовимірної матриці коефіцієнти було усереднено за віссю частот.

Частота основного тону отримувалася за допомогою алгоритму PYIN. Імовірнісний алгоритм YIN (PYIN) є модифікацією відомого алгоритму YIN для оцінки F0 [22]. Звичайний YIN є простим, але ефективним автокореляційним методом для покадрової монофонічної оцінки F0. Щоб усунути короткочасні помилки, вихідні дані оцінювачів частоти, як правило, піддаються постобробці, що призводить до більш плавного ходу.

F0 приймає вигляд вектору частот в залежності від проміжків часу. З цього відбираються для опису максимальне, мінімальне та середнє значення у проміжку 80 Гц — 450 Гц (мінімальна та максимальна частоти, на яких фізично здатні коливатися голосові зв’язки людини).

Формантні частоти F1 та F2 знаходилися за допомогою бібліотеки обробки аудіо parselmouth, що є обгорткою для модулів ПЗ Praat [23, 24]. Його алгоритм використовує кодування з лінійним предиктором. Повертаються два вектори формант — F1 та F2, — які можуть варіюватися за обсягом для різних аудіо.

До вектора ознак були відібрані такі дані формат: середня F1, медіанна F1, середня F2, медіанна F2, чотири центри кластерів обох формант, скомпонованих на двовимірній координатній площі (вісь абсцис — F1, Гц, вісь ординат — F2, Гц) (рис. 2.10). Для кластеризації був обраний метод k-середніх.

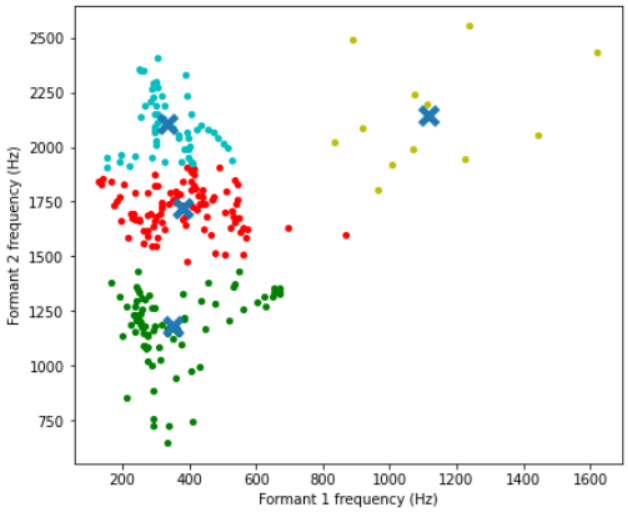

Рис. 2.10. Приклад знайдених кластерів формант (центри кластерів позначені хрестиками)

Спектральний центроїд розраховувався за допомогою бібліотеки обробки аудіо librosa [25]. З нього також відбиралися максимальні, мінімальні та середні значення.

Перед використанням даних було відкинуто екземпляри, які задовольняють наступним ознакам:

1. Max F0 ≈ Min F0 — людина не може говорити завжди на одних й тих же частотах. Такі показники говорять о невдалій оцінці частоти основного тону і не підходять для класифікації.

2. F0 в межах [80 Гц; 450 Гц] не знайдено — голосові зв'язки людини не можуть коливатися за межами цього проміжку. Якщо оцінена фундаментальна частота на усіх проміжках сигналу не входить в нього, то не має сенсу використовувати цей екземпляр даних.

3. Екземпляри, в яких не вдалося визначити жодної форманти, — такі аудіо файли є інформативними (речення були промовлені нерозбірливо і тихо, замість речення окремим файлом виявився лише запис шумів чи тиші тощо).

В результаті з 43832 файлів на етап навчання класифікаторів потрапило 43826, тобто неінформативними виявилися шість файлів. Вектор ознак складався з 38-ми дійсних чисел.

2.6. Ідентифікація особи — побудова блоку класифікації

Ідентифікація дикторів за векторами ознак аудіофайлів із мови є задачею класифікації. Математична постановка задачі формулюється наступним чином.

Для X— множини векторів описів об'єктів класифікації — та Y— множини найменувань класів — існує деяка залежність — зображення $y^*: X \to Y$, — значення якої відомі лише на об'єктах кінцевої навчальної вибірки $X^m: \{(x_1, y_1), \dots, (x_m, y_m)\}$. Необхідно побудувати алгоритм $a: X \to Y$, який здатний класифікувати довільний об'єкт $x \in X$.

2.6.1. Класифікація багатошаровими нейронними мережами

Класифікатором було обрано багатошарову повнозв'язну ШНМ прямого поширення. Її архітектура представлена на рис. 2.11. Обраний вид ШНМ був створений для класифікації, тому його використання є доцільним.

Так як вектор ознак є одновимірним і складається всього з 38 ознак, то глибока мережа з великою кількістю шарів та параметрів призведе лише для перенавчання. В процесі дослідження та експериментів було побудовано та протестовано декілька ШНМ, серед яких обрано найкращу — з точністю на тестовій вибірці вище 90% та без ознак значного перенавчання. Детальніше результати тестування різних мереж викладені в розділі 3.

Обрана мережа має два приховані шари по 128 та 256 нейронів відповідно із сигмоїдною функцією активації. Усього параметрів мережі — 66436, з яких навчаються — 66104. Вхідні дані нормалізувалися на шарах батч-нормалізації, а для запобігання перенавчання було використано шари проріджування — присвоєння нулів випадково обраним ознакам у процесі навчання. Коефіцієнт проріджування — це доля ознак, які обнуляться (в даному випадку 0,2 та 0,3).

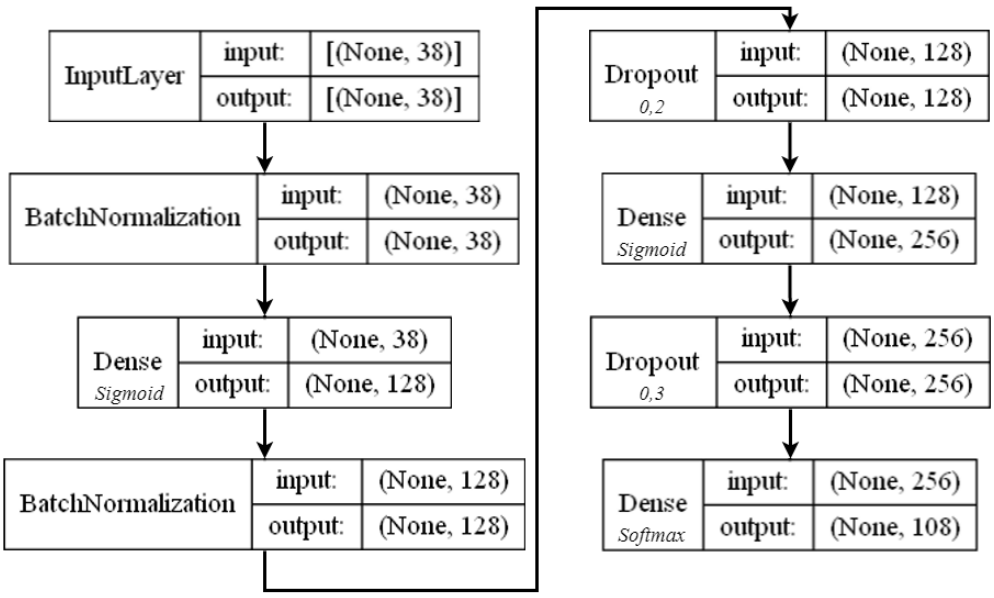

Рис. 2.11. Архітектура штучної нейронної мережі

Датасет був розбитий на тренувальну (70%), валідаційну (15%) та тестову (15%) вибірки. Для навчання ШНМ у якості функції втрат була обрана категоріальна перехресна ентропія. Оптимізатором виступив метод Adam.

Побудована ШНМ навчалася протягом 100 епох, кожна епоха складалася з 480-ти кроків. На кожному кроці подавалися набори по 64 екземпляри. Після навчання модель разом із вагами було збережено у форматі файлу h5 для подальшого використання. Результати тестування мережі на навчальній та тестовій вибірках детальніше викладені у розділі 3.

2.6.2. Ідентифікація з використанням моделей гаусових сумішей

Модель гаусових сумішей — параметрична модель, яка дає розподіл ймовірностей векторів ознак різних дикторів. Модель гаусових сумішей представляє певний набір даних у вигляді суміші або зваженої суми кількох індивідуальних гаусових розподілів [26]. Клас описується за формулою (2.5) [27]:

$$p\left(\overline{x}|\overline{\lambda}\right) = \sum_{i=0}^{M} w_i p_i(\overline{x}), \qquad \sum_{i=0}^{M} p_i = 1 \tag{2.5}$$

де $\overline{x}$ — N-вимірний вектор ознак;

M — кількість нормальних розподілів;

w_i — ваги компонентів моделі;

p_i — багатовимірні функції щільності розподілів складових.

Оцінка належності вхідного вектору ознак до класу розраховується методом максимізації правдоподібності (2.6) [27]:

$$p(\overline{x} \in \lambda) = \sum_{i=0}^{M} w_i p_i(\overline{x}) \tag{2.6}$$

де λ — це модель, до якої розраховується ймовірність відповідності.

2.6.2.1. Розширення вектору ознак мовця

Найрозповсюдженішими вхідними даними з аудіо для побудови моделей гаусових сумішей є MFCC завдяки повноті опису сигналу. До 20 ознак MFCC також було вирішено додати їх дельта-коефіцієнти (або похідну), які характеризують швидкість енергії та MFCC. Вони знаходяться за формулою (2.7):

$$d_n = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}, \tag{2.7}$$

де d_t — дельта-коефіцієнт з моменту часу t,
N — натуральне число (зазвичай дорівнює 2),
c_{t+n} — мел-кепстральний коефіцієнт з моменту часу $t + n$ (c_{t-n} — аналогічно).

В результаті отримано вектор ознак з 40-а характеристик.

2.6.2.2. Побудова моделей гаусових сумішей

Вибірку будо поділено на тренувальну (80%) та тестову (20%) Для кожного диктора з усіх його файлів з тренувальної вибірки було розраховано та збережено саму модель. В результаті отримано 108 файлів формату gmm. Класифікатор, заснований на цих моделях, послідовно оцінює належність вхідного вектору ознак до кожного з класів за формулою (2.6). Клас, модель якого надала максимальну оцінку вектору ознак, і є класом, до якого відносяться вхідні дані.

2.6.3. Агрегація результатів від штучної нейронної мережі та моделей гаусових сумішей у єдиний розпізнаний клас

Нейронна мережа та моделі гаусових сумішей працюють на різних входах та виявляють різні закономірності. Мова особи є багатоаспектною, і жодний класифікатор не здатний повністю охопити всі її закономірності. Використання двох

класифікаторів розширює область властивостей мови, які здатна охопити система розпізнавання.

Для остаточної відповіді результати від обидвох класифікаторів агрегуються. Якщо максимальні оцінки були надані одному й тому ж самому диктору, то додаткові дії не потребуються, інакше за формулою (2.8) обчислюються оцінки рішення класифікаторів. Класифікатор, який отримав більшу оцінку, вважається правим, та його відповідь і є розпізнаною особою.

$$F_A = \frac{p_i^A \cdot w_A}{\max(p^A) - \min(p^A)} - \frac{w_B}{(\max(p^B) - \min(p^B)) \cdot p_i^B}, \quad (2.8)$$

де:

F_A — оцінка рішення класифікатора A;

p^A , p^B — вектори оцінок належності вхідного вектору до кожного класу від класифікаторів A та B відповідно;

w_A, w_B — вагові коефіцієнти для класифікаторів A та B відповідно, які залежать від точності класифікатора та його вихідного вектору оцінок (для ШНМ $w = 0.2$, для МГС $w = 0.8$);

i — індекс класу-«переможця» A;

p_i^A — оцінка класу-«переможця» A від класифікатора A;

p_i^B — оцінка класу-«переможця» A від класифікатора B.

2.7. Висновки

У даному розділі було спроектовано архітектуру системи розпізнавання особи за голосом, обрано властивості сигналу для використання у якості інформативних ознак мови, специфіковано методи та особливості обробки сигналу та розпізнавання особи. Було визначено математичні та інформаційні засоби й інструменти реалізації програмного застосунку. Обрано набір вхідних даних.

В результаті експериментів змодельовано та навчено багатошарову нейронну мережу прямого поширення для задачі розпізнавання особи. Також розраховано

моделі гаусових сумішей кожного з класів мовця. Специфіковано метод агрегації виходів від обидвох класифікаторів для отримання остаточного результату.

