МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЧОВГАН ДМИТРО СЕРГІЙОВИЧ

Допускається до захисту:
завідувач кафедри інформаційних технологій,
д. т. н., доцент
_____ Т. В. Нескородєва
«_____»_____ 2022р.

ДОСЛІДЖЕННЯ ГОЛОСОВОГО АСИСТЕНТА В АВТОМОБІЛЯХ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Т. В. Нескородєва, завідувач кафедри інформаційних технологій,
д. т. н., доцент
науковий консультант:
Баркалов О.О.,
д-р техн. наук, професор

_____
(підпис)

Оцінка: _____ / _____ /_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця 2022

АНОТАЦІЯ

Човган Д.С. Дослідження голосового асистента в автомобілях. Спеціальність 122 «Комп'ютерні науки», Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

У кваліфікаційні роботі досліджено роботу голосового асистенту в цілому та роботу кожного модулю.

Досліджено основні підходи до розпізнавання голосу, виявлення сутностей та перетворення тексту на штучні голоси, що наявні сьогодні.

Виявлені головні проблеми низької швидкості моделей та технологій у сфері голосових асистентів.

Ключові слова: голосовий асистент, голосовий асистент у автомобілі, розпізнавання мовлення, підходи до розпізнавання мовлення.

80 с., 1 табл., 32 рис., 1 дод., 50 джерел.

Chovhan D. Cars voice assistant research. Specialty 122 "Computer science", Educational program "Computer data processing technologies (Data Science)". Vasyl Stus Donetsk National University, Vinnytsia, 2022.

In the graduated work I investigated how the voice assistant works in general, and how every module works.

The main approaches to voice recognition, entity detection, and text-to-speech conversion available today are shown.

The main problems of the low speed of development of new approaches and technologies in the field of voice assistants have been identified.

Keywords: voice assistant, voice assistant in the car, speech recognition, approaches to speech recognition.

80 pages, 1 tables, 32 pictures, 1 annex, 50 source

ЗМІСТ

ВСТУП

Метою даної роботи є дослідження роботи голосових асистентів, які використовуються в автомобілях.

Важливу роль у прийняті рішення щодо досліджень голосових асистентів, які використовуються у автомобілях, стала велика зацікавленість та популярність серед суспільства та інвесторів, сама ідея голосового асистенту.

Аналізуючи історію голосових асистентів, можна побачити, що фактично їх історія розпочалась ще у 1950-х роках, коли з'явився перший пристрій для розпізнавання голосу – Audrey (Одрі).

Не дивлячись на велику і відносно довгу історію, пік зростання зацікавленості голосових асистентів стався у 2010-х роках. Саме у 2010-х роках світ побачив щось схоже на те, чим зараз користується людство, а саме:

- Siri – компанія Apple;
- Google Assistant – компанія Google;
- Amazon Alexa – компанія Amazon.

Але усі ці асистенти не мають суттєвого впливу на життя людини. Вони здатні полегшити наше життя та можливо змінити погляд на методи взаємодії з деякими речами. Але значно більше користі приносять саме голосові помічники автомобілів, адже перш за все це безпека.

Перші голосові асистенти почали з'являтись у автомобілях ще у 2006 році. Дані голосові асистенти мали дуже обмежений функціонал та не використовували велику кількість технологій які доступні сьогодні.

У даній роботі, було прийнято рішення дослідити нові технології, які використовуються сьогодні та дослідити чи змінились технології з моменту виходу першого голосового асистенту в автомобілі. А саме:

- Дослідити структуру голосового асистенту;
- Дослідити патерни розпізнавання голосу, переваги того чи іншого підходу;
- Дослідити як змінилась архітектура голосових асистентів;

- Знайти певні недоліки у реалізації голосових асистентів.

Загалом за ці 16 років індустрія голосових асистентів зробила значні кроки і сьогодні асистент у автомобілі може дещо більше ніж просто переходити по меню за чітко визначеною командою. Але незважаючи на це, користуючись даним голосовим асистентом кожен день мною було знайдено безліч недоліків, які вказую на те, що дані голосові асистенти не повністю автономні, а те що автовиробники називають штучним інтелектом насправді не є ним. Адже іноді не спрацьовують навіть базові команди, які були сформульовані певним чином.

РОЗДІЛ 1
ОГЛЯД ТА АНАЛІЗ ТЕОРИТИЧНИХ ВІДОМОСТЕЙ

1.1 Мовлення

1.1.1 Загальні відомості про мовлення

Головною ідеєю будь-якого голосового асистенту є розуміння людини та виконання певного функціоналу на основі слів, які безпосередньо сказані людиною.

В основі будь-якого голосового асистенту лежить ідея спілкування людини з комп'ютером за допомогою мовлення. Адже саме мовлення є найбільш поширеним засобом спілкування. Саме за допомогою мовлення людина здатна виражати її почуття та думки.

У свою чергу за допомогою цієї ідеї було створено поняття – розпізнавання мовлення.

Розпізнавання мовлення (англ. speech recognition) – процес перетворення мовленнєвого сигналу в текстовий потік [1].

Іншими словами розпізнавання мовлення – це процес, який забезпечує комп'ютер можливістю розуміти людську мову.

Саме це поняття відноситься до галузі прикладної інформатики, інженерії та комп'ютерної лінгвістики.

Загалом мовлення є доволі складною формою даних, що має певні відмінності від інших форм. А саме:

- Мовний сигнал не містить інваріантних одиниць – це означає, що мовний сигнал, як і саме мовлення, може змінитись за умов, коли застосовується перетворення до мовлення;
- Мовлення може сприйматись та використовуватись нелінійним способом – дана відмінність відіграє дуже важливу роль, адже під час розмови, зазвичай, людина не дотримується певної часової

послідовності, натомість це можуть бути певні закінчення і тому подібне.

На перший погляд, до розпізнавання мовлення можна прирівняти розпізнавання тексту, адже текст – це форма даних, яка є письмовим відображенням мови.

Текст, як форма даних існує для збереження даних та можливості їх передачі на великі відстані. Так можна стверджувати лише на перший погляд, адже лише під час поверхового аналізу можна зрозуміти, що надрукований текст не володіє тими певними особливостями, що притаманні мовленню, а саме:

- Фонологічні варіації – різна вимова одних і тих самих слів;
- Індивідуальні відмінності – велика кількість синонімів у деяких мовах та різне значення слів в залежності від контексту;
- Фактори навколишнього середовища – фактори, що якимось чином впливають на розпізнавання голосу, наприклад, інші шуми у зоні досяжності мікрофону;
- Загальні проблеми – різна вимова в залежності від акценту людини.

Під час аналізу весь текст завантажується разом і це те, що вирішує головну проблему розпізнавання голосу. Система одразу може зрозуміти часові послідовності, а також ігнорується безліч інших аспектів таких як акцент, або інші дефекти вимови певних слів або звуків.

Незважаючи на всю складність обробки мовлення та певні їх особливості, саме розпізнавання мовлення з її особливостями надає велику кількість інформації, яку неможливо отримати з розпізнавання тексту. Зазвичай таку інформацію можна отримати за допомогою зміни тону та амплітуди звуку.

1.1.2 Обробка мовлення та види розпізнавання мовлення

Обробка мовлення – це дослідження мовних сигналів та методів їх обробки [2].

Загалом процес обробки мовлення складається з наступних аспектів:

- Введення мовного сигналу;

- Маніпулювання;
- Зберігання;
- Передача;
- Вивід мовного сигналу.

Найчастіше обробка мовлення застосовується для наступних цілей:

- ASR (Automatic Speech Recognition) [4];
- DSR (Digital Speech Recognition) [5];
- TTS synthesis (Text-to-Speech synthesis) [6];

Усі ці модулі є частиною SLD системи (Spoken Language Dialog System). А сама система є загальним поняттям яке дає змогу будь-якому голосовому асистенту розуміти людську мову та за допомогою додаткового функціоналу також і спілкуватись з людьми.

Процес обробки мовлення є доволі великим та обширним. Він складається з декількох етапів, які у свою чергу виконують певні поставлені задачі (рис. 1.1):

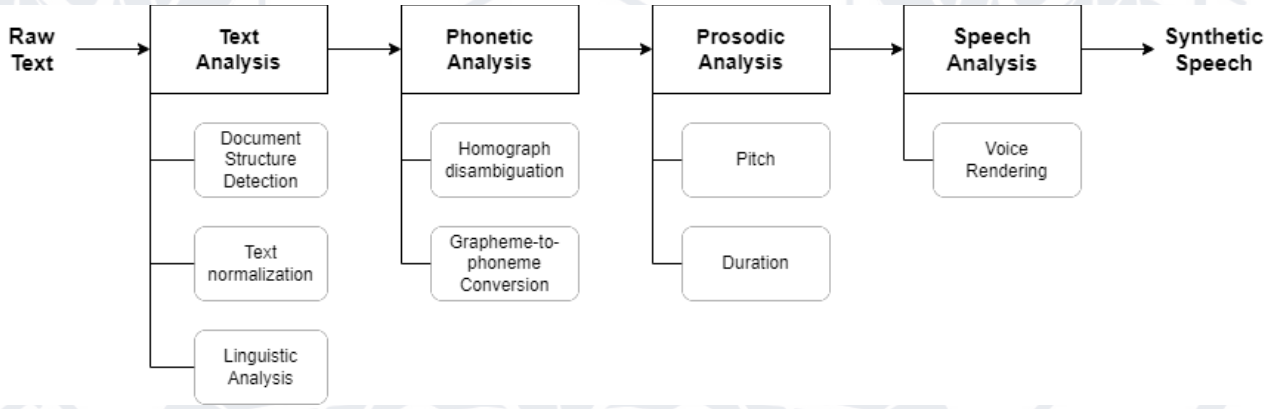

Рис. 1.1 Процес обробки мовлення

Raw Text (Необроблений текст) – звичайний текст, який подається на вхід для подальшого аналізу.

Text Analysis [7] (Аналіз тексту) – даний етап складається з декількох процесів:

- Document Structure Detection – структурний аналіз тексту для отримання повної структури усієї фрази;

- Text Normalization – процес приведення фрази до певного стандартизованого вигляду;
- Linguistic Analysis – обробка уже стандартизованого тексту методом виявлення ключових слів та його очищення від додаткових слів.

Phonetic Analysis [8] (Фонетичний аналіз) – процес аналізу розмовної мови, який складається з наступних процесів:

- Homograph disambiguation – усунення неоднозначних входжень;
- Grapheme-to-phoneme oncversion – процес генерування вимови слів на основі їх написання.

Prosodic Analysis [9] – аналіз фрази на основі моделей інтонації та наголосу в різних контекстах:

- Pitch (Висота звуку) – процес визначення висоти звуку, який відповідає за інтонацію та емоцію з якою було вимовлено те чи інше слово;
- Duration (Тривалість) – процес визначення тривалості з якою була вимовлена кожна фонема, за для покращення визначення емоційності.

Speech Analysis (Аналіз мовлення) – процес аналізу мовного сигналу:

- Voice rendering – передача голосу.

Також одним із найважливіших і початковим етапом у розпізнаванні голосу є діджиталізація та запис мовного сигналу. Даний етап оснований на процесі перетворення мовлення з аналогової форми у двійковий формат.

У якості передатчика найчастіше використовуються мікрофони, які безпосередньо зв’язані з платою.

Загалом етап діджиталізації можна поділити на три основних методи (рис. 1.2):

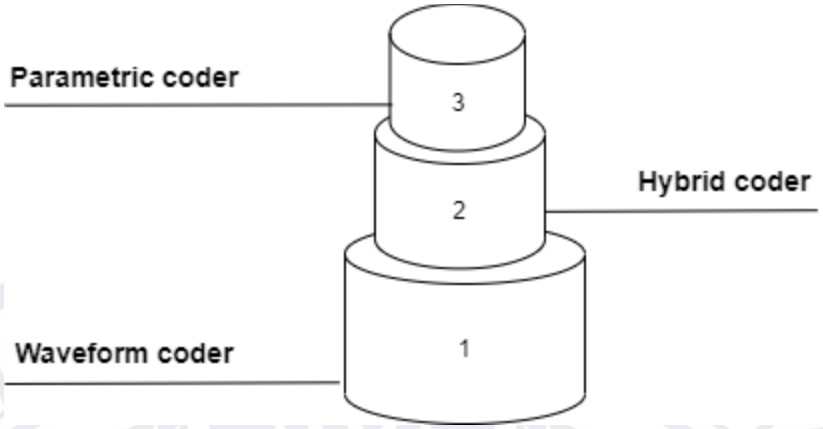

Рис. 1.2 Основні методи діджиталізації мовного сигналу

Waveform coder [10] – перетворює поточну голосову хвилю на серію репрезентативних нулів та одиниць.

Hybrid coder – використовує як голосові хвилі так і параметричні принципи у своїй роботі.

Parametric coder – виявляє певні характеристики, такі як висота тону та амплітуда.

1.2 Базова структура голосового асистенту

Не зважаючи на різноманітність функціоналу та різний підхід до ідеї голосових помічників, їх структура завжди має чотири основних модулі:

- Automatic Speech Recognition (ASR);
- Natural Language Understanding (NLU);
- Command execution;
- Text to speech (TTS).

Дані модулі сьогодні є загальноприйнятими, адже саме вони пояснюють послідовність дій та саму структуру проекту, що є важливим аспектом для створення повноцінного голосового асистента.

Кожний модуль є невід'ємною частиною голосового асистенту. За некоректного виконання одного з модулів, весь ланцюжок не буде виконано або буде виконано некоректно.

Так, мова користувача перетворюється на текст (ASR) після чого відправляється на обробку алгоритмів машинного навчання визначення наміру

користувача (NLU). В залежності від виявленого наміру активується потрібний клас у модулі виконання команд (Command Execution), який виконує запит користувача. Після завершення операції модуль виконання команд передає інформацію про статус виконання команди та її результат модулю синтезу мови (TTS), який у свою чергу сповіщає користувача.

1.2.1 Automatic speech recognition (ASR)

Першим модулем голосового помічника є Automatic Speech Recognition (ASR).

Automatic Speech Recognition (ASR) – це використання технологій машинного навчання або штучного інтелекту для перетворення людської мови в текст [3].

Загалом сьогодні існує два основних підходи для автоматичного розпізнавання голосу:

1. Традиційний гібридний підхід (hybrid approach);
2. Скрізний підхід до глибоко навчання [13] (end-to-end approach).

Враховуючи усі доступні технології традиційний гібридний підхід можна назвати застарілим. Якщо переглянути історію розвитку ASR технологій, то можна побачити, що даний підхід використовується упродовж 15 років. З огляду на стрімкість розвитку технологій, це є доволі довгим терміном. Але не зважаючи на це, велика кількість компаній все ще використовує даний підхід. Головною причиною цього є велика кількість доступних досліджень і літератури, які дозволяють побудувати надійну модель не дивлячись на те, що інші підходи забезпечують набагато вищий рейтинг розпізнавання голосу.

Розглянемо основні підходи для автоматичного розпізнавання голосу більш детально.

1.2.1.1 Гібридний підхід до автоматичного розпізнавання голосу

Гібридний підхід використовує традиційні GMM (гаусові моделі) та HMM (приховані марківські моделі) [11]. Але ці моделі мають один недолік – вони вимагають примусового вирівнювання даних.

Власне примусове вирівнювання це – це процес отримання текстової транскрипції звукового мовного сегмента та визначення того, де в часі зустрічаються певні слова у мовному сегменті.

Загалом гібридний підхід має наступну модель:

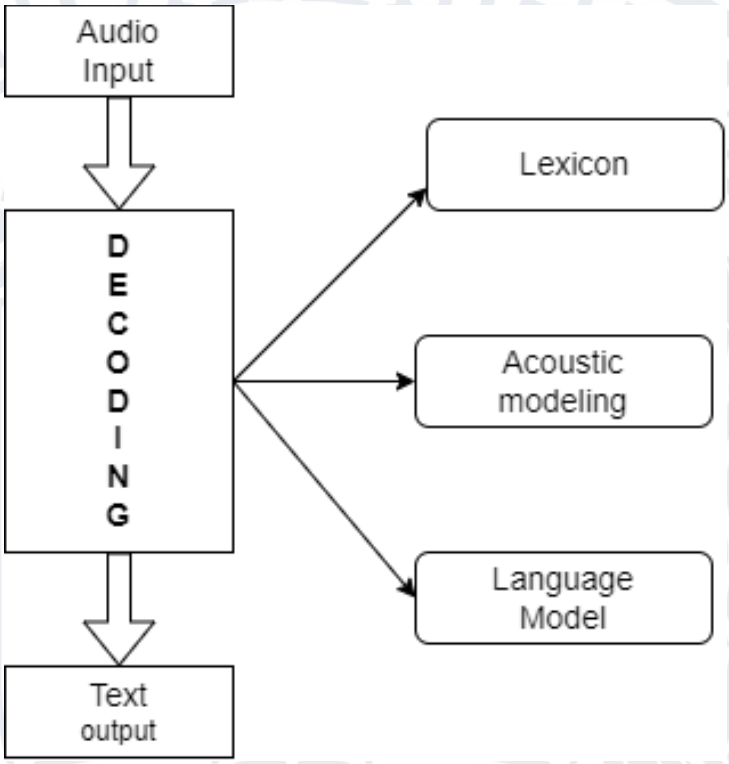

Рис. 1.3 Модель традиційного гібридного підходу розпізнавання голосу

На рисунку 1.3 можна побачити що даний підхід використовує лексичну, акустичну та мовну моделі для прогнозування транскрипції тої чи іншої фрази:

- Акустична модель [12] – моделює акустичні моделі мовлення. Завдання даної моделі полягає в тому, щоб передбачити, якій фонемі відповідає певна кожна форма хвилі. Дана модель зазвичай має справу із необробленими аудіо сигналами людської мови. Також до головних пунктів акустичної моделі можна віднести:
    - Зв’язок між акустичними знаннями і фонетикою розвивається саме у акустичній моделі;
    - Акустична модель відіграє важливу роль у продуктивності системи та відповідає за обчислювальне навантаження;

- Дана система використовує приховану модель Маркова (HMM), оскільки це один із найефективніших алгоритмів навчання та розпізнавання, який може бути використаний для акустичної моделі.

- Лексична модель [13] – дана модель описує як фонетично вимовляються слова. Зазвичай вам потрібен власний набір фонем для кожної мови, створений власноруч досвідченими фонетиками.
- Мовна модель [14] – моделює статистику мови. Дана модель виділяє певні гіпотези з набору фраз, які найімовірніше, будуть вимовлені. Головне завдання даної моделі полягає у тому, щоб передбачити альтернативи, які будуть використані та яку ймовірність вони будуть мати;
- Фінальним етапом даного підходу є декодинг. На даному етапі виходячи з усіх раніше отриманих даних, створюється транскрипція та подає на вихід певні гіпотези.

Не зважаючи на доволі значний недолік у вигляді примусового вирівнювання даних, даний підхід потребує значно меншу кількість даних, для тренування.

Загалом може вистачити лише 3000 годин даних для того, щоб система почала працювати достатньо добре, хоча і швидкість обробки буде не дуже висока.

За даним підходом існує доволі велика кількість різної інформації, але не дивлячись на це, як було уже сказано вище, він має один великий недолік, а саме – низька точність розпізнавання. За для уникнення цього недоліку все частіше у проектах починають використовувати інший підхід – розпізнавання наскрізь (End-to-End deep learning approaching).

Даний підхід в своїй основі використовує приховану модель Маркова (HMM).

Прихована модель Маркова – це статистична модель Маркова, в якій механізм моделювання вважається неспостережуваним (тобто прихованим)

процесом Маркова [15]. А сама прихована модель Маркова заснована на розширенні ланцюга Маркова.

Ланцюг Маркова – це модель, яка повідомляє нам щось про ймовірні послідовності випадкових величин, станів, кожна з яких, може приймати значення з деякого набору [16]. Ці набори можуть бути словами, тегами або символами, що представляють будь-що, наприклад, погоду. Ланцюг Маркова робить дуже сильне припущення, що якщо ми хочемо передбачити майбутнє в послідовності, все, що має значення, це поточний стан.

Прихована модель Маркова (HMM) дозволяє говорити як про спостережувані події (наприклад, слова, які ми бачимо у вхідних даних), так і про приховані події (наприклад, теги частини мови), які ми вважаємо причинними факторами в нашій ймовірній моделі.

Дана модель володіє доволі великою кількістю компонентів, а саме:

- T – довжина послідовності спостереження;
- N – кількість станів;
- M – кількість символів спостереження;
- Q – різні стани процесу Маркова;
- A – ймовірності переходу стану;
- B – матриця ймовірності спостереження;
- ? – розподіли початкових станів;
- O – послідовність спостереження.

У даному випадку спостереження (O – observations) – це наші дані для тренування, а кількість прихованих даних є нашим гіпер параметром (hyperparametr). Ці дані повинні контролюватись розробником.

Топологія HMM визначає набір дозволених переходів між станами. Загалом у прихованій моделі Маркова існує декілька топологій, а саме:

- Ергодичний повний зв’язок (ergodic-fully connected) (рис. 1.4) – кожен стан має перехід до іншого стану.

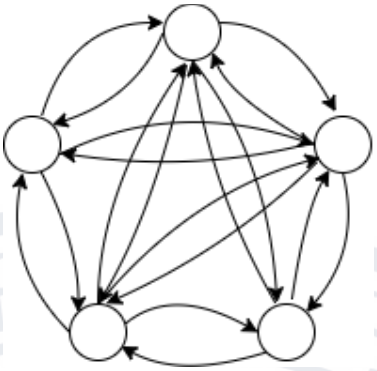

Рис. 1.4 Ергодичний повний зв’язок

- Перехід з ліва на право (left-to-right-transition) (рис. 1.5) – тільки в стані з більш високим індексом, ніж індекс поточного стану. Дана топологія завжди нав’язує часовий порядок, але сама ця топологія найчастіше використовується у випадках розпізнавання мови. Основними особливостями даного підходу є:
    - Мовлення ніколи не повертається назад;
    - Вимова відрізняється на кожному із етапів

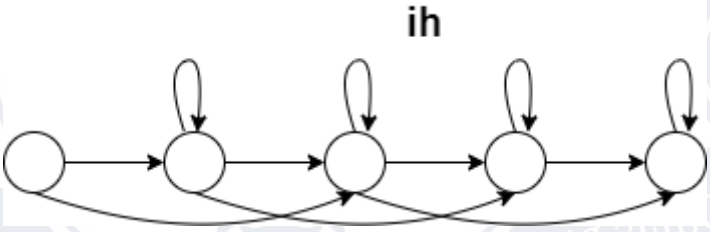

Рис. 1.5 Перехід з ліва на право

Під час використання прихованої моделі Маркова у сфері розпізнавання голосу, найчастіше усю фразу подають як послідовність спостережень. Та вже в залежності від цього використовують HMM для моделювання умовної мовної одиниці (phone, word) і об’єднання цих одиниць у більші одиниці (рис. 1.5, 1.6):

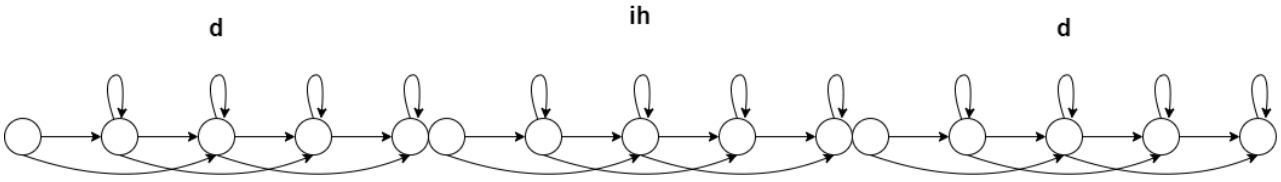

Рис. 1.6 Word model

Також існують різні типи прихованої моделі Маркова:

- Whole word HMM;
- Context-independent Phoneme HMM;
- Context-dependent Triphone HMM.