РОЗДІЛ 3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ СИСТЕМИ ІДЕНТИФІКАЦІЇ

3.1. Тестування та порівняння моделей багатошарових нейронних мереж

Датасет був розбитий на тренувальну, валідаційну та тестову вибірки. Тренувальна та валідаційна напряму брали участь у навчанні ШНМ, тому вони вже «бачені» класифікатором. Результати роботи класифікатора на тестовій вибірці є показником того, як система буде працювати з раніше не баченими даними, тобто як вона теоретично має працювати при практичному застосуванні.

У ході розробки було протестовано декілька архітектур ШНМ, та результати тестування наведені в табл. 3.1. Курсивом виділено архітектуру, яка була обрана для використання у фінальній версії системи.

Таблиця 3.1

Результати тестування нейронних мереж

№	Кількість прихованих шарів	Функції активації	Запобігання перенавчанню	Кількість параметрів, що навчаються	Точність на тренувальній та валідаційній вибірках	Точність на тестовій вибірці
1	3	relu, softmax	2 шари проріджения (0.4, 0.5)	71,928	92%	88%
2	3	sigmoid, relu, softmax	2 шари проріджения (0.3, 0.4)	132,408	97%	90%
3	*2*	*sigmoid, softmax*	*2 шари проріджения (0.2, 0.3)*	*66,104*	*95%*	*91%*
4	2	sigmoid, relu, softmax	2 шари проріджения (0.1, 0.3); регуляризатор	13,752	88%	85%
5	1	relu, softmax	шар проріджения (0.2); регуляризатор	9,592	88%	86%

Критеріями для відбору були точність вище 90% на тестовій вибірці на незначні ознаки перенавчання. Першій умові відповідають ШНМ № 2 та 3. ШНМ №4 та 5 показують, що при занадто малій кількості параметрів, що навчаються (вагів), та прихованих шарів, точність і на тренувальній, і на тестовій вибірках сильно падає. Це означає, що мережа не могла більш повно описати зв'язки між характеристиками сигналу мови достатньо повно.

Між ШНМ №2 та 3 вибір пав на №3. Хоча ШНМ №2 й демонструє вищу точність на тренувальній вибірці, її різниця з тестовою є вищою, що свідчить про перенавчання. ШНМ №2 має велику кількість параметрів, та завдяки цьому її ваги були сильно підігнані під тренувальні дані. Використовувати перенавчені мережі на практиці не є доцільним. ШНМ №3, однак, має меншу різницю між точністю на тренувальній та тестовій вибірках та й найкращі результати на тестовій серед всіх ШНМ, тому є найбільш придатною до використання.

3.2. Тестування моделей гаусових сумішей

Для тестування моделей гаусових сумішей файли з тестової вибірки були оцінені на схожість кожною з 108 моделей дикторів, та модель, що надала найбільшу оцінку, вказувала на клас, до якого належать вхідні дані.

В результаті на тестовій вибірці було отримано точність 93%. На тренувальній, в свою чергу, — 95%.

3.3. Порівняння штучних нейронних мереж прямого та моделей гаусових сумішей. Оцінка доцільності комбінування класифікаторів

За результатами тестування з пунктів 3.1 та 3.2 визначено, що моделі гаусових сумішей демонструють трохи вищу точність у порівнянні з нейронною мережею. Однак, замість вибору одного найкращого класифікатору була реалізована спроба об'єднати виходи від обох для покращення точності розпізнавання. Так як класифікатори виявляють відмінні патерни, що описують мову диктора, прийняття рішення з врахуванням обох відповідей має потенціал подолати недоліки обох класифікаторів. Доцільність використання такої агрегації має підтвердитись

підвищеною точністю розпізнавання порівняно з попередніми експериментами та вибором правильної відповіді в більшості випадків, коли класифікатори на виході мають різні розпізнані класи.

Результат тестування представлені у табл. 3.2. З всієї тестової вибірки класифікатори давали різні відповіді у 15,7% випадків. З цих спірних випадків частіше пріоритет надавався відповідям моделей гаусових сумішей (54,4%). Варто відзначити, що коли розпізнаним класом вважалася відповідь від ШНМ, то у 90% випадках таке рішення було вірним, але у випадку з МГС така частка вірних рішень падає до 67%. Це може бути пов’язано з вагами, призначеними класифікаторам (так як МГС показали більшу загальну точність, то вони мали більшу вагу).

Загалом у спірних випадках правильне рішення приймалося у 77,6% з них, та це підвищило остаточну точність розпізнавання до 94,8% — на 1,8% більше, ніж максимальна точність МГС. Це означає, що необхідність врахування відповідей обох класифікаторів підтвердилася, адже це покращує роботу системи розпізнавання.

Таблиця 3.2

Результати тестування агрегації виходів класифікатора

Класиф.	Точність на тестовій вибірці	Частка різних відповідей	Точність після агрегації	Класифікатор обрано (% від випадків різних відповідей)	Частка вірних виборів	Всього вірних виборів
ШНМ	91%	15,7%	94,8%	45,6%	90%	77,6%
МГС	93%			54,4%	67%	

3.4. Висновки

У даному розділі було протестовано блок класифікації системи розпізнавання. Воно включало в себе тестування декількох архітектур ШНМ для вибору найкращої, тестування класифікації за допомогою МГС та агрегації виходів обох класифікаторів.

Серед п’яти розроблених ШНМ було обрано модель №3, яка на тестовій вибірці показала точність 91%. Розпізнавання за допомогою МГС, в свою чергу, мало вищу точність — 93%.

На етапі розробки блоку класифікації було зроблене припущення, що комбінування ШНМ й МГС та зважене прийняття рішень у випадках, коли вони дають різні виходи для одного й того ж вхідного вектору, має підвищити якість розпізнавання. Припущення підтвердилося: в таких ситуаціях правильні рішення приймалися в 77,6% випадків, що підвищило точність розпізнавання до 94,8%.

Таким чином, якість системи розпізнавання відповідає поставленим в першому розділі вимогам, та отримані моделі та алгоритми будуть використані в кінцевому застосунку.

РОЗДІЛ 4 РЕАЛІЗАЦІЯ СИСТЕМИ ІДЕНТИФІКАЦІЇ ОСОБИ ЯК ПРОГРАМНОГО ЗАСТОСУНКУ

Задля зручнішого тестування технології ідентифікації особи та демонстрації результатів було розроблено віконний програмний застосунок з графічним інтерфейсом користувача. Він дозволяє користувачу обрати файл, виконати знешумлення та розпізнавання диктора. За необхідності за допомогою xml-файлу конфігурацій можна замінити h5-файл натренованої моделі ШНМ, шлях до файлів моделей гаусових сумішей та налаштувати інші параметри.

4.1. Архітектура програмного застосунку

Принцип побудови застосунку дотримується об’єктно-орієнтованого архітектурного підходу. Функціональні модулі представлені класами, які мають власні набори полів та ознак, що визначають їх роль у застосунку.

Якщо частину фунціоналу потрібно змінити, це з меншою ймовірністю вплине на решту модулів. У даному випадку більшість компонентів, які охоплюють абсолютно різні цілі (наприклад, витяг характеристик з аудіофайлу або оцінку того, наскільки ймовірно вектор ознак відповідає моделям дикторів), не повинні знати про специфікації один одного. Додатку можуть знадобитися зміни, коли передбачений набір характеристик сигналу вже не актуальний або потрібно замінити моделі класифікаторів, щоб розпізнати зовсім інші класи. У таких випадках зміни обмежуються оновленням окремих компонентів та редагуванням файлу конфігурації.

UML-діаграма класів програмного застосунку представлена на рис. 4.1. До класів, які відповідають за представлення даних користувачеві та обробку його команд належать MainWindow та AudioPlayer. Усі інші класи відповідають за логіку обробки аудіо та розпізнавання особи (класи бізнес-логіки).

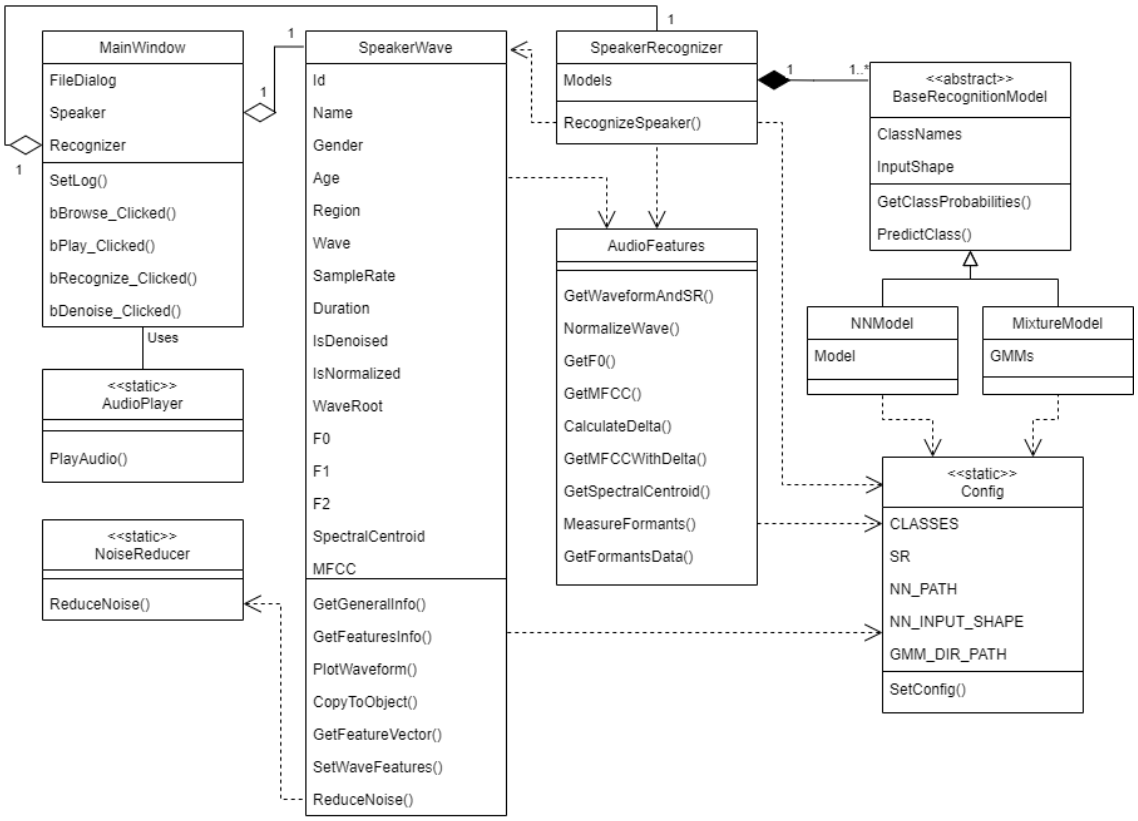

Рис. 4.1. Діаграма класів застосунку

MainWindow обробляє дії користувача в інтерфейсі, комунікації з класом AudioPlayer і класами бізнес-логіки, якщо це необхідно. Він має один екземпляр класів SpeakerWave і один екземпляр SpeakerRecognizer.

SpeakerWave містить WAVE-файл мовлення людини та його властивості. Він може включати метадані мовця (ім’я, стать, регіон, акцент, вік), деталі про файл (шлях до файлу, частота дискретизації, тривалість, сам сигнал) і розраховані голосові характеристики, які формуватимуть вхідні дані для класифікаторів. Він пов’язаний з класами AudioFeatures і статичним класом NoiseReducer для виконання відповідних функцій над собою.

SpeakerRecognizer містить один або кілька (у даному випадку два) об’єкти абстрактного класу BaseRecognitionModel — суперкласу для NNModel і MixtureModel. Використання абстракції аргументовано тим, що і ШНМ, і МГС можна узагальнити до класифікатора, який дає вихід заданої довжини, єдина відмінність полягає в реалізації. Класифікатори не знають про SpeakerRecognizer. SpeakerRecognizer викликає методи класу AudioFeatures, щоб обчислити вхідний

вектор SpeakerWave та надати його класифікаторам. Потім результати класифікаторів зважуються для отримання кінцевого результату визнання.

Config — це статичний клас, від якого залежать майже всі інші класи бізнес-логіки. Він надає доступ до конфігурацій користувача шляхом обробки XML-файлу. Конфігурації включають список класів, розмірність входу класифікаторів, шляхи до моделей класифікаторів, частоту дискретизації WAVE за замовчуванням.

4.2. Опис інтерфейсу користувача

Інтерфейс користувача був розроблений відповідно до висунутих до нього вимог. Інтерфейс містить необхідну та зрозумілу для користувача інформацію щодо роботи із застосунком, що спрощує його використання.