1.2.1.2 Скрізний підхід до глибоко навчання

Наскрізне навчання (End-to-end) можна віднести до складних систем навчання, які відображені за допомогою однієї моделі (зокрема глибокою нейронною мережею) [17]. Вона представляє повну цільову систему, минаючи проміжні рівні, які зазвичай присутні в традиційних конструкціях конвеєрів.

На перший погляд, даний підхід є доволі складним та не дуже затребуваним у порівнянні з традиційним гібридним підходом. Проте головною проблемою традиційного гібридного підходу є те, що там, кожен модуль потребує додаткової оптимізації за певними особливими критеріями.

В свою чергу ідея підходу наскрізного навчання, полягає в заміні доволі громіздкого ланцюгу традиційного гібридного підходу на доволі компактну систему (рис. 1.7):

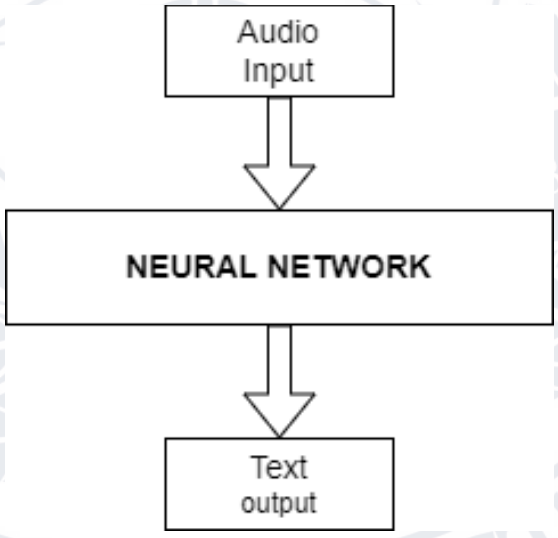

Рис. 1.7 Скрізний підхід до глибокого навчання

У якості акустичної моделі, даний підхід використовує модель нейронної мережі.

Іншою, доволі вагомою перевагою даного підходу є те, що даний підхід дозволяє спроєктувати модель, яка доволі добре справляється з поставленим

завданням без глибоких знань про саму проблему, незважаючи на складність цієї проблеми.

Беручи до уваги, що у даній моделі відсутня будь-яка попередня обробка даних, вона потребує досить великої кількості даних для навчання. Найчастіше, щоб змусити дану систему працювати на високому рівні, потрібно використати датасет щонайменше на 100000 годин. Але незважаючи на це, результат який можна отримати в кінці, буде значно кращий аніж при традиційному гібридному підході.

Власне незважаючи на значні показники даної моделі, поки що вона використовується не дуже часто. І причиною цьому є доволі мала кількість інформації порівняно з традиційною моделлю.

За допомогою End-to-End моделі можна напряму перетворювати послідовність вхідних акустичних даних на послідовність слів. Попередньо дані не потрібно вирівнювати як це було у традиційній моделі.

В теорії End-to-End не потрібна лексична і мовна моделі. Але за умови використання мовної моделі можна отримати більший точні результати розпізнавання.

У End-to-End найчастіше використовуються такі архітектури як:

- Listen Attend and Spell (LAS);
- Recurrent Neural Network Transducers (RNNT).

1.2.1.3 Акустична модель

Однією із основоположних моделей є саме акустична модель, власне дана модель відноситься до процесу статичних представлень для комп'ютерного вектора ознак.

Дана модель, тим чи іншим чином, представлена як і у традиційному гібридному так і у скрізному підході до машинного навчання.

Загалом незважаючи на це, існує доволі велика кількість різних моделей, які можуть бути використані у якості основоположної моделі у процесі акустичного моделювання, а саме:

- Прихована модель Маркова (HMM);
- Сегментальна модель (Segmental model);
- Супер-сегментальна модель (Super-segmental mode);
- Нейроні мережі (Neural networks);
- Модель максимальної ентропії (Maximal entropy model);
- Модель умовних випадкових полів (conditional random fields).

Але найчастіше використовуються дві моделі, а саме прихована модель Маркова та Нейроні мережі.

Власне прихована модель Маркова використовуєть у традиційному гібридному підході, у свою чергу скрізний підхід використовує модель нейронних мереж, так як саме у цьому і полягає вся ідея даного підходу.

1.2.1.4 Мовна модель

Мовна модель включає у себе концептуальні недоліки мови, доступної для передбачення ймовірностей. Після послідовності слів модель пропонує ймовірність кожного слова.

Дана модель вирішує наступні проблеми:

- Ідентифікує слова та фрази із різним тоном;
- Допомагає визначити ймовірність кожного слова зважаючи на акустику;
- Допомагає розв’язувати неоднозначності у парі з іншими модулями.

Загалом існує два типи мовної моделі:

- Природня мовна модель (Natural Language model)
- Статистична мовна модель (Statistic language model)

1.2.2 Natural-language understanding (NLU)

Natural-language understanding (NLU) – це галузь штучного інтелекту, що використовує комп'ютерне програмне забезпечення для розуміння вхідних даних як пропозицій з використанням тексту або мови [18].

В загальному NLU є однією із підмножин обробки природної мови (NLP).

NLP (Native-language processing) – це галузь штучного інтелекту, яка дозволяє комп'ютеру розуміти людську мову [19, 20].

NLP аналізує та намагається зрозуміти текст незалежно від того чи це розмовна мова, чи написаний текст. У свою чергу NLU дає змогу здійснювати діалог з комп'ютером. Тобто саме NLU дозволяє комп'ютеру розуміти людську мову без використання людиною формальної комп'ютерної мови [21].

Так, як NLU є частиною NLP та відповідно до цього використовує функціонал NLP, коротко розглянемо технологію Native-language processing.

NLP використовує штучний інтелект, для того, щоб приймати реальні дані, обробляти їх та відображати таким чином, щоб комп'ютер міг їх розуміти.

Усього існує два основні етапи обробки мови [22]:

- Попередня обробка даних (data preprocessing) (рис. 1.8) – головним завданням даного етапу є попередня обробка [23]. Саме даний етап займається очищенням тексту та виділенням певних особливостей, які у свою чергу потрапляють до самого алгоритму.
- Розробка алгоритмів (algorithm development) – даний етап передбачає розробку алгоритму для обробки попередньо очищених даних.

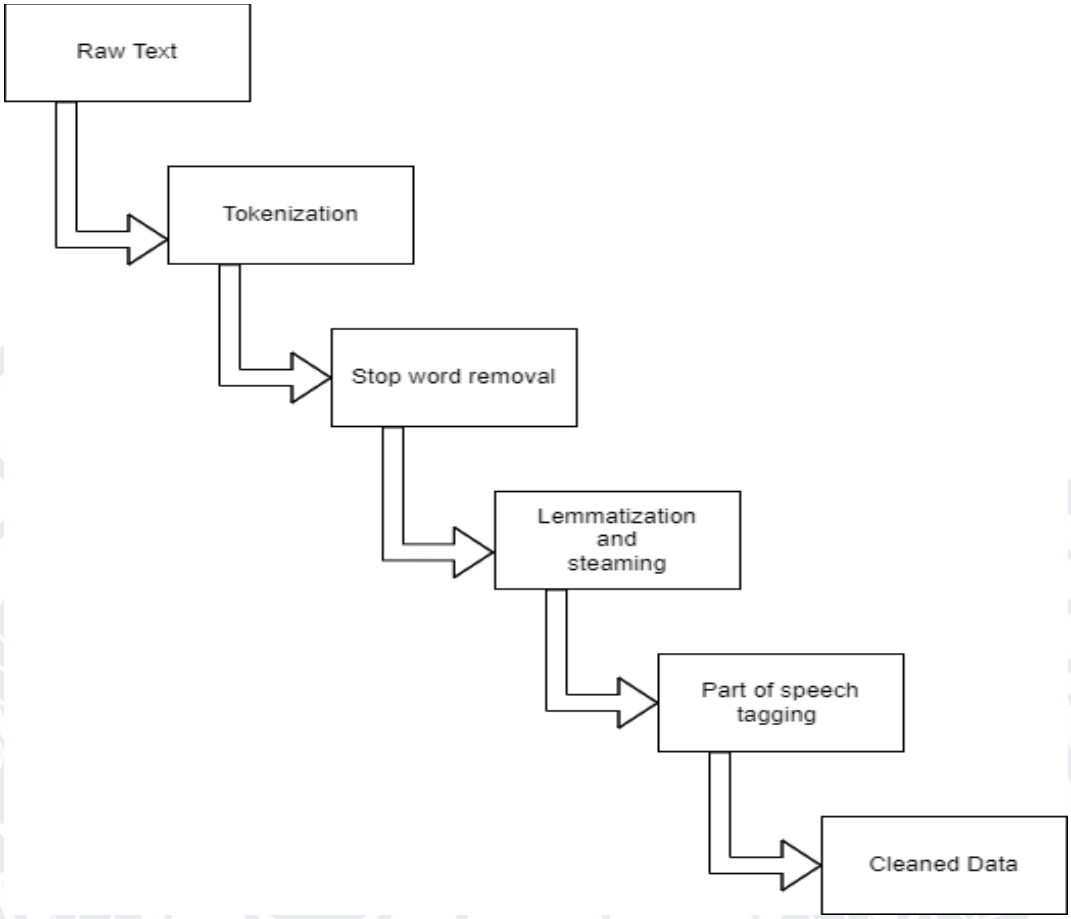

Рис. 1.8 Способи попередньої обробки даних

Загалом існує декілька способів попередньої обробки даних:

- Токенізація (Tokenization) – процес розбиття тексту на слова. Загалом токенізація – це процес заміни конфіденційних даних унікальними [24].
- Видалення стоп слів (Stop word removal) – процес видалення загальних слів із тексту та залишення ключових [25];
- Лематизація (Lemmatization and stemming) – процес групування різних форм одного і того ж слова воєдино [26];
- Тегування частини мови (Part-of-speech tagging) – процес позначення слів на основі частини мови (іменник, дієслово, прикметник тощо) [27].

Розглянемо деякі способи попередньої обробки даних більш детально.

Спосіб видалення стоп слів полягає у тому, що система повинна видалити усі загальні слова, тобто загальновживані слова (рис. 1.9). Зазвичай сама система пропонує лист слів, які потрібно ігнорувати під час індексації записів та під час отриманні результатів пошукового запису.

Рис. 1.9 Процес видалення стоп слів

Сьогодні більшість алгоритмів, або так названих «engine», запрограмовані на видалення деяких слів з будь-якого індексованого запису. Власне список слів, що не підлягають додаванню, називається стоп-листом.

Стоп-лист загалом не є актуальним під час пошуку, оскільки дані слова часто зустрічаються у реченнях певної мови, для якої і була налаштована індексація.

Також, з метою економії простору та часу, даний лист видаляється під час індексації та ігноруються під час пошуку. Хоча, не дивлячись на це, деякі «двигуни» дозволяють використовувати стоп-слова в пошуку, методом підстановки знаку «+» перед кожним стоп словом.

Лематизація (рис. 1.10), є важливим аспектом у процесі розуміння природної мови (NLU) та процесі обробки природної мови (NLP). Також лематизація є доволі важливим процесом у штучному інтелекті та аналітиці великої кількості даних.