Графічний інтерфейс користувача складається з наступних вікон: головне вікно, діалогове вікно вибору файлу та діалогове вікно помилок. Схема переходів між вікнами представлена на рис. 4.2.

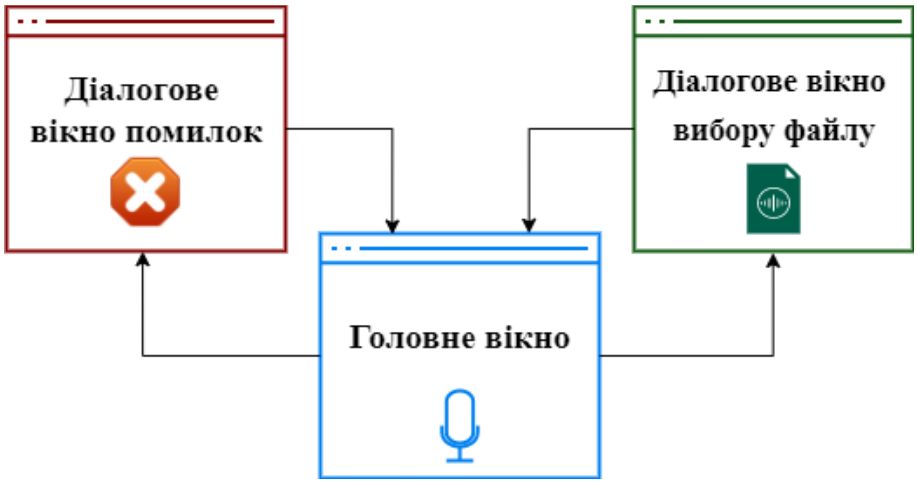

Рис. 4.2. Схема переходів між вікнами застосунку

Головне вікно застосунку представлено на рис. 4.3. Довге поле поряд з «Обрати файл» призначене для назви обраного файлу, а велике поле праворуч — для інформативних повідомлень.

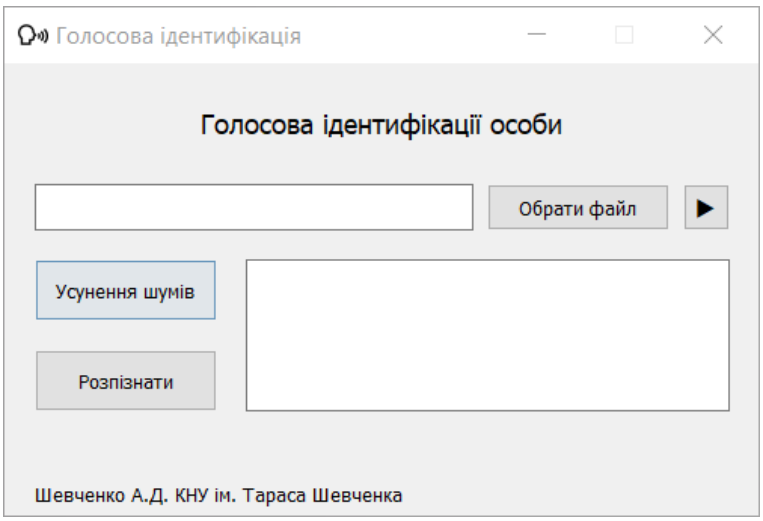

Рис. 4.3 Головне вікно застосунку

Кнопка «Обрати файл» відкриває вікно вибору файлу (рис. 4.4). Користувачу дається можливість обрати лише файл у форматі WAVE.

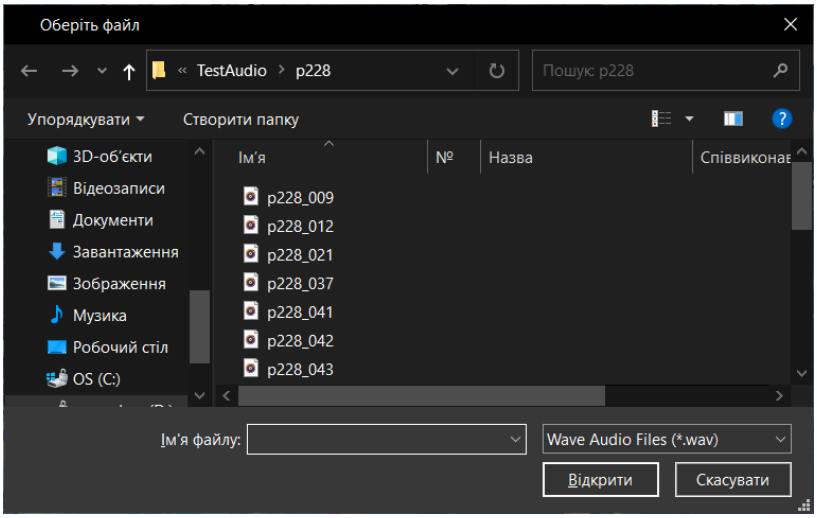

Рис. 4.4. Вікно вибору файлу

Кнопка «Усунення шумів» запускає процес знешумлення аудіо. Після його завершення з'являється повідомлення про успіх. Повторне знешумлення неможливе (рис. 4.5).

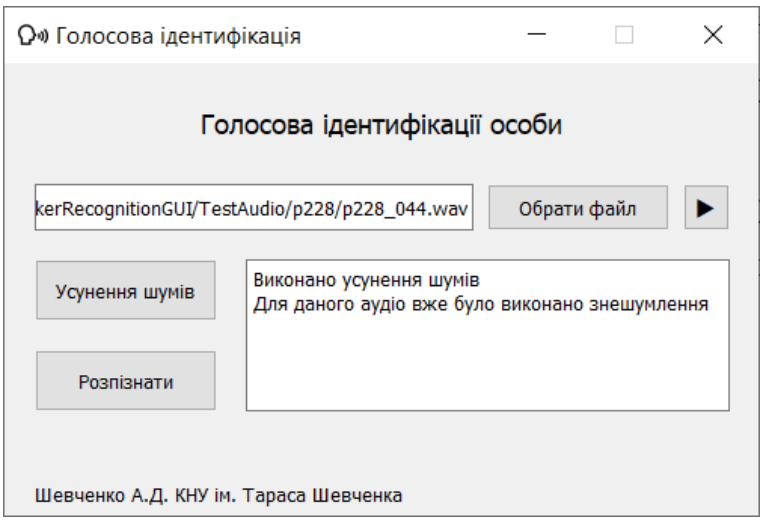

Рис. 4.5 Головне вікно після знешумлення аудіо та повторної спроби

Кнопка «Розпізнати» запускає процес розпізнавання диктора, який складається з витягнення ознак та подання ознак до класифікаторів. По закінченню з'являється повідомлення з ім'ям розпізнаної особи (рис. 4.6)

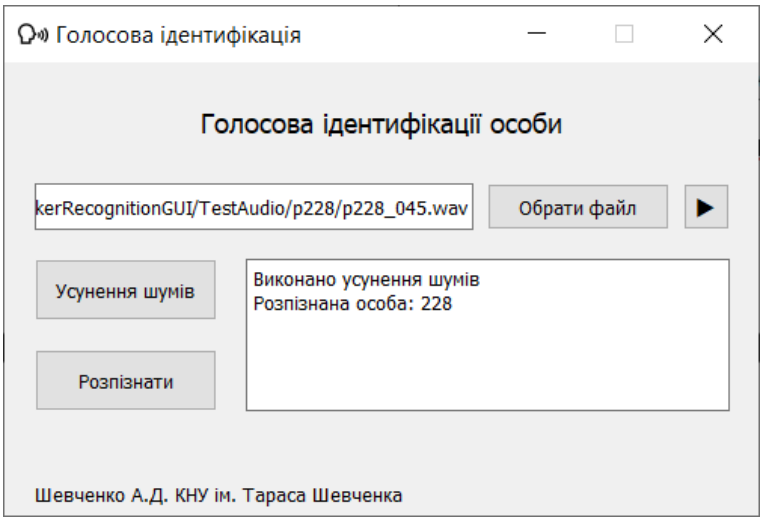

Рис. 4.6. Головне вікно після розпізнавання особи

При спробі знешумити аудіо або розпізнати особу, якщо користувач ще не обрав файл, з'являється вікно помилки (рис. 4.7). Воно також відкривається у випадку появи виключення на будь-якому етапі роботи із застосунком та містить його текст.

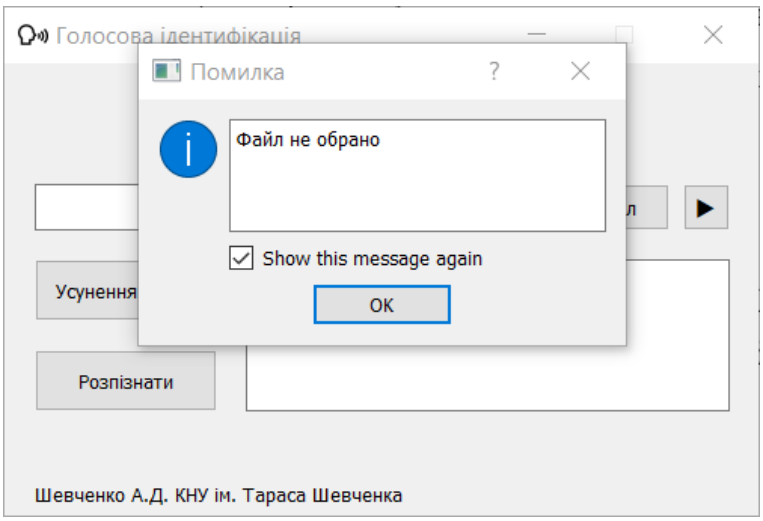

Рис. 4.7. Вікно помилки за відсутності відкритого файлу

4.3. Інструктивний матеріал користувача для експлуатації програмного застосунку системи голосової ідентифікації особи

4.3.1. Встановлення застосунку

Для того, щоб встановити ПЗ, необхідно виконати наступні кроки:

1. Встановити на ПК python (посилання на завантаження: [28]) та pip (зробити це можна за інструкцією у посиланні [29]), якщо вони не були встановлені раніше.

2. Розпакувати архів «SpeakerRecognitionApp.zip» до будь-якої теки.

3. Відкрити командну строку, перейти в ній до теки з розпакованим архівом та запустити команду (без лапок): «pip install -r requirements.txt»

4.3.2. Експлуатація застосунку

Для налаштування користувацький конфігурацій необхідно будь-яким текстовим редактором відкрити файл «config.xml», що знаходиться в теці, в яку було розпаковано архів із застосунком, та відредагувати необхідні ділянки тексту згідно з рис. 4.8.

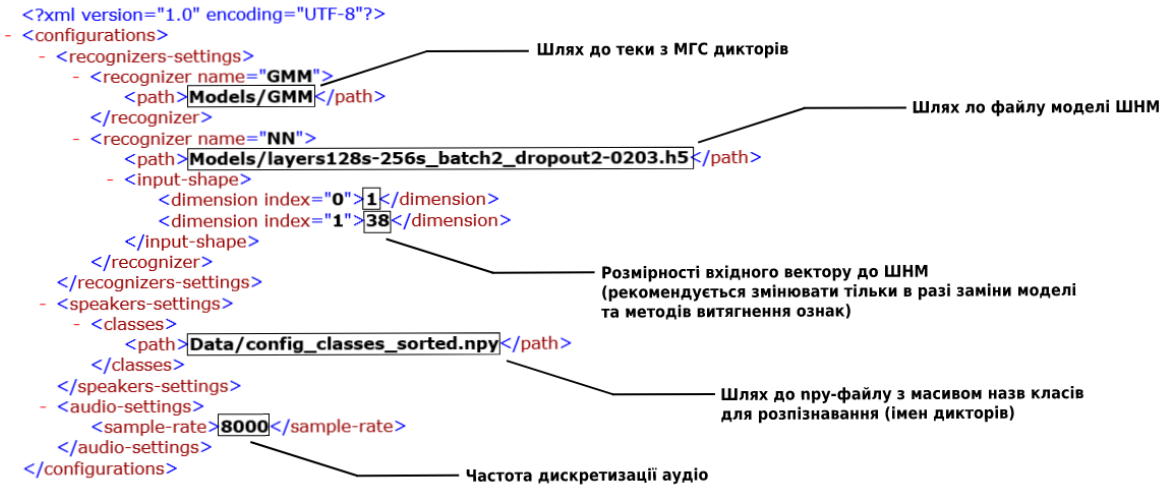

Рис. 4.8. Налаштування конфігурацій за допомогою файлу config.xml

Для відкриття застосунку треба відкрити файл «SpeakerRecognitionApp.exe», що знаходиться в теці, в яку було розпаковано архів із застосунком.

Для відкриття файлу необхідно натиснути на кнопку «Обрати файл» та відкрити необхідний WAV-файл.

Для прослуховування відкритого файлу — натиснути на кнопку «▷».

Для знешумлення відкритого файлу — на кнопку «Усунення шумів».

Для розпізнавання диктора з запису мови у відкритому файлі необхідно натиснути на кнопку «Розпізнати».

4.4. Висновки

Даний розділ було присвячено проектуванню програмного застосунку розпізнавання особи з інтерфейсом користувача. Аргументовано обрані архітектурні підходи та описано базові компоненти ПЗ. Розроблено програмний застосунок, який відповідає поставленим в першому розділі вимогам.

Детально описано інтерфейс користувача та особливості роботи з ним. Викладено інструкцію до встановлення, налаштування та експлуатації ПЗ.

ВИСНОВКИ

Метою даної випускної кваліфікаційної роботи було поставлено дослідження методів обробки аудіофайлів, методів класифікації нейронними мережами та моделями гаусових сумішей, проведення експериментальних дослідів по ідентифікації особи за її голосом. Для її досягнення досліджено стан галузі біометричної аутентифікації, особливості людського голосу та інформаційних систем розпізнавання осіб за голосом, розглянуто поширені методи вирішення задачі.

Після аналізу перелічених пунктів було спроектовано та розроблено систему голосової ідентифікації особи. Система базується на витягненні акустичних характеристик із запису мови людини та подальшому віднесенню отриманих даних до одного з наперед заданих класів (дикторів). Вона складалася з трьох блоків: блоку попередньої обробки, блоку витягнення ознак та блоку класифікації. Новизна рішення полягає у гібридному блоку класифікації, який складається з двох класифікаторів: багатошарової нейронної мережі прямого поширення й моделей гаусових сумішей, — та агрегатора їх виходів, який приймає рішення, яка саме особа була розпізнана, на основі зважених виходів обох класифікаторів.

Класифікатори було навчено на наборі даних з 43832 аудіо файлів від 108 дикторів. Багатошарова нейронна мережа на тестовій вибірці продемонструвала точність у 91%, а класифікатор на основі моделей гаусових сумішей — 93%. Дані результати доводять, що акустичні ознаки, витягнені з аудіо, достатньо повно описують форму людського голосу, а класифікатори окремо справляються із поставленою задачею, при чому моделі гаусових сумішей є більш точними.

Агрегація результатів обох виходів, яка була націлена на підвищення точності, також справилася зі своєю задачею: частка коректно розпізнаних осіб сягнула 94,8%. Це означає, що комбінування класифікаторів, які здатні знаходити різні патерни у вхідних даних, дозволяє більш повно описати унікальні характеристики голосу особи.

До системи розпізнавання було створено програмний застосунок із інтерфейсом користувача, описано його структуру, викладено інструкцію з експлуатації.

Застосунок дозволяє як і протестувати розроблену архітектуру ШНМ або побудовані моделі гаусових сумішей дикторів на класах із використаного в даній роботі набору даних, так і персоналізувати налаштування: замінити модель ШНМ, моделі гаусових сумішей, класи для розпізнавання тощо. Застосунок є придатним як і для практичного застосування для розпізнавання осіб, на даних котрих була навчена система в рамках цієї роботи, так і для демонстрації результатів досліджень в області голосової ідентифікації.

Пов’язані з даною випускною кваліфікаційною роботою тези були опубліковані під назвою «Інформаційна технологія голосової ідентифікації за біометричними показниками людини» в збірнику «Сучасні аспекти та перспективні напрямки розвитку науки: матеріали III Міжнародної студентської наукової конференції», виданому ГО «Молодіжна наукова ліга» 6 травня 2022 [30].

Голосова біометрія є актуальною темою в багатьох галузях, та отримані в даній роботі результати можуть послужити підпіллям для майбутніх досліджень та розробок, націлених на більшу точність розпізнавання та повніший опис індивідуальних голосових характеристик особи у цифровому форматі.

ЛІТЕРАТУРА

1. Fœssel M. Biométrie : les nouvelles formes de l'identité / Michaël Fœssel, Antoine Garapon // Esprit. – 2006. – Août/septembre, no. 8. – P. 165.

2. Брагина Е.К. Современные методы биометрической аутентификации: обзор, анализ и определение перспектив развития. / Е.К. Брагина, С.С. Соколов // Вестник Астраханского государственного технического университета. – 2016. – Т. 1, №61. – С. 40–45.

3. Сорокин В. Н. Распознавание личности по голосу: аналитический обзор. / В. Н. Сорокин, В. В. Вьюгин, А. А. Тананыкин // Информационные процессы. – 2012. – Т. 12, №. 1. – С. 1–30.

4. Asci F. Machine-Learning Analysis of Voice Samples Recorded through Smartphones: The Combined Effect of Ageing and Gender / F. Asci, G. Costantini, A. Zampogna et al. // Sensors. – 2020. – Vol. 20, no. 18. – P. 5022.

5. Патент US 4752958 A / Сполучені Штати Америки. Device for speaker's verification / M. Cavazza, A. Ciaramella. Подано 17.05.1984 ; опубл. 21.06.1988. – 24 с.

6. Луценко Х.В. Голосова ідентифікація диктора як один із сучасних біометричних методів ідентифікації особи / Х.В. Луценко, К. В. Нікулін // Теорія та практика судової експертизи і криміналістики. – 2019. – № 19. – С. 239–255.

7. Niessen M. Speaker specific features in vowels / Maria Niessen. – 2004. – 49 p.

8. Томчук К. К. Сегментация речевых сигналов для задач автоматической обработки речи / К. К. Томчук – С.-Петерб. гос. ун-т аэрокосм. приборостроения, 2017.

9. Furui S. Talker recognition by statistical features of speech sounds. / S. Furui, F. Itakura // Electronics & communications in Japan. – 1973. – P. 62–71.

10. Gish H. Methods and experiments for text-independent speaker recognition over telephone channels / H. Gish, M. Krasner, W. Russell, J. Wolf // IEEE International

Conference on Acoustics, Speech, and Signal Processing, Tokyo, Japan. – [S. l.], 1986. – P. 865–868.

11. Shahin I. M. A. Speaker Identification in a Shouted Talking Environment Based on Novel Third-Order Circular Suprasegmental Hidden Markov Models / Ismail M. A. Shahin // Circuits, Systems, and Signal Processing. – 2015. – Vol. 35, no. 10. – P. 3770–3792.

12. Reynolds D. A. Robust text-independent speaker identification using Gaussian mixture speaker models / D. A. Reynolds, R. C. Rose // IEEE Transactions on Speech and Audio Processing. – 1995. – Vol. 3, no. 1. – P. 72–83.

13. Yu Y.-Q. Densely Connected Time Delay Neural Network for Speaker Verification / Ya-Qi Yu, Wu-Jun Li // Interspeech 2020. – ISCA, 2020. – P. 921–925.

14. Chen Y.-h. Locally-connected and convolutional neural networks for small footprint speaker recognition / Yu-hsin Chen, Ignacio Lopez-Moreno, Tara N. Sainath // Interspeech 2015. – ISCA, 2015.

15. Salehghaffari H. Speaker verification using convolutional neural networks / Hossein Salehghaffari // arXiv:1803.05427. – 2018.

16. Ren J. Look, listen and learn—A multimodal LSTM for speaker identification / J. Ren, Y. Hu, Y. Tai, C. Wang, L. Xu // Proceedings of the AAAI conference on artificial intelligence. – 2016. – Vol. 30, no. 1. – P. 3581–3587.

17. Kaggle: Your Machine Learning and Data Science Community [Електронний ресурс]. – Режим доступу: https://www.kaggle.com.

18. Yamagishi J. CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit (version 0.92) [Електронний ресурс] / Junichi Yamagishi, Christophe Veaux, Kirsten MacDonald // Edinburgh DataShare. – Режим доступу: https://datashare.ed.ac.uk/handle/10283/3443.

19. Sainburg T. Noise reduction in python using spectral gating [Електронний ресурс] / Tim Sainburg. – 2019. – Режим доступу до ресурсу: https://pypi.org/project/noisereduce/.

20. What are formants? [Електронний ресурс] // Welcome to SWPhonetics. – Режим доступу: https://swphonetics.com/praat/tutorials/what-are-formants/.

21. Ittichaichareon C. Speech recognition using MFCC / C. Ittichaichareon, S. Suksri, T. Yingthawornsuk // International conference on computer graphics, simulation and modeling. – 2012. – P. 135–138.

22. Mauch M. pYIN: A fundamental frequency estimator using probabilistic threshold distributions / M. Mauch, S. Dixon // 2014 ieee international conference on acoustics, speech and signal processing (icassp). – IEEE, 2014. – P. 659–663.

23. Parselmouth – Praat in Python, the Pythonic way – Parselmouth 0.4.1 documentation [Електронний ресурс]. – Режим доступу: https://parselmouth.readthedocs.io/.

24. Sound: To Formant (burg)... [Електронний ресурс] // Phonetic Sciences, Amsterdam. – Режим доступу: https://www.fon.hum.uva.nl/praat/manual/Sound_To_Formant_burg___.html/.

25. Librosa [Електронний ресурс] // Librosa. – Режим доступу: https://librosa.org/.

26. Nayana P. K. Comparison of Text Independent Speaker Identification Systems using GMM and i-Vector Methods / P. K. Nayana, Dominic Mathew, Abraham Thomas // Procedia Computer Science. – 2017. – Vol. 115. – P. 47–54.

27. Клименко Н. С. Исследование эффективности бустинга в задаче текстонезависимой идентификации диктора / Н. С. Клименко, И. Г. Герасимов // Искусственный интеллект. – 2014. – №4(66). – С. 191–201.

28. Download Python [Електронний ресурс]. – Режим доступу: https://www.python.org/downloads/.

29. How to Install PIP on Windows ? [Електронний ресурс] // GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/how-to-install-pip-on-windows/.

30. Шевченко А. Д. Інформаційна технологія голосової ідентифікації за біометричними показниками людини / А. Д. Шевченко, Т. В. Ковалюк // Сучасні аспекти та перспективні напрямки розвитку науки: матеріали III Міжнародної студентської наукової конференції, м. Дніпро, 6 травня, 2022рік / ГО «Молодіжна наукова ліга». — Вінниця: ГО «Європейська наукова платформа», 2022. — С. 168–170.

31. ДСТУ 3008-2015: Державний стандарт України «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення». – Київ: ДП «УкрНПНЦ», 2016. – 25 с.

32. Методичні вказівки до виконання випускної кваліфікаційної роботи здобувача вищої освіти другого (магістерського) рівня вищої освіти за спеціальністю 121 “Інженерія програмного забезпечення” / Київський національний університет імені Тараса Шевченка; [уклад.: О. С. Бичков, В. Л. Шевченко, Д. С. Берестов, Е. В. Меркулова]. – К. : Видавничо-поліграфічний центр «Київський університет», 2020. – 48 с.