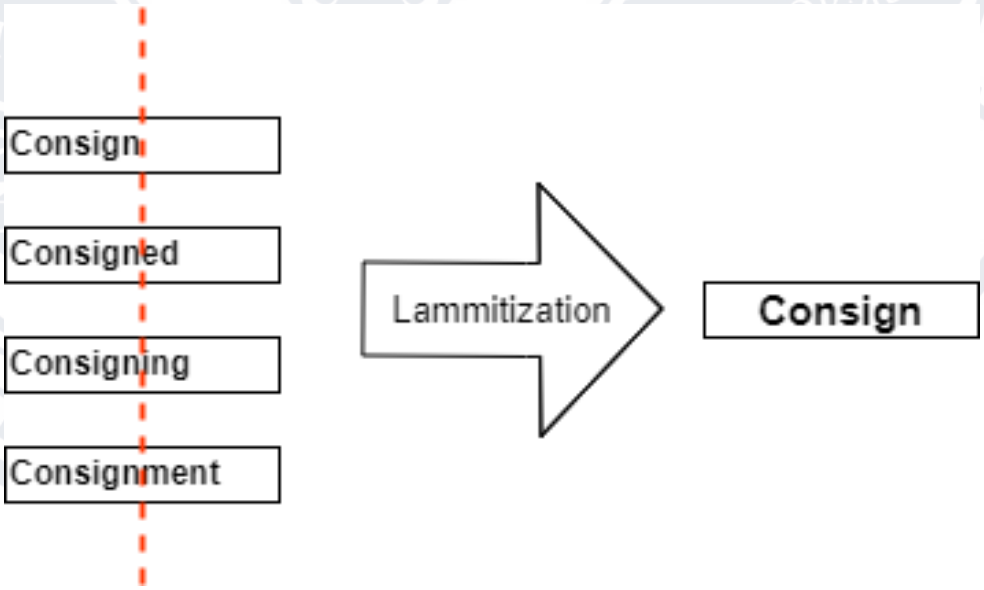

Рис. 1.10 Приклад лематизації

Велика кількість алгоритмів використовує правила лінгвістичної морфології у парі з словником певної мови, для групування слів, що використовуються у мові. В свою чергу глибоке навчання використовується для аналізу та розуміння процесу групування. З огляду на це, можна сказати що процес лематизації використовується під час згадування будь-якої флективної форми.

Під час пошуку, лематизація дозволяє користувачу, отримати будь-яку версію базового слова та отримати відповідний результат. Також, дивлячись на те, що алгоритми, які використовуються у пошукових системах, використовують даний процес лематизації, користувач також може запитати будь-яку флективну форму слова та отримати потрібний йому результат, відповідно.

У якості прикладу візьмемо слово «картки». Якщо користувач надасть слово «картки», що використовується у множині, система з легкістю визначить, що це слово вживається саме у множині і видасть потрібний результат.

За процесом попередньої обробки слідує процес розробки алгоритму, який і буде використовувати попередньо оброблені дані.

Загалом існує доволі велика кількість алгоритмів обробки природньої мови, але зазвичай використовується лише два, які найчастіше і є основними:

- Система заснована на правилах;
- Система на основі машинного навчання.

Система заснована на правилах, використовує ретельно оброблені лінгвістичні правила [28].

Даний підхід вважається доволі старим, адже був заснований ще на ранніх етапах розвитку такого напрямку, як обробка природньої мови. Але незважаючи на її вік, дана система є доволі популярною і сьогодні, та нерідко використовується у сучасних проектах.

У свою чергу, система на основі машинного навчання, є доволі новою і надає набагато більший функціонал [28].

Дана система працює на основі статистичних методів. Такі системи, навчаються виконувати задачі на основі вибірок даних для навчання, і

відповідно до цього, коригують свої методи в залежності від кількості даних, які були опрацьовані. Тобто, у результаті отримуємо доволі примітивну залежність, чим більше даних було опрацьовано, тим кращий результат буде отримано.

Алгоритми обробко природньої мови використовують комбінації машинного навчання, глибокого навчання та нейронних мереж, що дозволяє їм відточувати свої власні правила методом повторної обробки та навчання.

Загалом існує два основних підходи обробки природньої мови:

- Синтаксичний аналіз;
- Семантичний аналіз.

Синтаксис – це порядок слів у реченні, яке відповідає за зміст речення [29]. NLP використовує синтаксичний аналіз для оцінки значення усього речення основуючись на правилах граматики.

Синтаксичний підхід включає у себе [30]:

- Парсинг – граматичний розбір речення;
- Сегментація слів – взяття рядка тексту та отримання з нього самих слів;
- Розподіл речень – розділ усього тексту на речення;
- Морфологічна сегментація – розподіл слів на морфеми.

У свою чергу семантичний аналіз також передбачає аналіз значення слів. Це використовується для того, щоб зрозуміти значення слова, та сенс усього речення.

Загалом існують наступні прийоми семантичного аналізу [31]:

- Розпізнавання сенсу слова – допомагає отримати значення слова у певному контексті;
- Розпізнавання іменованих сутностей – слугує для того, щоб отримати слова, які можна розділити на групи;
- Генерація природньої мови – використовує базу даних для визначення семантики слів та створення нового тексту.

Не дивлячись на велике різноманіття різних підходів, будь-які сучасні підходи до обробки природньої мови базуються на глибокому навчанні. Так як, саме глибок навчання може забезпечити дослідження та використання шаблонів

для покращення розпізнавання. Але хоч і глибоке навчання пропонує перевагу, воно має доволі вагомий недолік, а саме те, що потребує досить велику вибірку даних для навчання та виділення відповідних кореляцій.

Попередні методи обробки природньої мови були більш простими, оскільки використовували підхід заснований на правилах.

Суть підходу заснованого на правилах, полягала у тому, що алгоритмам машинного навчання напряму вказувалось, які слова потрібно шукати у тексті і відповідно даний підхід давав конкретні відповіді, якщо потрібні слова або фрази були знайдені.

Але водночас, попередні методи є менш продуктивними. Нові підходи, є гнучкішими та мають більш інтуїтивний підхід. Фактично, у віддаленій перспективі, можна сказати, що саме так будь-яка дитина вивчає мову.

Підводячи підсумок, головним завдання NLP як і NLU є розуміння людської мови, але за однією відмінностю. NLU повинне спілкуватись з непідготовленими людьми та розуміти їх наміри. І власне це і є головною перевагою NLU.

Ця галузь виходить за рамки звичайного розуміння і тлумачення сенсу слів. Дана технологія запрограмована таким чином, щоб розуміти сенс сказаної фрази, незважаючи на численні помилки у вимові. Наприклад: помилки у словах, неправильний наголос, тощо.

NLU аналізує дані для того, щоб визначити їх значення. Для цього використовуються алгоритми перетворення людської мови в структуровану онтологію. Де штучний інтелект виділяє час, намір, настрій та місце розташування.

Структурована онтологія – це модель даних, яка складається з семантичних та прагматичних визначень.

Загалом NLU має дві фундаментальні концепції:

- Розпізнавання намірів (Intents);
- Розпізнавання об’єктів (Entities).

Розпізнавання намірів — це процес визначення настрою користувача у тексті та визначення його мети [32]. Це перша та найважливіша частина NLU, оскільки вона встановлює сенс тексту.

Саме розпізнавання намірів дозволяє зрозуміти яка мета взаємодії, тобто, що саме користувач хоче зробити і яку відповідь отримати, відповідно у подальшому викликати на виконання саме виконання того чи іншого процесу.

Розпізнавання об'єктів — це особливий тип NLU, який фокусується на ідентифікації сутностей у повідомленні, а потім на отриманні найважливішої інформації про ці сутності [33]. Існує два типи об'єктів:

- Іменовані об'єкти – згруповані за категоріями, такими як: люди, компанії та місця розташування;
- Числові об'єкти – розпізнаються як: числа, валюти та відсотки.

1.2.3 Command execution

Command execution – окремий модуль, який призначений для виконання команд.

Найчастіше даний модуль розробляється окремо від інших елементів голосового асистенту. Фактично, він не повинен турбуватись про інші етапи.

Головне завдання command execution полягає у тому, щоб виконати певну дію в залежності від даних, які він отримав на вхід від попереднього модулю.

Сам функціонал поділяється на два різних види:

- Embedded (вбудований) – це функціонал, який не потребує виходу у мережу.
- Offboard (серверний) – це фунціонал, який оснований лише на виході до мережі інтернет.

Якщо взяти для прикладу голосовий асистент автомобіля, то Embedded частина може відповідати за наступні команди: дозволяють змінити гучність аудіо системи, зателефонувати за певним певним номером телефону (якщо доступна книга контактів), вимкнути певні асистенти тощо.

У свою чергу Offboard частина відповідає лише за команди, які потребують виходу у мережу інтернет.

Найкращим прикладом використання даного модулю є команда запиту погоди. Дана інформація зберігається на серверах для доступу до яких потрібен вихід до мережі інтернет.

Найчастіше виявлення того, чи потрібна відповідь сервера, визначається специфікацією, що відображена у форматі «.xml». Приклад специфікації можна побачити на лістингу 1.1:

```
<INTENT_Offboard_Req_Weather>
   <Offboard_Intent/>
   <SLOTS>
      <SLOT_Country/>
      <SLOT_City/>
      <SLOT_Temperature_Value/>
      <SLOT_Temperature_Scale/>
   </SLOTS>
</INTENT_Offboard_Req_Weather>
```

Лістинг 1.1 Приклад специфікації «.xml»

Як можна побачити на лістингу 1.1, даний Intent, повинен очікувати результат із серверу.

Найчастіше час очікування результату запиту на сервер дорівнює близько 3 секунд. Такий час обирається за декількох причин, а саме:

- Людський фактор. У даному випадку мається на увазі те, що якщо результат не буде отримано за 4-5 секунд, ймовірніше за все користувач натисне кнопку відміни і спробує використати фразу ще раз і скоріше за все результат знову не буде отримано;
- Якщо результат серверу не буде отримано за більш ніж 3 секунди, системі не залишиться часу, щоб обробити запит повторно і це призведе до того самого сценарію коли користувач натисне кнопку відміни.

У ситуації якщо результат від серверу не було доставлено, викликається embedded частина. У такому випадку потребується повторний аналіз усього речення.

Найчастіше будь-який запит на сервер має наступний вигляд (рис. 1.11):

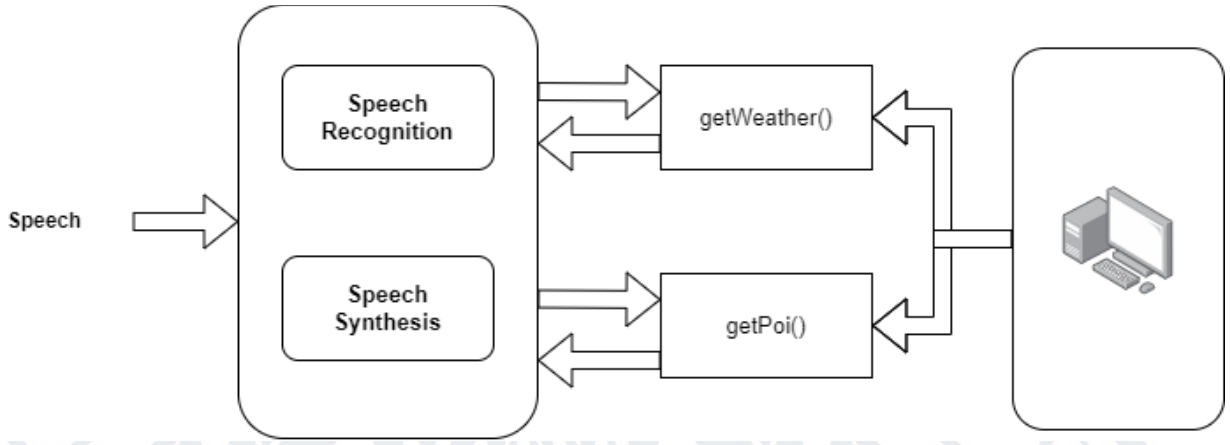

Рис. 1.11 Запит на сервер

Також є комбінований функціонал, який потребує дані як і з Offboard так і з Embedded частин. Найчастіше такий функціонал є виключенням із правил. Адже головна стратегія у більшості ситуацій – це чітке розрізняння Embedded та Offboard частин для запобігання колізій та зменшення вірогідності виникнення різних багів.

Не зважаючи на те, що модулі відрізняються за своєю реалізацією, дані, які вони отримають на вхід є однаковими, а точніше типи цих даних є однаковим. А саме – наміри та об'єкти.

Так як певні наміри можуть бути отримані лише від серверу, а деякі у свою чергу, можуть бути отримані лише з вбудованого модулю, система з легкістю може розпізнати, який функціонал користувач хоче отримати і відповідно, що їй потрібно для цього зробити.

1.2.4 Text-to-speech (TTS)

Text-to-speech (TTS) – це технологія, яка перетворює текст у штучний голос.

Дана технологія найчастіше використовується як допоміжна. Адже фактично даний модуль відповідає лише за те, щоб озвучити інформацію, яка була отримана у ході виконання команди.

Не зважаючи на те, що дана технологія є допоміжною, без неї сучасний голосовий асистент не може існувати. Також саме ця технологія, дає змогу

кінцевому користувачу відчути те, що він дійсно працює з голосовим асистентом, який обладнаний штучним інтелектом і має змогу відповісти на будь-яке його питання.

Дана технологія має свою власну велику історію. На сьогоднішній день існує два види TTS:

- Стандартний TTS – даний вид використовується у таких програмах як Google Translator, Bing Translator та інших. Він є доволі поширеним і володіє не найкращими функціоналом та параметрами. Головною його проблемою є те, що він звучить роботизовано.
- Нейронний TTS – даний вид, найчастіше використовується у сучасних голосових асистентах. На відміну від стандартних TTS має вимову більш наближену до людської.

Будь-яка технологія TTS має певну структуру, яка складається з декількох блоків, що поступово трансформують текст у мову.

Дана структура складається із наступних блоків:

- Препроцесинг
- Енкодинг
- Декодинг
- Вокодер

Кожен із блоків має свій функціонал, який повинен бути обов'язково виконаний (рис. 1.12):

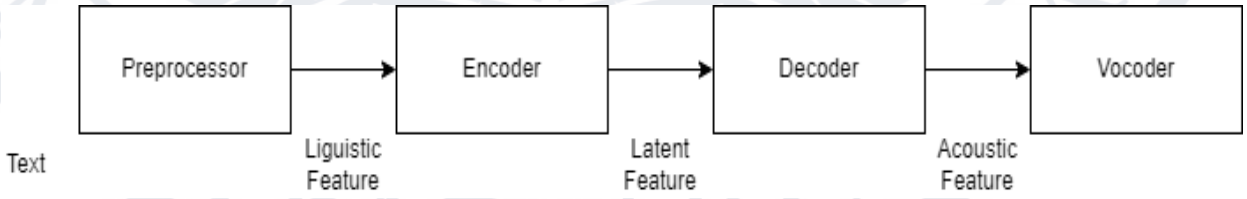

Рис. 1.12 Високорівнева діаграма перетворення тексту у мову

Розглянувши рисунок 1.12 можна побачити, що першим йде блок препроцесингу, де вхідними даними є звичайний текст.

Даний блок отримує на вхід звичайний текст, та виконує певні операції для задання лінгвістичної функції у якій будуть міститись фонеми.

Перш за все, блок препроцесингу розбиває задане речення на слова. Після чого, для кожного слова задається окремий параметр:

- Фонема (Phonemes) – до кожного слова вказується фонетична транскрипція, яка в свою чергу вказує на те, як саме має бути виконано те чи інше слово;
- Продовжність фонеми (Phoneme duration) – вказує на те, скільки часу має бути використано на кожну фонему в фінальному аудіо файлі. Фактично – це швидкість вимови;
- Висота звуку (Pitch) – даний параметр задає висоту звуку, яка відповідає за інтонацію та емоцію з якою буде вимовлено те чи інше слово;
- Енергія (Energy) – вказує величину кадрового рівня для мел-спектрограми (Mel Spectrogram) і безпосередньо впливає на гучність.

Сама лінгвістична функція, складається лише з фонем. А інші параметри, такі як: продовжність фонеми, висота звуку енергія, використовуються для тренування: предиктора тривалості, предиктора висоти звуку та предиктора енергії відповідно. Дані параметри використовуються для отримання більш природнього звуку.

Після того, як блок препроцесингу згенерував лінгвістичну функцію, викликається блок кодування, тобто сам кодувальник.

Даний блок на вхід приймає лінгвістичну функцію, а на виході отримуємо приховану функцію (рис. 1.13), яка у свою чергу містить n-вимірний графік.

Дана функція є доволі важливою у всій технології TTS, адже інші функції, такі як графік динаміків, об’єднуються та передаються до декодувальника.

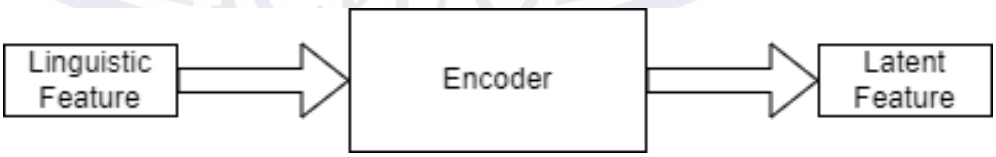

Рис. 1.13 Операція кодувальника

Також приховані функції використовуються у прогнозуванні енергії, висоти та тривалості звуку. А ці параметри, у свою чергу, відіграють ключову роль у природності звуку який ми отримаємо на виході.

Декодувальник приймає на вхід оброблену приховану функцію, та перетворює інформацію з неї на акустичну функцію (Мел-спектрограму) (рис. 1.14).

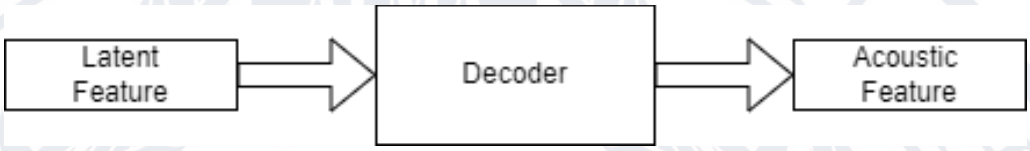

Рис. 1.14 Декодувальник

Головною причиною того, чому декодувальник перетворює дані прихованої функції у акустичну модель, а не напряму у аудіо файл, є те, що звук містить більше інформації про дисперсію, що викликає великий інформаційний розрив між входом та виходом перетворення тексту в аудіо.

Даний блок слугує для того, щоб перетворити акустичну функцію (Мел-спектрограму) уже на звичну нам звукову хвилю.

Дане перетворення можна виконати за допомогою математичної моделі Гріффіна Ліма, але на практиці даний алгоритм використовується доволі рідко через те, що він є доволі повільним.

Сьогодні набагато кращим і швидшим методом, яким найчастіше користуються усі голосові асистенти є метод нейронної мережі. Ідея даного методу полягає у тому, що ми можемо натренувати нейрону мережу вивчати відображення мел-спектрограми до аудіо хвилі.

1.3 Висновки до розділу 1

Поняття голосового асистенту, на перший погляд є доволі простим та незначним. Але якщо розглянути, як саме він влаштований, можна побачити доволі велику та громістку структуру, що несе у собі безліч нюансів, які на перший погляд можуть бути непомітними.

Будь-який голосовий асистент складається із декількох модулів, а саме:

- Automated Speech Recognition;
- Natural language understanding;
- Command execution;
- Text to speech.

Кожен із цих модулів має доволі великий запас різних технологій, які тривалий час використовувались та використовуються і зараз у голосових асистентах.

Якщо розглянути кожний модуль окремо, то можна зрозуміти, що велика кількість підходів є застарілою і на перший погляд не має сенсу. Але всупереч цьому можна надати велику кількість прикладів різних асистентів, які функціонують саме на цих застарілих підходах.

Головною причиною цього є те, що сам підхід до створення технології не потрібно змінювати цілком, адже набагато легше та продуктивніше буде внести зміни у існуючий підхід.

Саме шлях внесення змін, на даний момент, обрали майже усі компанії, які є на ринку голосових асистентів.

Повертаючись до теми голосових асистентів у автомобілях, можна сказати що єдина особливість, якою володіють подібні асистенти – це їх функціонал.

Адже як показує практика, останнім часом голосові асистенти компаній, які раніше існували лише на комп'ютерах або розумних колонках, зараз починають вмонтовуватись у автомобілі.

РОЗДІЛ 2
ПРАКТИЧНА ЧАСТИНА

2.1 Теорема вибірки та частота дискретизації

Обробка мовного сигналу, який отримує комп'ютер перед усім потребує дискретизації і зберігання аналового варіанту мовного сигналу, який генерується на вході мікрофону.

Загалом існує два параметра від яких залежить розбірливість та обсяг інформації, а саме:

- Частота дискретизації – це параметр, який контролює процес дискретизації [34];
- Кількість бітів на вибірку – даний параметр контролює роздільну здатність біта [35].

Теорема вибірки, або Sempling Theorem, визначає мінімальну частоту дискретизації при якій сигнал безперервного часу повинен бути рівномірно дискретизований, щоб вихідний сигнал міг бути повністю відновлений та реконструйований лише за допомогою цих вибірок.

Найчастіше дану теорему називають «Теорема вибірки Шеннона».

Загалом, теорема вибірки має наступне формулювання:

якщо безперервний часовий сигнал не містить компонентів із частотою більше ніж W hz, то його можна повністю визначити за допомогою однорідних вибірок, які можуть бути відібрані з частотою f_s вибірок за секунду [36], де:

$$f_s \geq 2W, \tag{2.1}$$

де f_s - кількість відліків за секунду, s;

W – частота дикретизації, hz.

Або протягом періоду відбору вибірок:

$$T \leq \frac{1}{2W} \tag{2.2}$$

де T - період вибірки, s;

W – частота дикретизації, hz.

Використовуючи об’єктно-орієнтовану мову програмування Python, наведемо приклад теореми вибірки.

Повний код можна побачити на лістингу А.4.

Python - високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і легкості читання коду [37].

Використаємо наступні бібліотеки:

- Numpy – дана бібліотека підтримує великі багатовимірні масиви та матриці, а також велику колекцію математичних функцій високого рівня для роботи з цими масивами [38].
- Matplotlib – дана бібліотека використовується для створення статичних, анімованих та інтерактивних візуалізацій на Python. Найчастіше дана бібліотека використовується саме для побудови графіків [39].

Спочатку застосуємо вибірку сигналу, що обмежена частотою 30 Hz із частотою дискретизації 600 Hz та відобразимо дану вибірку на графіку.

Для цього використаємо наступні функції:

- Numpy:
  - cos – дана функція допомагає обчислювати тригонометричний косинус для усіх X;
  - pi – константа яка містить значення числа π;
  - linspace – повертає числові пробіли рівномірно відносно інтервалу, а у якості кроку використовує номер зразка;
  - floor – математична функція, яка повертає цілу частину числа типу float, тобто нецілого числа;

- ceil – закругляє кожне значення масиву до більшого;
- arrange – повертає масив з рівномірно розташованими елементами відповідно до інтервалу. Інтервал, який повертає дана функція є напіввідкритим, тобто [Початок, Кінець)

- Matplotlib:
  - Pyplot.plot() – використовується для побудови графіків;
  - Pyplot.show() – використовується для виведення графіків;
  - Pyplot.axis() – використовується для задання певних властивостей для графіку.

Для побудови графіку задамо наступні параметри:

- f – частота у герцах (Hz)
- t_min – мінімальний діапазон часу;
- t_max – максимальний діапазон часу;
- s_range – величина послідовності.

У своєму досліджені у якості параметрів мною були задані наступні числа (Таблиця 2.1):

Таблиця 2.1 – Задані параметри використані у коді

f	t_min	t_max	S_range
30	-0.20	0.20	300

Після того, як усі параметри були задані, використаємо функцію linspace із бібліотеки Numpy, для того, щоб створити рівномірну послідовність між мінімальним та максимальним діапазоном часу.