ДОДАТКИ

Додаток А Лістинг коду програми

Додаток А.1 Лістинг коду файлу main.py

```
import sys
import ctypes
from PyQt5 import QtWidgets, QtGui
from GUI.MainWindow import Ui_MainWindow

from FileModel import FileModel
from Speaker import *
from AudioPlayer import *
from SpeakerRecognizer import SpeakerRecognizer
from Config import Config

# для обробки помилок
def ShowErrorMsg(e):
    if hasattr(e, 'message'):
        mes = e.message
    else:
        mes = str(e)
    errorDialog = QtWidgets.QErrorMessage()
    errorDialog.setWindowTitle("Помилка")
    errorDialog.showMessage(mes)
    errorDialog.exec_()
    return

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.FileModel = FileModel()  # для перевірки коректності при відкритті файлу
        self.FileDialog = QtWidgets.QFileDialog()
        self.Speaker = SpeakerWave()
        self.Recognizer = SpeakerRecognizer()

        # події
        self.ui.bBrowse.clicked.connect(self.bBrowse_Clicked)
        self.ui.bPlay.clicked.connect(self.bPlay_Clicked)
        self.ui.bDenoise.clicked.connect(self.bDenoise_Clicked)
        self.ui.bRecognize.clicked.connect(self.bRecognize_Clicked)

    def SetLog(self, msg):
        if isinstance(msg, str):
            self.ui.teLog.moveCursor(QtGui.QTextCursor.End)
            self.ui.teLog.insertPlainText(msg + '\n')
        return
```

```
def bBrowse_Clicked(self):
    print("browse")
    try:
        self.fileName, _ = self.FileDialog.getOpenFileName(
            self,
            "Оберіть файл",
            "",
            "Wave Audio Files (*.wav)")
        self.FileDialog.close()
        if self.fileName:
            self.FileModel.setFileName(self.fileName)
            self.ui.leFileName.setText(self.fileName)
            self.ui.teLog.clear()

            self.Speaker = SetWaveFromFile(self.Speaker, self.fileName, reduceNoise=False)
            print("\nAudio data: \n")
            print(self.Speaker.GetGeneralInfo())
        return
    except Exception as e:
        ShowErrorMsg(e)
        return

def bPlay_Clicked(self):
    print("play")
    try:
        if self.Speaker is not None and len(self.Speaker.Wave) > 0:
            AudioPlayer.PlayAudio(self.Speaker.Wave, self.Speaker.SampleRate)
        else:
            ShowErrorMsg("Файл не обрано")
    except Exception as e:
        ShowErrorMsg(e)
    return

def bDenoise_Clicked(self):
    print("denoise")
    if self.Speaker is None or len(self.Speaker.Wave) <= 0:
        ShowErrorMsg("Файл не обрано")
        return
    try:
        if not self.Speaker.IsDenoised:
            self.Speaker = ReduceNoise(self.Speaker)
            self.SetLog("Виконано усунення шумів")
        else:
            self.SetLog("Для даного аудіо вже було виконано знешумлення")
    except Exception as e:
        ShowErrorMsg(e)
    return

def bRecognize_Clicked(self):
    if self.Speaker is None or len(self.Speaker.Wave) <= 0:
        ShowErrorMsg("Файл не обрано")
        return
    try:
        print("recognise")
        predictedClass = str(self.Recognizer.RecognizeSpeaker(self.Speaker))
        print("Predicted speaker: ", predictedClass)
```

```
            self.SetLog("Розпізнана особа: " + predictedClass)
        except Exception as e:
            ShowErrorMsg(e)
        return


myappid = u'mycompany.myproduct.subproduct.version'
ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(myappid)
app = QtWidgets.QApplication(sys.argv)
flag = True
try:
    Config.SetConfig()
except Exception as e:
    ShowErrorMsg("Виникла помилка під час зчитування файлу конфігурації: {e}".format(e=str(e)))
    flag = False

if flag:
    start_win = MainWindow()
    start_win.show()
    sys.exit(app.exec_())
```

Додаток А.2 Лістинг коду файлу AudioFeatures.py

```
import numpy as np
import librosa
import statistics
import parselmouth
from parselmouth.praat import call
from sklearn.cluster import KMeans
from Config import Config
import python_speech_features as mfcc
from sklearn import preprocessing

def GetWaveformAndSR(file_path, sr=Config.SR):
    samples, sample_rate = librosa.load(file_path, sr=sr)
    return samples, sample_rate

def NormalizeWave(wave):
    return wave*1.0/max(abs(wave))

def GetF0(wave, sr = Config.SR):
    f0, voiced_flag, voiced_probs = librosa.pyin(wave, sr=sr, fmin=librosa.note_to_hz('C2'),
fmax=librosa.note_to_hz('D5'), frame_length=256)
    times = librosa.times_like(f0)
    f0_filtered = np.copy(f0)
    f0_filtered[f0_filtered<=80] = None
    f0_filtered[f0_filtered>=450] = None
    f0_clean = f0_filtered[~np.isnan(f0_filtered)]
    return round(np.max(f0_clean),2), round(np.min(f0_clean),2), round(np.mean(f0_clean),2)

def GetMFCC(wave, sr=8000):
    mfccs = librosa.feature.mfcc(y=wave, sr=sr)
    mfccsProcessed = np.mean(mfccs.T, axis=0)
    return mfccsProcessed
```

```
def CalculateDelta(array):
    rows, cols = array.shape
    deltas = np.zeros((rows, 20))
    N = 2
    for i in range(rows):
        index = []
        j = 1
        while j <= N:
            if i - j < 0:
                first = 0
            else:
                first = i - j
            if i + j > rows - 1:
                second = rows - 1
            else:
                second = i + j
            index.append((second, first))
            j += 1
        deltas[i] = (array[index[0][0]] - array[index[0][1]] + (2 * (array[index[1][0]] - array[index[1][1]]))) / 10
    return deltas


def GetMFCCWithDelta(audio, rate):
    mfcc_feat = mfcc.mfcc(audio, rate, 0.025, 0.01, 20, appendEnergy=True)
    mfcc_feat = preprocessing.scale(mfcc_feat)
    delta = CalculateDelta(mfcc_feat)
    combined = np.hstack((mfcc_feat, delta))
    return combined

def GetSpectralCentroid(wave, sr=Config.SR):
    specCent = librosa.feature.spectral_centroid(y=wave, sr=sr)[0]
    return np.max(specCent), np.min(specCent), np.mean(specCent)

def MeasureFormants(sound, wave_file, f0min, f0max):
    sound = parselmouth.Sound(sound)
    pitch = sound.to_pitch()
    pointProcess = call(sound, "To PointProcess (periodic, cc)", f0min, f0max)
    formants = call(sound, "To Formant (burg)", 0.0, 6, 5500, 0.025, 50)
    numPoints = call(pointProcess, "Get number of points")

    f1_list = []
    f2_list = []

    for point in range(0, numPoints):
        point += 1
        t = call(pointProcess, "Get time from index", point)
        f1 = call(formants, "Get value at time", 1, t, 'Hertz', 'Linear')
        f2 = call(formants, "Get value at time", 2, t, 'Hertz', 'Linear')
        f1_list.append(f1)
        f2_list.append(f2)

    f1_list = [f1 for f1 in f1_list if str(f1) != 'nan']
    f2_list = [f2 for f2 in f2_list if str(f2) != 'nan']
    if (len(f1_list) == 0 or len(f2_list) == 0):
        raise Exception('Formants could not be calculated')
    f1_mean = statistics.mean(f1_list)
    f2_mean = statistics.mean(f2_list)
```

```
    f1_median = statistics.median(f1_list)
    f2_median = statistics.median(f2_list)
    return f1_mean, f2_mean, f1_median, f2_median, f1_list, f2_list

def GetFormantsData(wavepath):
    sound = parselmouth.Sound(wavepath)
    (f1_mean, f2_mean, f1_median, f2_median, f1_list, f2_list) = MeasureFormants(
        sound, wavepath, 60, 550)
    X = np.column_stack((f1_list, f2_list))
    kmeans = KMeans(n_clusters=4, random_state=0).fit(X)
    centroids = kmeans.cluster_centers_
    labels = kmeans.labels_
    return f1_mean, f1_median, f2_mean, f2_median, centroids


def DetectLeadingSilence(samples, chunk_size=1, thres_delim = 5, maxlen = 4*Config.SR):
    slen = len(samples)
    if slen<=maxlen:
        return 0
    thres = np.abs(max(samples) - np.mean(samples)) / thres_delim
    trim_frames = 0
    while np.abs(samples[trim_frames:trim_frames + chunk_size]) < thres and trim_frames<slen-maxlen:
        trim_frames += chunk_size
    return trim_frames
```

Додаток А.3 Лістинг коду файлу SpeakerRecognizer.py

```
import numpy as np
from NNModel import NNModel
from MixtureModel import MixtureModel
from Speaker import SpeakerWave, SetWaveFeatures
from AudioFeatures import GetMFCCWithDelta
from scipy.io.wavfile import read
from Config import Config

class SpeakerRecognizer:
    def __init__(self):
        self.Models = {
            "NN" :
                { "Model": NNModel(), "Weight": 0.2 },
            "GMM":
                { "Model": MixtureModel(), "Weight": 0.8 }
        }

    def RecognizeSpeaker(self, s, extend=False):
        if not isinstance(s, SpeakerWave):
            raise TypeError("На вхід необхідно подати об'єкт класу SpeakerWave")
        s = SetWaveFeatures(s)
        NNInput = s.GetFeatureVector()
        GMMSR, GMMAudio = read(s.WaveRoot)
        GMMInput = GetMFCCWithDelta(GMMAudio, GMMSR)
        NNProbs = self.Models["NN"]["Model"].GetClassProbabilities(NNInput)
        NNWinner = np.argmax(NNProbs)
        GMMProbs = self.Models["GMM"]["Model"].GetClassProbabilities(GMMInput)
        GMMWinner = np.argmax(GMMProbs)
        winner = -1
```

```
            if (NNWinner != GMMWinner):
                NNScore = NNProbs[NNWinner]*self.Models["NN"]["Weight"] / (NNProbs.max()-NNProbs.min()) - (self.Models["GMM"]["Weight"] / GMMProbs[NNWinner]) / (GMMProbs.max()-GMMProbs.min())
                GMMScore = GMMProbs[GMMWinner]*self.Models["GMM"]["Weight"] / (GMMProbs.max()-GMMProbs.min()) - (self.Models["NN"]["Weight"] / NNProbs[GMMWinner]) / (NNProbs.max()-NNProbs.min())
                winner = NNWinner if NNScore > GMMScore else GMMWinner
            else:
                winner = NNWinner
            if extend:
                return Config.CLASSES[winner], Config.CLASSES[NNWinner], Config.CLASSES[GMMWinner]
            else:
                return Config.CLASSES[winner]
```

Додаток А.4 Лістинг коду файлу NNModel.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
from tensorflow.keras import backend
from tensorflow.keras.models import load_model
from keras.utils.vis_utils import plot_model
import numpy as np
from Config import Config
from BaseRecognitionModel import BaseRecognitionModel

class NNModel(BaseRecognitionModel):
    def __init__(self):
        super(NNModel, self).__init__()
        self.Model = load_model(Config.NN_PATH) # завантажити файл моделі
        self.InputShape = Config.NN_INPUT_SHAPE
        plot_model(self.Model, to_file='model_plot.png', show_shapes=True, show_layer_names=False)

    def PredictClass(self, inputVector):
        pred = self.GetClassProbabilities(inputVector)
        pred = np.argmax(pred)
        winner = self.ClassNames[pred]
        return winner

    def GetClassProbabilities(self, inputVector):
        X = np.stack(inputVector).astype('float32')
        inputVector = np.reshape(X, self.InputShape)
        pred = self.Model.predict(inputVector)
        return np.transpose(pred).flatten()
```

Додаток А.5 Лістинг коду файлу MixtureModel.py

```
import _pickle as cPickle
import os
import numpy as np
from sklearn.mixture import GaussianMixture as GMM
from sklearn.preprocessing import minmax_scale
from Config import Config
from BaseRecognitionModel import BaseRecognitionModel
```

```
class MixtureModel(BaseRecognitionModel):
    def __init__(self):
        super(MixtureModel, self).__init__()
        self.GMMs = []
        gmm_paths = [os.path.join(Config.GMM_DIR_PATH,fname) for fname in
os.listdir(Config.GMM_DIR_PATH) if fname.endswith('.gmm')]
        gmm_paths = np.sort(gmm_paths)
        self.GMMs = [cPickle.load(open(fname,'rb')) for fname in gmm_paths]

    def PredictClass(self, inputVector):
        log_likelihood = self.GetClassProbabilities(inputVector)
        winner = np.argmax(log_likelihood)
        return Config.CLASSES[winner]

    def GetClassProbabilities(self, inputVector):
        log_likelihood = np.zeros(len(self.GMMs))
        for i in range(len(self.GMMs)):
            gmm = self.GMMs[i]
            scores = np.array(gmm.score(inputVector))
            log_likelihood[i] = scores.sum()
        log_likelihood = minmax_scale(log_likelihood)
        return log_likelihood / log_likelihood.sum()
```

Додаток А.6 Лістинг коду файлу Speaker.py