У результаті виконання наведеної вище функції, отримаємо масив даних довжиною у 300 елементів (Лістинг 2.1):

```
[-0.2        -0.19866221 -0.19732441 -0.19598662 -0.19464883 -0.19331104
 -0.19197324 -0.19063545 -0.18929766 -0.18795987 -0.18662207 -0.18528428
 -0.18394649 -0.1826087  -0.1812709  -0.17993311 -0.17859532 -0.17725753
 -0.17591973 -0.17458194 -0.17324415 -0.17190635 -0.17056856 -0.16923077
 -0.16789298 -0.16655518 -0.16521739 -0.1638796  -0.16254181 -0.16120401
 -0.15986622 -0.15852843 -0.15719064 -0.15585284 -0.15451505 -0.15317726
 -0.15183946 -0.15050167 -0.14916388 -0.14782609 -0.14648829 -0.1451505
                          ......
  0.1451505   0.14648829  0.14782609  0.14916388  0.15050167  0.15183946
  0.15317726  0.15451505  0.15585284  0.15719064  0.15852843  0.15986622
  0.16120401  0.16254181  0.1638796   0.16521739  0.16655518  0.16789298
  0.16923077  0.17056856  0.17190635  0.17324415  0.17458194  0.17591973
  0.17725753  0.17859532  0.17993311  0.1812709   0.1826087   0.18394649
  0.18528428  0.18662207  0.18795987  0.18929766  0.19063545  0.19197324
  0.19331104  0.19464883  0.19598662  0.19732441  0.19866221  0.2        ]
```

Лістинг 2.1 Масив даних отриманий у ході створення рівномірної послідовності чисел

На основі отриманого масиву, знайдемо вибірку дискретизації сигналу, для цього використаємо наступну формулу:

$$\cos(2 * \pi * t) + \cos(2 * \pi * f * t), \qquad (2.3)$$

де f – частота, Hz;

t – діапазон часу, s.

У результаті використання даної формули отримаємо ще одну вибірку даних, довжиною у 300 чисел, яка буде вказувати на дискретизацію сигналу (Лістинг 2.2):

```
[ 1.30901699e+00  1.28537396e+00  1.20045653e+00  1.06013891e+00
  8.73798058e-01  6.53720528e-01  4.14325280e-01  1.71251611e-01
 -5.96301076e-02 -2.63222299e-01 -4.26155178e-01 -5.37632458e-01
 -5.90114273e-01 -5.79794131e-01 -5.06839717e-01 -3.75382321e-01
                         .......
 -3.75382321e-01 -5.06839717e-01 -5.79794131e-01 -5.90114273e-01
 -5.37632458e-01 -4.26155178e-01 -2.63222299e-01 -5.96301076e-02
  1.71251611e-01  4.14325280e-01  6.53720528e-01  8.73798058e-01
  1.06013891e+00  1.20045653e+00  1.28537396e+00  1.30901699e+00]
```

Лістинг 2.2 Масив даних який вказує на дискретизацію сигналу

На основі отриманих даних, побудуємо графік сигналу (рис. 2.1):

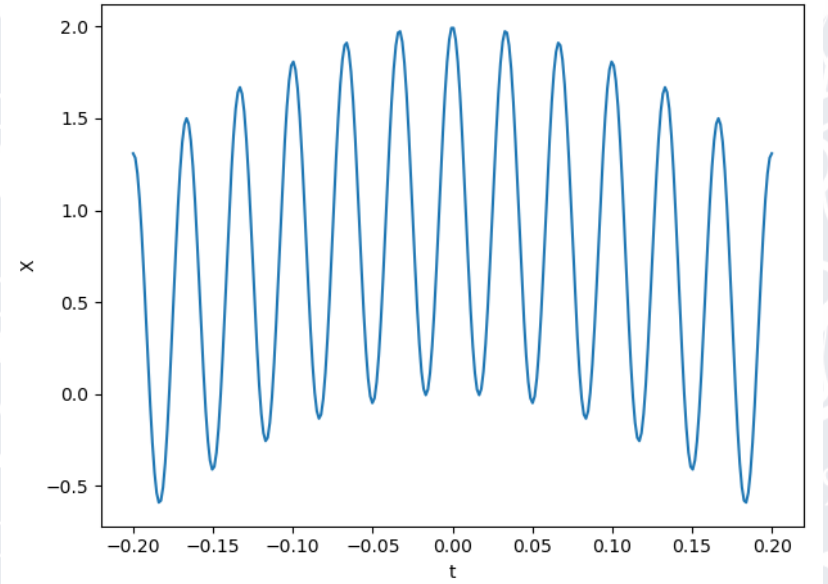

Рис. 2.1 Графік сигналу

Далі додамо на графік вибірку сигналу з частотою дискретизації 60 Hz за допомогою частоти Найквіста.

Для цього додамо нові параметри:

- T – час, він дорівнює одиниці поділеній на частоту дискретизації;
- n_min – мінімальна частота Найквіста вона дорівнює мінімальному діапазону часу поділеному на час;
- n_max – максимальна частота Найквіста вона дорівнює максимальному діапазону часу поділеному на час.

Далі, за допомогою функції arrange, отримаємо масив з рівномірно розташованими елементами відповідно до заданого інтервалу, у якості інтервалу використаємо мінімальну та максимальну частоту Найквіста, відповідно. У результаті отримаємо наступну вибірку даних (Лістинг 2.3):

```
[-12. -11. -10.  -9.  -8.  -7.  -6.  -5.  -4.  -3.  -2.  -1.   0.   1.
   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.]
```

Лістинг 2.3 Вибірка даних отримана з використанням частот Найквіста

Далі за допомогою наступної формули обчислимо дискретизацію сигналу:

$$\cos(2 * pi * n * T) + \cos\,(2 * pi * f * n * t), \qquad (2.3)$$

де f – частота, Hz;

t – діапазон часу, s;

T – період вибірки, s.

У результаті отримаємо масив із 24 елементів, які визначають дискретизацію сигналу (Лістинг 2.4):

```
[ 1.30901699 -0.59326336  1.5        -0.41221475  1.66913061 -0.25685517
  1.80901699 -0.1339746   1.91354546 -0.04894348  1.9781476  -0.0054781
  2.         -0.0054781   1.9781476  -0.04894348  1.91354546 -0.1339746
  1.80901699 -0.25685517  1.66913061 -0.41221475  1.5        -0.59326336]
```

Лістинг 2.4 Точки дискретизації сигналу

На основі отриманих даних побудуємо новий графік (рис. 2.2):

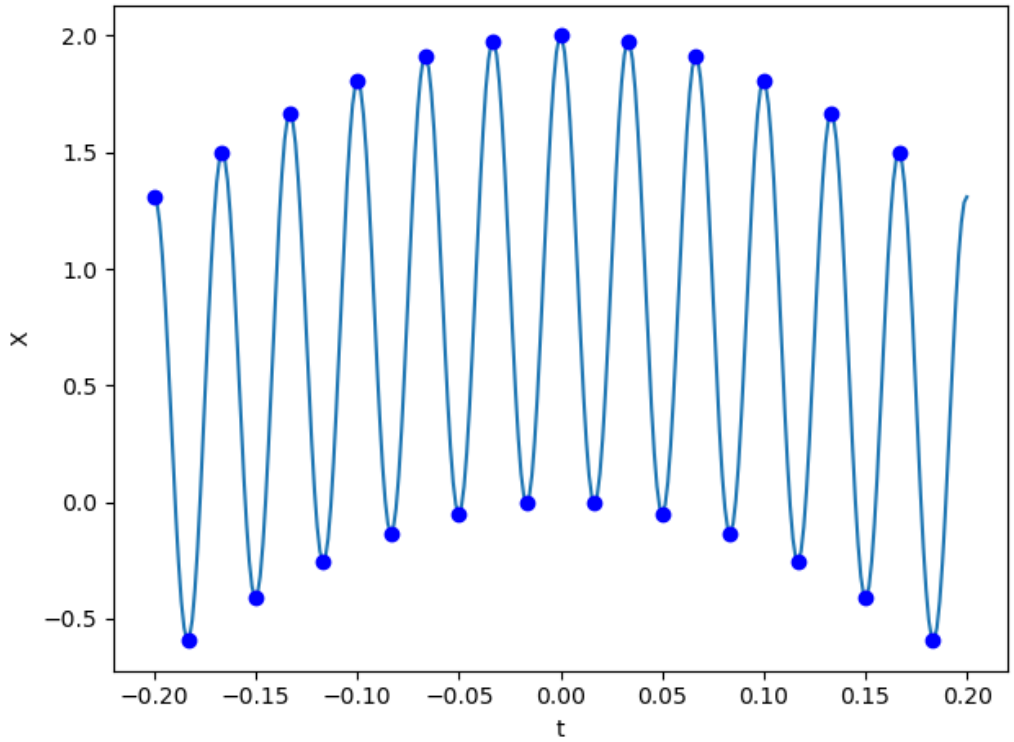

Рис. 2.2 Графік дискретизації сигналу

Далі, додамо на графік вибірку сигналу з частотою дискретизації 25 Hz на основі підвибірки Найквіста.

Для цього використаємо ті самі значення що і раніше, але змінимо значення часу з 1/60 на 1/25, так як частота дискретизації була змінена.

Після чого перерахуємо значення мінімальної та максимальної частоти Найквіста та згенеруємо новий масив даних на основі цих значень.

У результаті ми отримаємо наступний масив (Лістинг 2.5):

```
[ 1.30901699  0.84484379 -0.08004837  0.06728969  1.27760016  2.
  1.27760016  0.06728969 -0.08004837  0.84484379]
```

Лістинг 2.5 Масив даних отриманий при частоті дискретизації 25

Додамо на графік масив отриманих даних (рис. 2.3):

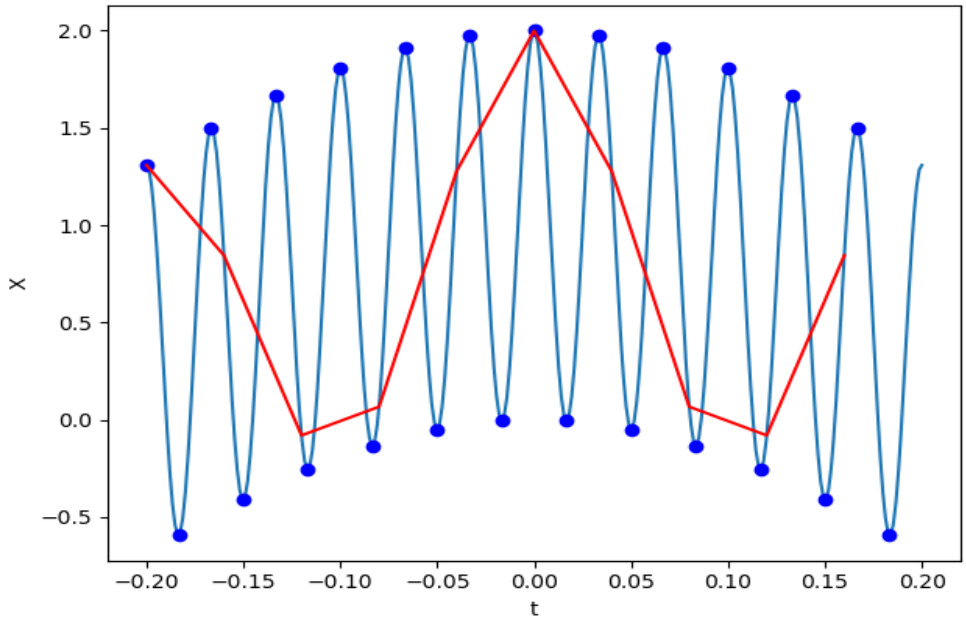

Рис. 2.3 Графік із підмножиною Найквіста

Використаємо функцію axis для задання мінімального та максимального значення для осей X та Y. Та виведемо фінальний графік (рис. 2.4):

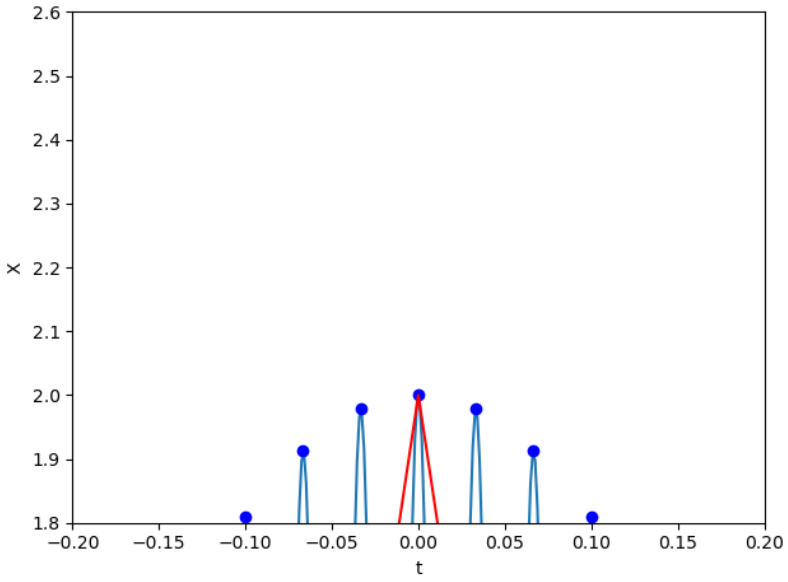

Рис. 2.4 Фінальний графік

Сформуємо та дамо визначення отриманому графіку:

- Синя лінія – відображає вибірку з обмеженою частотою 30 Hz, з частотою дискретизації 600 Hz;
- Сині крапки – вибірка сигналу з частотою дискретезації 60 Hz використовуючи частоту Найквіста;

- Червона лінія – позначає останню частину вибірки, яка є частотою нижче частоти Нейквіста. У даному випадку частота дискретизації дорівнює 25 Hz.

В загальному синя лінія являє собою вихідний сигнал, сині крапки позначають вибірки, які були отримані з частотою Найквіста, а червона лінія позначає вибірки, які отримані при частоті 25 Hz.

Як можна побачити синіх крапок доволі багато, і цього може бути достатньо, щоб відновити синю лінію, тобто вихідний сигнал. А в свою чергу, червоних недостатньо, щоб вловити коливання сигналу.

2.2 Статична мовна модель

Статична мовна модель, або Static Language Model – дана модель використовує традиційні статистичні методи, такі як N-грами, приховані моделі Маркова (HMM) і певні лінгвістичні правила, щоб дізнатися про розподіл ймовірностей слів [40].

Сьогодні статистична мовна модель є доволі легкою у побудові, адже існує безліч готових бібліотек та даних які можна використати.

Для прикладу побудуємо дану модель використовуючи мову програмування Python (див. Лістинг А.2).

За для отримання додаткового функціоналу, були використані наступні бібілотеки:

- Natural Language Toolkit (nltk) – це набір бібліотек і програм для символічної та статистичної обробки природної мови для англійської мови, написаної мовою програмування Python [41]. Також однією з найбільших переваг даної бібліотеки є те, що вона є кросплатформенною.
- Collections – це об'єкт, який використовується для зберігання різних об'єктів і забезпечення способу доступу до вміщених об'єктів і проведення ітерацій над ними [42].

Під час створення моделі були використані наступні функції з додаткових модулів:

- NLTK:
    - Trigrams – повертає триграми, створені з послідовності елементів, як ітератор;
    - Downloads – завантажує потрібні дані;
- Collections:
    - Defaultdict – це підклас класу словника, який повертає об'єкт, схожий на словник. Функціональність як звичайного словника так і defaultdict майже однакова, за винятком того факту, що defaultdict ніколи не піднімає KeyError. Він надає значення за замовчуванням для ключа, якого не існує.

У якості контейнеру даних мною був використаний корпус Reuters. Саме цей корпус підтримує триграми. Також мною було обрано саме даний корпус, за причини що він є доволі обширним і налічує більше 10 тисяч різних документів, із загальною кількістю слів близько 1.3 мільйона.

Як було згадано вище, на сьогоднішній день побудувати статичну модель доволі просто, особливо з використанням додаткових бібліотек. Тому реалізація моделі містить лише 5 функцій, а саме:

1. store_the_model;
2. download_data;
3. transform_counts_to_prob;
4. trigram_model;
5. basic_model.

Також мною було додано стандартну функцію виклику «if __name__ == '__main__':». Вона викликає на виконання усі інші функції.

Дана функція була додана на випадок, якщо потрібно буде запустити дану мовну модель безпосередньо з файлу де вона створена (Лістинг 2.5):

```
if __name__ == '__main__':
    download_data(["reuters", "punkt"])
    b_model = basic_model()
```

Лістинг 2.5 Базова функція виклику

Першою на виклик подається функція завантаження даних – download_data. Дана функція має наступний вигляд (Лістинг 2.6):

```
def download_data(data_list):
    [download(data) for data in data_list]
```

Лістинг 2.6 Функція завантаження даних

Загалом, дана функція є базовою та головною її задачею є завантажити усі потрібні дані, а саме корпуси Reuters та Punkt. Вона використовує вбудований метод бібліотеки nltk для завантаження усіх потрібних даних, які було передано їй як параметр.

Після того, як усі дані було завантажено, викликається функція store_the_model (Лістинг 2.7):

```
def store_the_model():
    return defaultdict(lambda: defaultdict(lambda: 0))
```

Лістинг 2.7 Функція збереження моделі

Дана функція використовує один із методів бібліотеки Collections, а саме методі defaultdict. Ця функція відповідає за створення словника, який містить саму мовну модель.

Далі викликається на виконання функція trigram_model, яка є уже доволі важливою. Функція має наступний вигляд (Лістинг 2.8):

```
def trigram_model(model):
    for sentence in reuters.sents():
        for word_1, word_2, word_3 in trigrams(sentence, pad_right=True,
pad_left=True):
            model[(word_1, word_2)][word_3] += 1
    return model
```

Лістинг 2.8 Функція розбиття тексту на триграми

Дана функція відповідає за розбиття тексту на триграми за допомогою функції trigrams() та підрахування кількості входжень заданого користувачем слова у наборі даних, тобто у корпусі Reuters.

Однією із найголовніших є функція transform_counts_to_prob (Лістинг 2.9):

```
def transform_counts_to_prob(model):
    for word_1_word_2 in model:
        total_count = float(sum(model[word_1_word_2].values()))
        for word_3 in model[word_1_word_2]:
            model[word_1_word_2][word_3] /= total_count
    return model
```

Лістинг 2.9 Функція обчислення ймовірностей

Дана функція відповідає за обчислення ймовірностей третього слова враховуючи попередні два слова, які також мають бути заданими окремо.

Фактично дана функція це те, що і дає нам нашу мовну модель.

Фінальною є функція basic_model. Вона має наступний вигляд (Лістинг 2.10):

```
def basic_model():
    download_data(["reuters", "punkt"])
    model = store_the_model()
    model = trigram_model(model)
    model = transform_counts_to_prob(model)
    return model
```

Лістинг 2.10 Функція basic_model

Дана функція є так званою «обгорткою», вона слугує для виклику усіх інших функцій які формують саму мовну модель та на виході повертає створену модель для того, щоб її можна було вільно використовувати без додаткових маніпуляцій у місці де потрібен лише об'єкт даної мовної моделі.

Дана модель, налаштована таким чином, щоб на основі заданих слів, передбачити третє слово яке може бути використано. Також до кожного слова буде запропоновано вірогідність з якої воно може бути вжито.

Проведемо тестування. Для цього підберемо декілька пар слів які логічно можуть бути використані у тексті.

У якості першої пари слів, було обрано - «the» та «today».

Оскільки, була додана функція для повернення об'єкту моделі, було створено окремий файл для тестування. У ньому було створено об'єкт моделі та призведено її до виконання за допомогою наступного коду (Лістинг 2.11):

```
result = dict(basic_model["today", "the"])
for key, value in result.items():
    print(f"Word - {key}\nProbability - {value}")
    print('='*50)
```

Лістинг 2.11 Виклик моделі для тестування

У ході тестування даної пари слів було отримано наступний результат (рис. 2.5):

```
Word - public; Probability - 0.05555555555555555
================================================================
Word - European; Probability - 0.05555555555555555
================================================================
Word - Bank; Probability - 0.05555555555555555
================================================================
Word - price; Probability - 0.1111111111111111
================================================================
Word - emirate; Probability - 0.05555555555555555
================================================================
Word - overseas; Probability - 0.05555555555555555
================================================================
Word - newspaper; Probability - 0.05555555555555555
================================================================
Word - company; Probability - 0.16666666666666666
================================================================
Word - Turkish; Probability - 0.05555555555555555
================================================================
Word - increase; Probability - 0.05555555555555555
================================================================
Word - options; Probability - 0.05555555555555555
================================================================
Word - Higher; Probability - 0.05555555555555555
================================================================
Word - pound; Probability - 0.05555555555555555
================================================================
Word - Italian; Probability - 0.05555555555555555
================================================================
Word - time; Probability - 0.05555555555555555
================================================================
```

Рис. 2.5 Результат мовної моделі для слів «the» та «today»

Як можна побачити на рисунку 2.5, найбільш відповідним словом для «the» та «today» є слово «company». Наступним за ймовірністю йде слово «price».

Проаналізувавши усі отримані слова, тільки два з них є дійсно найбільш підходящими.

Тепер використаємо модель для таких слів, як «business» та «the».

На рисунку 2.6 можна побачити результат тестування даної пари:

```
Word - Indians; Probability - 0.3333333333333333
============================================================
Word - following; Probability - 0.6666666666666666
============================================================
```

Рис. 2.6 Результат мовної моделі для слів «business» та «the»

Опираючись на дані з рисунку 2.6, можна зробити висновок, що тестування даної пари слів виявилось досить неефективним, адже загалом було запропоновано лише два можливі інтерпретації, що є доволі поганим результатом.

Не зважаючи на те, що результати є доволі поганими, така поведінка є передбачуваною, адже результат роботи даної мовної моделі залежить від датасету, який був використаний для її навчання.

Проведемо ще один тест із двома словами, які доволі часто зустрічаються, а саме - «the» та «price».

На рисунку 2.7 можна побачити результат який було отримано передаючи слова «the» та «price» до мовної моделі:

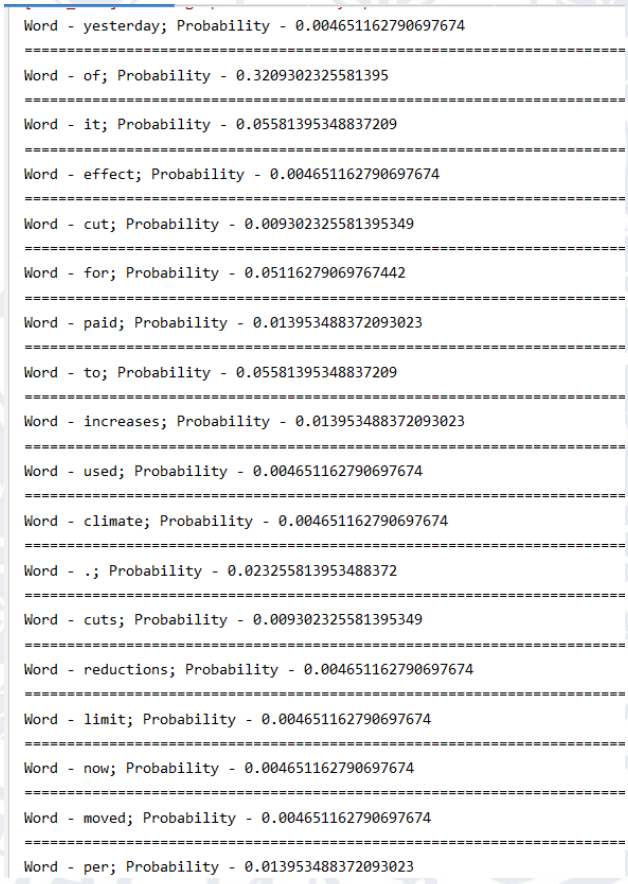

Рис. 2.7 Результат мовної моделі для слів «the» та «price»

На даному рисунку зображена лише частина отриманих результатів, так як дані слова є доволі базовими, вони зустрічаються досить часто.