```
import NoiseReducer
import numpy as np
import matplotlib.pyplot as plt
from AudioFeatures import GetWaveformAndSR, NormalizeWave, GetF0, GetMFCC,
GetSpectralCentroid, GetFormantsData, DetectLeadingSilence

class SpeakerWave:
    def __init__(self, Id=-1, Gender="", Name="", Age=0, Accent="Unknown", Region="Unknown",
HasStationaryNoise=False, IsDenoised=False, WaveRoot="",
                 SampleRate=8000, Duration=0, Wave=[]):
        self.IsNormalized = False
        self.IsTrimmed = False
        self.FormantClusters = []  # чотири кластери формант
        self.MeanF1 = 0
        self.MedianF1 = 0
        self.MeanF2 = 0
        self.MedianF2 = 0
        self.MaxF0 = 0
        self.MinF0 = 0
        self.MeanF0 = 0
        self.SpectralCentroid = []
        self.Spectrogram = []
        self.Melpectrogram = []
        self.MFCC = []  # 20 признаков

        self.Conn = []
        self.SampleRate = SampleRate
        self.Duration = Duration
        self.Wave = Wave

        self.Id = Id
```

```
        self.Gender = Gender
        self.Name = Name
        self.Age = Age
        self.Accent = Accent
        self.Region = Region
        self.HasStationaryNoise = HasStationaryNoise
        self.IsDenoised = IsDenoised
        self.WaveRoot = WaveRoot

    def GetGeneralInfo(self):
        return ("Id = {Id}, Name = {Name}, Gender = {Gender}, Age = {Age}, Accent = {Accent}, \n" +
                "HasStationaryNoise = {HasStationaryNoise}, IsDenoised = {IsDenoised}, \n" +
                "WaveRoot = {WaveRoot} Duration = {Duration}\n").format(Id=self.Id, Name=self.Name,
Gender=self.Gender,
                                                        Age=self.Age, Accent=self.Accent,
                                                        HasStationaryNoise=self.HasStationaryNoise,
                                                        IsDenoised=self.IsDenoised,
                                                        WaveRoot=self.WaveRoot, Duration=self.Duration)

    def GetFeaturesInfo(self):
        return ("Id = {Id}, Name = {Name}, Duration = {Duration} \n" +
                "FormantClusters = {FormantClusters}, F0 = {F0}, \n" +
                "F1 mean = {MeanF1}, F1 median = {MedianF1}, F2 mean = {MeanF2}, F2 median =
{MedianF2}, \n" +
                "SpectralCentroid = {SpectralCentroid}, IsDenoised = {IsDenoised}, \n" +
                "MFCC = {MFCC}\n").format(Id=self.Id, Name=self.Name, Duration=self.Duration,
                                          FormantClusters=self.FormantClusters,
                                          F0=[self.MaxF0, self.MinF0, self.MeanF0],
                                          MeanF1=self.MeanF1, MedianF1=self.MedianF1, MeanF2=self.MeanF2,
                                          MedianF2=self.MedianF2,
                                          SpectralCentroid=self.SpectralCentroid,
                                          IsDenoised=self.IsDenoised, MFCC=self.MFCC)

    def PlotWaveform(self):
        time = np.arange(0, len(self.Wave)) * (1.0 / self.SampleRate)
        fig, ax = plt.subplots(1)
        ax.plot(time, self.Wave)

    def CopyToObject(self, speaker2):
        speaker2.IsNormalized = self.IsNormalized
        speaker2.IsTrimmed = self.IsTrimmed
        speaker2.FormantClusters = self.FormantClusters
        speaker2.MeanF1 = self.MeanF1
        speaker2.MedianF1 = self.MedianF1
        speaker2.MeanF2 = self.MeanF2
        speaker2.MedianF2 = self.MedianF2
        speaker2.MaxF0 = self.MaxF0
        speaker2.MinF0 = self.MinF0
        speaker2.MeanF0 = self.MeanF0
        speaker2.SpectralCentroid = self.SpectralCentroid
        speaker2.Spectrogram = self.Spectrogram
        speaker2.Melpectrogram = self.Melpectrogram
        speaker2.MFCC = self.MFCC

        speaker2.Conn = self.Conn
        speaker2.SampleRate = self.SampleRate
        speaker2.Duration = self.Duration
```

```
        speaker2.Wave = self.Wave

        speaker2.Id = self.Id
        speaker2.Gender = self.Gender
        speaker2.Name = self.Name
        speaker2.Age = self.Age
        speaker2.Accent = self.Accent
        speaker2.Region = self.Region
        speaker2.HasStationaryNoise = self.HasStationaryNoise
        speaker2.IsDenoised = self.IsDenoised
        speaker2.WaveRoot = self.WaveRoot
        return speaker2

    def GetFeatureVector(self):
        result = np.array([])
        result = np.append(result, self.FormantClusters.flatten())
        result = np.append(result, [self.MeanF1, self.MedianF1, self.MeanF2, self.MedianF2])
        result = np.append(result, [self.MaxF0, self.MinF0, self.MeanF0])
        result = np.append(result, self.SpectralCentroid)
        result = np.append(result, self.MFCC)
        return result


def SetWaveFromFile(s, filePath, normalize = True, reduceNoise = False):
    if not isinstance(s, SpeakerWave):
        raise TypeError("На вхід необхідно подати об'єкт класу SpeakerWave")
    s.WaveRoot = filePath
    s.Wave, s.SampleRate = GetWaveformAndSR(filePath)
    s.Duration = len(s.Wave) / float(s.SampleRate)
    if normalize:
        s.Wave = NormalizeWave(s.Wave)
    s.IsNormalized = normalize
    if reduceNoise:
        s = ReduceNoise(s)
    s.IsDenoised = reduceNoise
    return s

def ReduceNoise(s, stationary=False):
    if not isinstance(s, SpeakerWave):
        raise TypeError("На вхід необхідно подати об'єкт класу SpeakerWave")
    s.Wave = NoiseReducer.reduce_noise(y=s.Wave, sr=s.SampleRate, stationary=stationary, n_fft=1024)
    s.Wave = NormalizeWave(s.Wave)
    s.IsNormalized = True
    s.IsDenoised = True
    return s

def DeleteSilence(s):
    frames = len(s.Wave)
    maxframes = int(3 * s.SampleRate)
    trim = DetectLeadingSilence(s.Wave[::-1], maxlen=maxframes)
    new_samples = s.Wave[0:frames - trim]
    frames = len(new_samples)
    trim = DetectLeadingSilence(new_samples[::], maxlen=maxframes)
    s.Wave = new_samples[trim:frames]
    s.Duration = len(s.Wave) / float(s.SampleRate)
    s.Wave = NormalizeWave(s.Wave)
    return s
```

```
def SetWaveFeatures(s, deleteSilence=True):
    if s == None:
        return
    if deleteSilence:
        s=DeleteSilence(s)

    s.MaxF0, s.MinF0, s.MeanF0 = GetF0(s.Wave, s.SampleRate)
    print("F0 has been calculated")
    s.MeanF1, s.MedianF1, s.MeanF2, s.MedianF2, s.FormantClusters = GetFormantsData(s.WaveRoot)
    print("Clusters Data has been calculated")
    s.SpectralCentroid = np.round(np.array(GetSpectralCentroid(wave=s.Wave, sr=s.SampleRate)), 2)
    print("Spectral centroid has been calculated")
    s.MFCC = GetMFCC(s.Wave, s.SampleRate)
    print("MFCCs have been calculated")
    return s
```

Додаток Б Документи про участь у міжнародній конференції

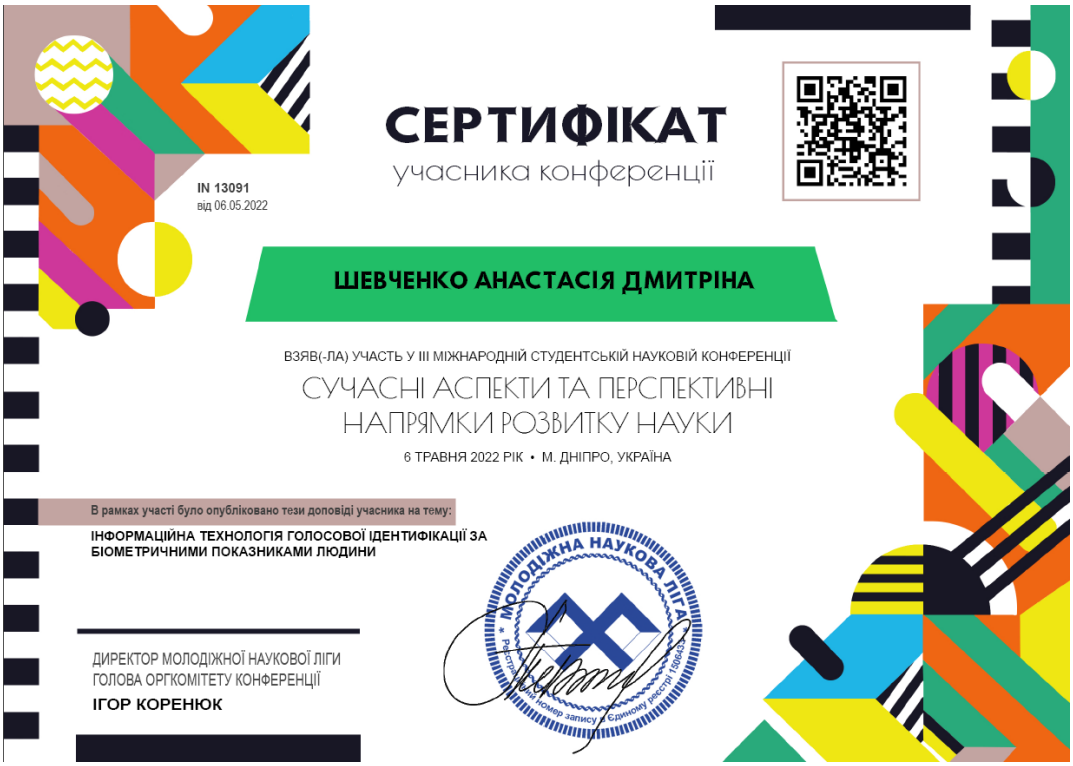

Рис. Б.1 Сертифікат учасника конференції

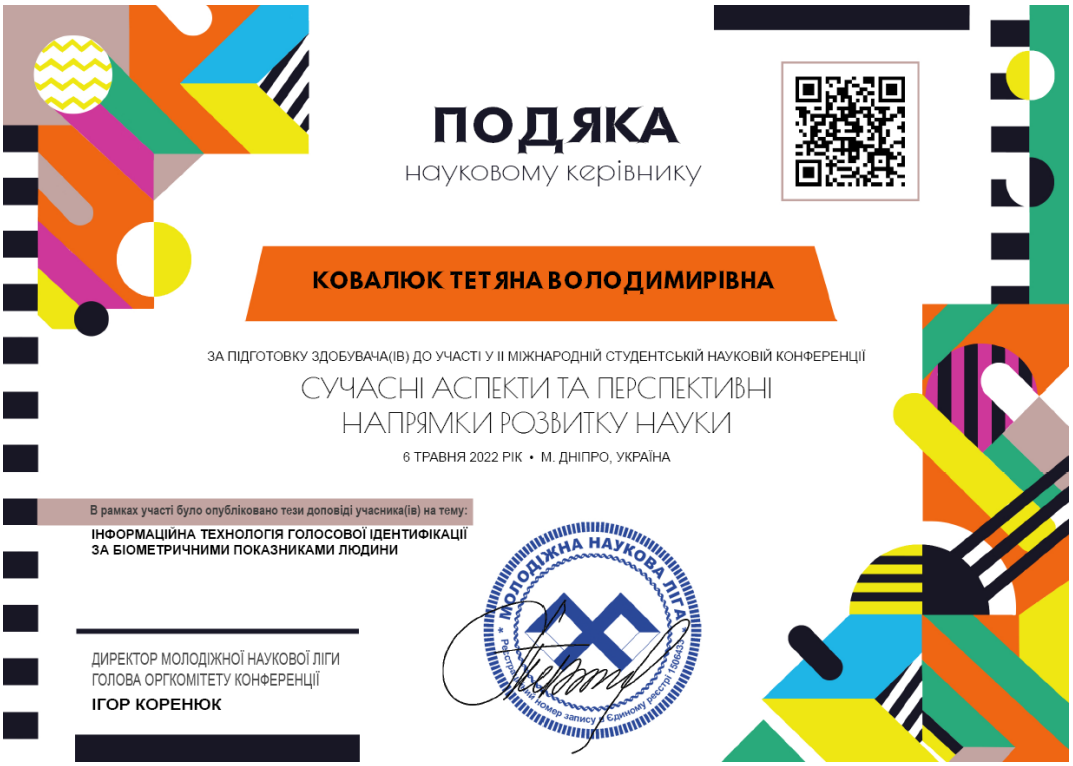

Рис. Б.2. Подяка науковому керівнику

Taras Shevchenko National University of Kyiv

A biometrics-based voice recognition information technology

Software Architecture Document (SAD)

CONTENT OWNER: Anastastasiia Shevchenko

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
1	v1.0	Sunday, April 24

1 Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- **Section "1.1 Document Management and Configuration Control Information"** explains revision history. This tells you if you're looking at the correct version of the SAD.
- **Section "1.2 Purpose and Scope of the SAD"** explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you're seeking is likely to be in this document.
- **Section "1.3 How the SAD Is Organized"** explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.
- **Section "1.4 Stakeholder Representation"** explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.
- **Section "1.5 Viewpoint Definitions"** explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 0, there is a corresponding view defined in **Section "3 Views"**. This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.
- **Section "1.6 How a View is Documented"** explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 Document Management and Configuration Control Information

Revision Number: 1

Revision Release Date: Sunday, April 24, 2022

Purpose of Revision: The first installment of the Software Architecture Document

Scope of Revision: none

1.2 Purpose and Scope of the SAD

This SAD specifies the software architecture for a biometrics-based voice recognition information technology**.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

1.2.1 Scope

This SAD describes a speaker recognition desktop application that is used for practical, testing or presentation purposes, when the classifiers have already been trained and tested. From the larger project of researching audio features and biometrics, building and training classifiers' models, making a comparative analysis of used methods, only a final stage of providing a functional speaker recognition application is covered.

1.2.2 Purpose

Biometrics are body measurements and calculations related to human characteristics. Biometric authentication is used in computer science as a form of identification and access control. It is also used to identify individuals in groups that are under surveillance. Biometric voice recognition is a type of biometric authentication that has advantages such as easy and understandable development and usage. Biometric voice recognition systems are used to solve a problem of speaker identification/recognition, there is a need for such system in criminalistics, surveillance, anti-terrorist monitoring, security etc.

The software which is described in this SAD is a result of a research of speaker recognition systems and methods, modelling and training two classifiers on a dataset of 43826 audio files from 108 speakers with different accents. The application is meant to use the developed feature extraction modules to extract data about speaker's voice from a WAVE file and pass them as an input to two classifiers – a feedforward neural network and a Gaussian mixture model, – aggregate their responses and get a final recognition result. One of the application's purposes is to provide a user-friendly way to test and present the results of this research project.

Application also maintains an option to change pre-trained classifiers, their input formats, recognizable classes (speaker names), so it can not only be used for the purpose of identifying an already chosen set of speakers from the training dataset, but to be more customizable for the end users.

1.3 How the SAD Is Organized

This SAD is organized into the following sections:

Section "1 Documentation Roadmap" provides information about this document and its intended audience. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.

Section "2 Architecture Background" explains why the architecture is what it is. It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.

Section "3 Views" specifies the software architecture. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 0), and is a representation of one or more structures present in the software (see Section 0).

Sections "4 Referenced Materials" and "5 Directory" provide reference information for the reader. Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD. A matrix of how serious a concern is to a particular stakeholder is presented in table 1.

Table 1: Stakeholders' concerns matrix

Concern / Stakeholder	UI	Programming language	Performance	Architecture	Development costs	Price
Customer	High	Low	High	Medium	High	High
End user	High	None	High	Low	Medium	High
Developer	Medium	High	Medium	High	High	Low
Tester	High	Low	High	Medium	Medium	None
Project manager	Medium	Medium	Low	Medium	High	High
Maintainer	High	Medium	High	Low	Low	Medium
Acquirer	Low	None	Medium	Low	High	High

1.5 Viewpoint Definitions

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 2: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoint(s) that apply to that class of stakeholder's concerns
Customer	Logical View, Use Case View
End user	Use Case View
Developer	Logical View, Use Case View
Tester	Use Case View
Project manager	Use Case View
Maintainer	Logical View, Use Case View

Stakeholder	Viewpoint(s) that apply to that class of stakeholder's concerns
Acquirer	Use Case View

1.5.1 Use Case Viewpoint Definition

Abstract

Use Case Viewpoint represents simplified basic scenarios that an End User performs while working with the application.

Stakeholders and Their Concerns Addressed

Customer. Customer is interested in the Use Case View to confirm that the application satisfies requirements provided for system architecture and functionality.

End user. End user can learn how their work routine with an application is supposed to be from the Use Case View. However, it does not provide the documentation for the UI.

Developer. Software developer requires a representation of foreseen scenarios to then provide a functional code that satisfies the functionality covered by scenarios.

Tester. Tester needs to know all the scenarios that end users are going to experience to diligently go through every one of them to check and report all possible outcomes.

Project manager. Project manager is interested in the Use Case View to confirm that the application satisfies requirements provided for system architecture and functionality. They also can refer to Use Case View to estimate the complexity of the system to plan out finances and human labor needed.

Maintainer. Maintainer needs to know all the scenarios to consult end users on how to use the application or modify it when needed.

Acquirer. Acquirer is interested in the Use Case View to confirm that the application satisfies requirements provided for system architecture and functionality and the price of the application is fitting for its level of complexity.

Elements, Relations, Properties, and Constraints

The Use Case View is presented by a UML diagram of use cases, which has the following element types:

- **Actor.** Behaviored classifier which specifies a role played by an external entity that interacts with the subject (e.g., by exchanging signals and data), a human user of the designed system, some other system or hardware using services of the subject. Represented by a stick man figure.
- **Use case.** A function that a system performs to achieve the user's goal. Depicted by an ellipse.
- **System.** The scope of a system, also known as a system boundary. Depicted by a half-transparent rectangle.

The elements are connected by following relations:

- **Association.** Actor and use case can be associated to indicate that the actor participates in that use case. Therefore, an association correspond to a sequence of actions between the actor and use case in achieving the use case. Depicted by a thin line.
- **Extend.** An extend relationship specifies how the behavior of the extension use case can be optionally inserted into the behavior defined for the base use case. Depicted by a dashed arrow with the label "<<extend>>" on it. The arrow points from extension use case to extendable use cases.

Language(s) to Model/Represent Conforming Views

For the purpose of representing a Use Case View a UML diagram of use cases was used.

1.5.2 Logical Viewpoint Definition

Abstract

Logical Viewpoint is a basis for understanding the structure and organization of the design of the system. The purpose of the Logical View is to specify the functional requirements of the system. In this SAD, Logical View is presented by a layered architecture of the application and class diagram.

Stakeholders and Their Concerns Addressed

Customer. Customer is interested in the Logical View when they have specific requirements to realization methods (for example, optimization methods for the sake of bettering performance). Otherwise, the specifics are not relevant to them.

Developer. Developer needs a Logical View to create the application according to requirements to structure, functionality, performance etc.

Maintainer. Maintainer needs the information on how the application is supposed to be organized, how and which classes are connected so when modifying is required they can alter the system without ruining its integrity and structure.

Elements, Relations, Properties, and Constraints

Software's layered architecture diagram includes the following elements:

- **Layer.** A separate functional part of the system that interacts in some sequential and hierarchical way with the other layers. Depicted by a rounded rectangle.
- **Component.** Represents a module, a subsystem or an application. Depicted by an ellipse.

Layers are connected with other layers and components are connected with other components by the following relations:

- **Dependency.** Depicts a dependency between two diagram blocks. Depicted by an arrow that points from dependent block to its dependency.

UML class diagram includes the following elements:

- **Class.** Class represents a set of objects that have the same structure, behavior, and relationships with objects of other classes. It can have attributes and methods. Attribute is a typed value that defines the properties and behavior of the object. Method is a function that can be applied to the objects of a given

class. Class is depicted by a 3-row table, where the first one is the name of a class, the second contains attributes and the third – methods.

- **Static class.** A special case of a class. A static class is similar to a class that is both abstract and sealed. The difference between a static class and a non-static class is that a static class cannot be instantiated or inherited and that all of the members of the class are static in nature. Depicted similarly to class, but with the label "<<static>>" at the top of the name row.
- **Abstract class.** A special case of a class that cannot be used to create objects (to access it, it must be inherited from another class). Depicted similarly to class, but with the label "<<abstract>>" at the top of the name row.

The elements are connected by following relations:

- **Dependency.** Means the relation between two or more classes in which a change in one may force changes in the other. However, it will always create a weaker relationship. Dependency indicates that one class depends on another. Depicted by a dashed arrow that points from dependent block to its dependency.
- **Association.** This kind of relationship represents static relationships between classes. Association is mostly verb or a verb phrase or noun or noun phrase. Depicted by a thin line with a verb label near the active class.
- **Aggregation.** A special type of association that models a whole- part relationship between aggregate and its parts. In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the college class will remain even if the student is not available. Depicted by a thin line with a diamond at the end of it, pointing at the container.
- **Composition.** A special type of aggregation which denotes strong ownership between two classes when one class is a part of another class. The contained class's instance lifecycle is dependent on the container class's instance lifecycle. Depicted by a thin line with a filled diamond at the end of it, pointing at the container.
- **Generalization.** A generalization helps to connect a subclass to its superclass. A sub-class is inherited from its superclass. Generalization relationship can't be used to model interface implementation. Class diagram allows inheriting from multiple superclasses. Depicted by a thin line with a triangle at the end of it, pointing at the superclass.

Language(s) to Model/Represent Conforming Views

For the purpose of representing a Logical View a UML diagram of classes and a simplified scheme of software's layered architecture were used.

1.6 How a View is Documented

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

Section 3.i: Name of view.

Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.

Section 3.i.2: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture

background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.

Section 3.i.3: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view.

2 Achitecture Background

2.1 Problem Background

2.1.1 System Overview

The system is an application that allows to recognize a speaker from a WAVE-file of their speech. It can be used for practical use in person authentication from pre-defined classes, as well as presenting and testing the results of researching NN architectures, GMMs and voice features extraction.

2.1.2 Goals and Context

The goal is to create a speaker recognition application that allows to recognize a speaker from pre-defined classes with high accuracy (above 90%). Since biometric authentication is used in such fields as criminalistics, surveillance, anti-terrorist monitoring, security, mistakes can be crucial.

One of the problems that all speaker recognition systems face is voice's variability depending of speaker's mood, health, microphone quality, environmental noises. Our goal is to minimize these effects to provide a practically useful software.

From the user's perspective, the application should be easily understandable and resistant to failures. In security the time of processing the data to provide or deny access is a significant aspect. User-friendly interface and fast work routines are necessary to create a competent speaker recognition software.

2.1.3 Significant Driving Requirements

2.1.3.1 Functional requirements

1. A singe instance of an application can be used by only one user. Each user has equal rights.
2. Opening a file dialog when user selects to browse a file and letting them to choose only WAVE-files.
3. Playing loaded audio when user choses to and showing an error message if no audio was loaded.
4. Reducing noise from loaded audio when user chooses to and showing an error message if no audio was loaded. Refusing to reduce noise from audio if it has already been done.
5. Starting a speaker recognition process when user chooses to and showing an error message if no audio was loaded. When a person is recognized, the application has to show the user recognized class (speaker's name)
6. The speaker recognition process has to consist from three stages: recognition via a feedforward NN, recognition via GMM and aggregating their results based on classifiers' outputs and pre-defined weights.
    a. Recognition via a feedforward NN has to include two stages: feature extraction and class prediction. Class prediction should provide an output of probabilities that an audio belongs to each of pre-defined classes.
    b. Recognition via GMM has to include two stages: feature extraction and class prediction. Class prediction has to provide an output of scores that each speaker's speech model calculated (a score describes a likeability of an audio corresponding to speaker's speech model).

c. Aggregation has to return the common recognized speaker if both classifiers have selected the same “winner” (gave the best score to a one speaker). Otherwise, while aggregating results, the scores of each other’s winners has to be taken into accounts, as well as classifiers’ weights and ranges between the best score and the worst score.

7. If an exception during any of the listed process has occurred, the application can’t shut down or crash. It has to inform user about it and continue processing user’s inputs.
8. User configurations have to be edited via an XML file. If any exception occurs during initiating configurations, the application must inform user about it and shut down.

2.1.3.2 Non-functional requirements

1. User-friendly graphic interface.
2. The recognition process should not exceed a time span of 1 minute. Other user actions like opening and listening to an audio file have to have a response time less than 5 seconds.

2.2 Solution Background

2.2.1 Architectural Approaches

The architecture follows a layered architecture approach. This approach works on principle of separation of concerns. Software design is divided into layer laid over one another. Each layer performs a dedicated responsibility.

Data and control flows from one layer to another crossing every layer in design. These layers also increase the degree of Abstraction in the design. As stability is proportional to abstraction to certain extent, it also improves stability of software to some limit. Isolation between layers keeps other layers immune from the modifications in one layer. While it doesn't offer much scalability, the developed application itself does not require it.

With relatively independent and isolated layers, if a part of a component needs to change, it’s less likely to affect the rest of the modules. In our case, most components that cover completely different goals (like extracting a set of features from an audio file or making estimates of how likely an array corresponds to established models) don’t need to know about each other’s specifications, and when they do, they are isolated by layer borders.

The application may need changes when a set of features that serve as an input to classifiers is no longer relevant, when the certain extraction methods have to be changes or the classifiers’ models have to be replaced to recognize completely different classes. In such cases, the changes are limited by updating separate components and editing a configuration file.

2.2.2 Requirements Coverage

The basic scenarios, functions and entities within the systems are provided in detail in Section 0, so it’s suggested to refer to it to fully comprehend which ones of requirements are satisfied and how. But in short, almost all of them have been covered by the system.

3 Views

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

3.1 Use Case View

3.1.1 View Description

Use Case View represents simplified basic scenarios that an End User performs while working with the application.

3.1.2 Architecture Background

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated. The "actors" are people or entities operating under defined roles within the system.

The speaker recognition system operates with only one actor – a user. It includes following use cases:

1. Configure settings – user edits and saves an XML file, that is processed by the system at the launch.
2. Invalid configuration file – extends Configure settings. Occurs when at the launch the XML edited by user is absent or invalid. An application shows a pop-up with the specification of the error and shuts down.
3. Open audio file – user clicks on a “Browse” button to open a file dialog and chooses a WAVE file. Can result to loading the chosen file to the system for further processing or closing the file dialog without picking anything (system remains without loaded audio file). If a file has been already opened before the use case, a new file replaces the old one.
4. Denoise audio – user clicks on a “Denoise” button to reduce noise from a chosen file.
5. Recognize speaker – user clicks on a “Recognize” button to identify the speaker by the recording of their speech. Internally a processes of feature extraction, NN and GMM classification and result aggregation are run. After successful recognition the user is given a class (speaker’s name) that the chosen audio belongs to.
6. Invalid or absent audio – extends Recognize speaker and Denoise audio. Occurs when the user tried to start one of the extended use cases before choosing a file beforehand. An application shows a pop-up with the specification of the error and continues working.

3.1.3 Context Diagram

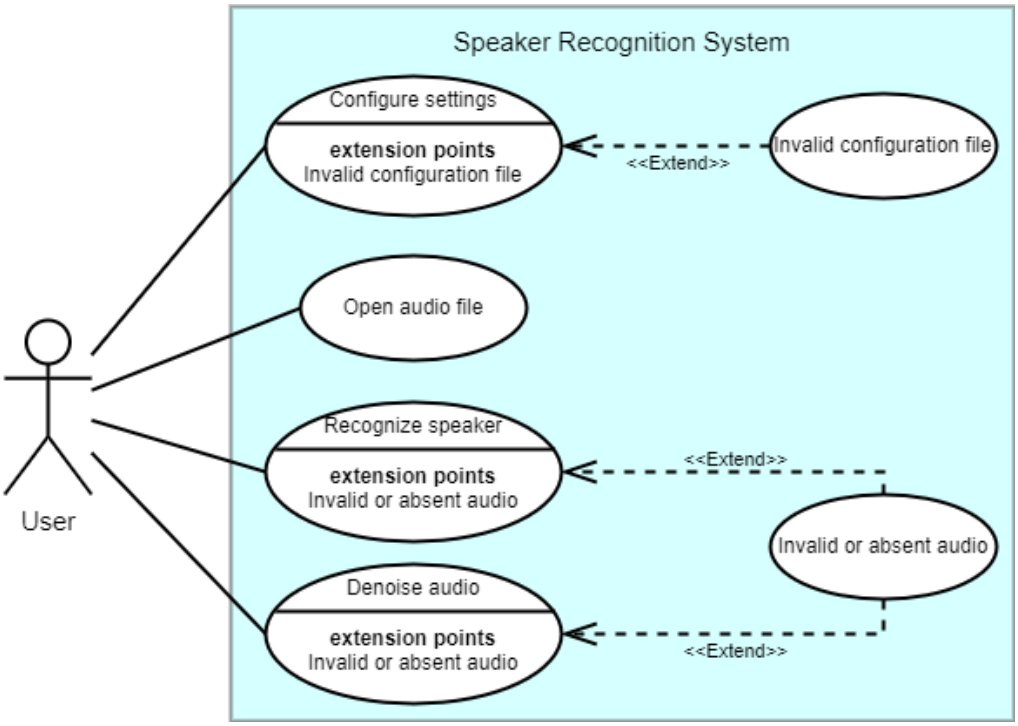

Figure 1: Use Case diagram

3.2 Logical View

3.2.1 View Description

Logical View is a basis for understanding the structure and organization of the design of the system. The purpose of the Logical View is to specify the functional requirements of the system. In this SAD, Logical View is presented by a layered architecture of the application and class diagram.

3.2.2 Architecture Background

The system is built with a layered architecture in mind. Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Each layer of the layered architecture pattern has a specific role and responsibility within the application. One of the powerful features of the layered architecture pattern is the separation of concerns among components. Components within a specific layer deal only with logic that pertains to that layer. For example, components in the presentation layer deal only with presentation logic, whereas components residing in the business layer deal only with business logic.

The system consists of three layers: Client, Presentation, Business (see *Figure 2*).

1. Client Layer consists of graphic use interface, created using PyQt5 tools. It's represented by a window interface.
2. Presentation Layer handles user's inputs and communicates to Client layer to present the outcome. It also communicates to Business Layer to run the processes the user has requested.
3. Business Layer is not aware of Client and Presentation layers, remaining separate. It implements the core logic of the speaker recognition system and provides output. The components included are Initiation

configurations, Wave-file processing, Audio denoising, Feature extraction and Speaker recognition. Within the layer, Speaker recognition component consists of two classifiers and an aggregator. Classifiers are not aware of the aggregator. Feature Extraction component is also not dependent on most components, while Speaker recognition component heavily depends on it.

The other part of Logical View is a class diagram (see *Figure 3*). The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

Classes MainWindow and AudioPlayer belong to the Presentation Layer. Every other class belongs to Business Layer.

MainWindow handles user's work routine within interface, communication to AudioPlayer and Business Layer's classes if needed. It has one instance of SpeakerWave and a one instance SpeakerRecognizer classes.

SpeakerWave represents a WAVE-file of person's speech and its properties. It may include speaker metadata (name, gender, region, accent, age), WAVE details (file path, sample rate, duration, the wave signal itself) and calculated voice features that will form a feature vector, an input to classifiers. It communicates to AudioFeatures class and a static NoiseReducer class to perform respective functions on itself.

SpeakerRecognizer contains one or more (two in our case) objects of an abstract BaseRecognitionModel class, that is a superclass to NNModel and MixtureModel. The abstraction is used because both NN and GMM can be generalized to a classifier that gives an output of a set length, the only difference is realization. Classifiers are not aware of SpeakerRecognizer. SpeakerRecognizer performs processes such as calling AudioFeatures class to calculate SpeakerWave's input vector to then provide it to classifiers. Classifiers' outputs are then weighted to produce the final result of recognition.

Config is a static class that almost every other Business Layer's class is dependent on. It provides access to user's configuration by processing an XML file. The configurations include a list of classes, input shapes of classifiers, paths to classifiers' models, default WAVE sample rate.

3.2.3 Context Diagram

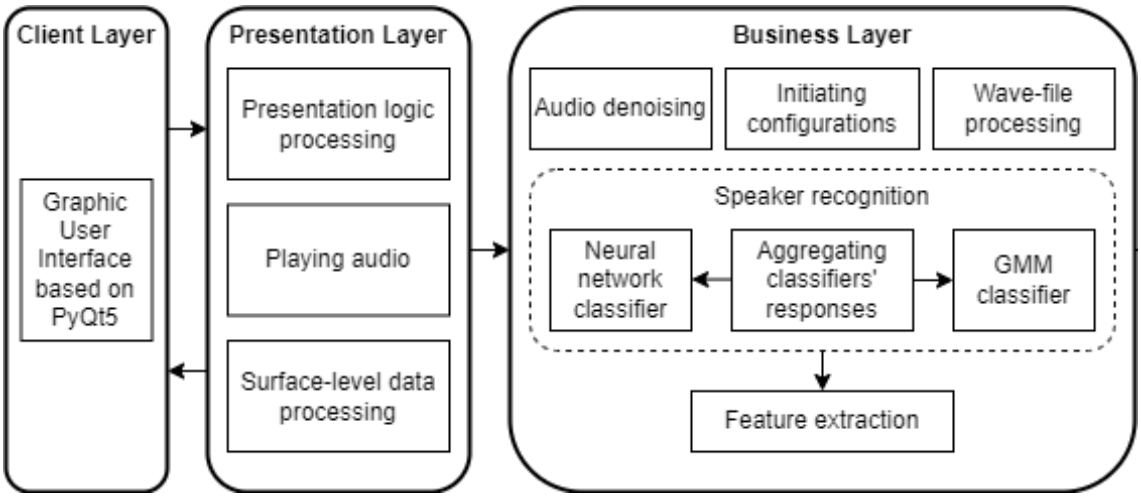

Figure 2: Software's layered architecture diagram

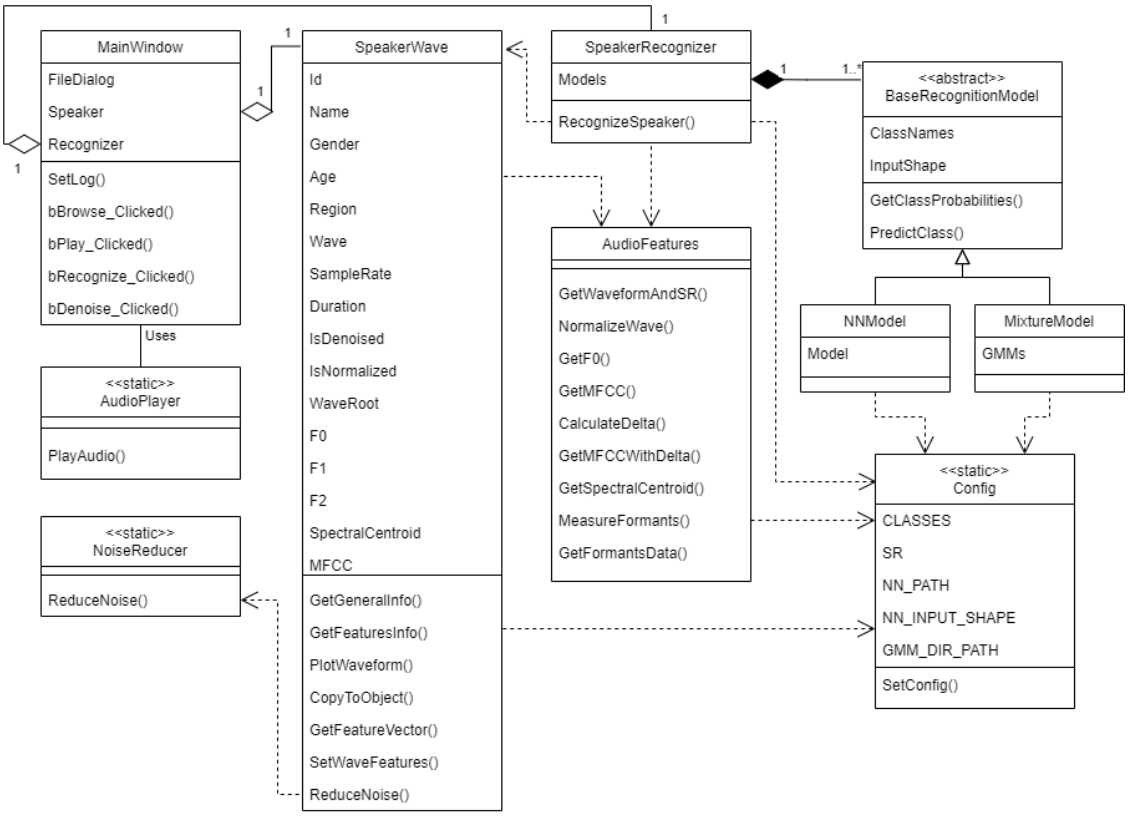

Figure 3: Class diagram

4 Referenced Materials

Barbacci 2003	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs)*, Third Edition (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 2003	Bass, Clements, Kazman, *Software Architecture in Practice*, second edition, Addison Wesley Longman, 2003.
Clements 2001	Clements, Kazman, Klein, *Evaluating Software Architectures: Methods and Case Studies,* Addison Wesley Longman, 2001.
Clements 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford, *Documenting Software Architectures: Views and Beyond*, Addison Wesley Longman, 2002.
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.
Reynolds 1995	Reynolds, Rose, Robust text-independent speaker identification using Gaussian mixture speaker models. IEEE transactions on speech and audio processing, 1995, 3.1: 72-83.
Sarangi 2020	Sarangi, Sahidullah, Saha, *Optimization of data-driven filterbank for automatic speaker verification*, Digital Signal Processing, 2020, 104: 102795.

5 Directory

5.1 Glossary

Term	Definition
feature extraction	In machine learning, pattern recognition, and image processing, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction [Sarangi 2020].
Gaussian mixture model	A probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians [Reynolds 1995].
neural network	A subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

5.2 Acronym List

COTS	Commercial-Off-The-Shelf
GMM	Gaussian Mixture Model
IEEE	Institute of Electrical and Electronics Engineers
NN	Neural Network
OS	Operating System
QAW	Quality Attribute Workshop
SAD	Software Architecture Document
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
UML	Unified Modeling Language