За отриманими результатами найкращим словом є «of», що є доволі логічним, адже у англійській мові досить часто зустрічається вислів «the price of». У свою чергу слово, яке найрідше зустрічається, але все ж його можливо зустріти при використані слів «the» та «price» є слово «yesterday».

Так, як під час тестування найкраще себе показали слова «today» та «the», «the» та «price» спробуємо їх використати повторно. А саме спробуємо згенерувати текст за допомогою нашої моделі використовуючи саме ці слова.

Для цього була створена додаткова функція для тестування, яка використовує нашу мовну модель для генерації тексту (Лістинги 2.12, А.3):

```
def create_random_text(model, text):
    sentence_finished = False
    while not sentence_finished:
        r = random.random()
        accumulator = .0
        for word in model[tuple(text[-2:])].keys():
            accumulator += model[tuple(text[-2:])][word]
            if accumulator >= r:
                text.append(word)
                break
        if text[-2:] == [None, None]:
            sentence_finished = True
    return text
```

Лістинг 2.12 Функція генерації випадкового тексту за допомогою мовної моделі

Результат роботи програми використовуючи слова «the» та «price» можна побачити на рисунку 2.8:

```
the price would no longer employed by the end of 1987 .

Process finished with exit code 0
```

Рис. 2.8 Випадково згенерований текст на основі базової мовної моделі

Загалом отримане речення на перший погляд виглядає досить добре структурованим та легко сприймається. Що нам говорить про те, що навіть базова мовна модель працює добре.

Далі проведемо тестування статичної мовної моделі на основі інших двох слів - «today» та «the». Для цього використаємо нашу статичну мовну модель та функцію, яка була створена для тестування.

Результат роботи програми використовуючи слова «the» та «price» можна побачити на рисунку 2.9:

```
today the overseas operations lost 400 , 000 shares under the definitive pacts are likely to speed up its forecast of 450 , 000 Revs 14 . 6 mln dlrs and 230 mln dlrs .
.

Process finished with exit code 0
```

Рис. 2.9 Випадково згенерований текст на основі базової мовної моделі

Аналізуючи отриманий текст можна побачити, що перша його частина є добре структурованою та володіє певним сенсом, що до другої частини речення, то вона не дуже добре пов’язується з першою і прочитати цю частину досить важко.

У підсумку можна сказати, що навіть статична мовна модель, може бути використана за певних умов. Але її продуктивність буде залежна від вибірки використаних даних.

2.3 Моделювання мовлення в текст

Переведення мовлення у текст – це програмне забезпечення для розпізнавання мовлення [43].

Таке програмне забезпечення дозволяє розпізнавати та перекладати розмовну мову в текст за допомогою обчислювальної лінгвістики.

Також часто можна зустріти подібне програмне забезпечення у категорії розпізнавання мовлення або комп'ютерне розпізнавання мовлення. Конкретні програми, інструменти та пристрої можуть транскрибувати аудіопотоки в режимі реального часу, щоб відображати текст і діяти на нього.

Побудуємо базову модель на мові програмування Python. Дана модель буде основана на машинному навчанні та буде відповідати за розпізнавання мови базуючись на певній вибірці даних.

Так як навчання моделі займає досить великий проміжок часу, було прийнято рішення обрати певну область мовлення, що може бути розпізнано. А саме, була обрана область розпізнавання чисел, які вимовляє людина.

Приймаючи до уваги обрану область, було знайдено датасет, який складається із аудіофайлів.

Загалом цей датасет складається із 3000 аудіофайлів, які мають розширення «.wav».

Даний датасет можна розбити на 10 частин, адже сам він містить по 300 аудіозаписів кожного числа від 0 до 9. Тобто кожне число має 300 різних варіантів вимови.

Для того, щоб зробити розпізнання більш точним, кожні з цих 300 записів, можна розбити ще на 6 частин, адже кожне число має по 50 аудіозаписів записаних різними людьми.

Наповнення датасету та його структура допоможе нам зробити розпізнавання більш точним та уникнути певних проблем з акцентом та неточною вимовою.

Загалом структура датасету має наступний вигляд (рис. 2.10):

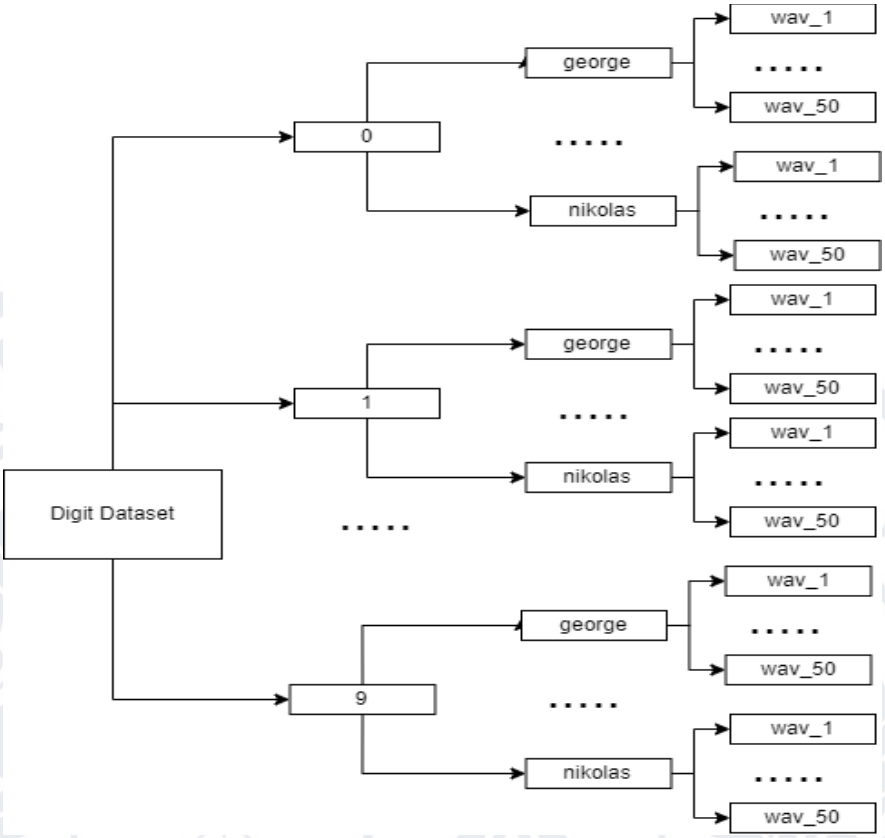

Рис. 2.10 Структура датасету

Виконавши поверхневий аналіз можна сказати, що даний датасет є доволі непоганим, адже для кожного числа ми маємо по 300 аудіозаписів із різним акцентом та різною вимовою.

Далі, розглянемо усі бібліотеки, що були використані під час написання даної програми:

- pandas;
- matplotlib;
- sklearn;
- pickle;
- numpy;
- scipy;
- python_speech_features;
- tensorflow;
- keras;

- os.

Розглянемо кожну бібліотеку більш детально.

Pandas – це бібліотека програмного забезпечення, написана для Python [44]. Дана бібліотека використовується для маніпулювання даними та їх аналізу. Перш за все дана бібліотека надає доступ до таких типів даних, як DataFrame та Series.

Ці типи даних є доволі зручними для обробки великої кількості даних, саме тому, дуже часто цю бібліотеку використовують у таких областях програмування як Data Science та Machine Learning. На додаток до доволі зручних типів даних, дана бібліотека надає безліч функціоналу, який дозволяє з легкістю маніпулювати об'єктами типу DataFrame та Series.

Також Pandas досить добре інтегрована у саму мову програмування Python, та надає безліч різного функціоналу, що допомагає з легкістю перетворити стовпець або рядок у список або словник. Подібні перетворення значно розширюють можливості та області використання даної бібліотеки, адже таким чином дозволяють використовувати функціонал звичайних списків або словників.

Бібліотека Matplotlib, найчастіше використовується для створення статичних, анімованих та інтерактивних візуалізацій на Python.

У даному випадку ця бібілотека була використана для графіку коливання сигналів аудіофайла, візуалізації спектрограм та візуалізації функції точності розпізнавання.

Бібліотека sklearn, або якщо точніше Scikit-learn, використовується для машинного навчання [45, 46]. Дана бібліотека пропонує різноманітні алгоритми класифікації, регресії та кластеризації, включаючи опорний вектор, метод випадкового лісу (random forest), посилення градієнта, алгоритм k-середніх (k-means) та DBSCAN.

Модуль pickle – вмонтований модуль

у мові програмування Python, він реалізує двійкові протоколи для серіалізації та десеріалізації структури об'єктів Python.

Головною перевагою бібліотеки Numpy, є те, що вона підтримує великі багатовимірні масиви та матриці, а також велику колекцію математичних функцій високого рівня для роботи з цими масивами.

SciPy – це бібліотека числових підпрограм для мови програмування Python [47].

Дана бібліотека містить у собі алгоритми оптимізації, інтегрування, інтерполяції, власні задачі, алгебраїчні рівняння, диференціальні рівняння та багато інших класів задач, які можуть знадобитись під час будування моделі розпізнавання мовлення.

SciPy також надає спеціалізовані структури даних, такі як розріджені матриці та k-вимірні дерева.

Бібліотека python_speech_features надає загальні функції мовлення для ASR, включаючи MFCC (Кепстральний коефіцієнт Mel Frequency) [48].

TensorFlow – це бібліотека програмного забезпечення з відкритим кодом для високопродуктивних чисельних обчислень [49]. Його гнучка архітектура дозволяє легко розгортати обчислення на різноманітних платформах (CPU, GPU, TPU), від настільних комп'ютерів до кластерів серверів і мобільних і периферійних пристроїв.

Keras – це доволі потужна, але водночас проста бібліотека Python з відкритим кодом для розробки та оцінки моделей глибокого навчання. Дана бібліотека є частиною бібліотеки TensorFlow і дозволяє визначати і тренувати моделі нейронних мереж всього за кілька рядків коду [50].

Модуль os є вбудованим у мову програмування Python. Даний модуль надає функції для взаємодії з операційною системою. Також модуль забезпечує портативний спосіб використання функціональних можливостей, залежних від операційної системи. Та одним із головних переваг, які надає даний модуль доволі велика кількість функцій для взаємодії з файловою системою незалежно від операційної системи.

Для того щоб знайти відміні риси мови, буде застосовано процедуру кодування голосу за допомогою MFCC (Mel Frequency Cepstral Coefficient).

MFCC, або Mel Frequency Cepstral Coefficient – це функція, яка широко використовується в автоматичному розпізнаванні мови. Дана функція була представлена Девісом і Мермельштайном у 1980-х роках і з тих пір залишаються найсучаснішими.

До появи MFCC лінійні коефіцієнти прогнозування (LPC) і лінійні кепстральні коефіцієнти (LPCC) були основним типом функції для автоматичного розпізнавання мовлення (ASR), особливо з класифікаторами HMM.

Саме тому було прийнято рішення використати бібліотеку python_speech_features, адже саме вона містить у собі уже реалізовану функцію для побудови мел-спектрограм.

Розглянемо реалізацію даного програмного засобу.

Реалізація програми є доволі невеликою, адже на сьогодні існує велика кількість різних бібліотек, та функцій які уже реалізують певний складний функціонал, який не доводиться писати самому.

З огляду на те, щоб можна було легко змінити реалізацію програми, мною було прийнято рішення розбити програму на декілька функцій, а саме:

- get_the_wav_files;
- import_audio_files;
- confvert_to_mfcc;
- set_the_target_label;
- create_model;
- compile_the_model;
- reshape_the_data;
- train_and_evaluate.

Також програма містить головну функцію яка викликає усі інші і слугує функцією запуску програми (Лістинг 2.13):

```
if __name__ == '__main__':
    path_to_wavs = "recordings/"
    size = 48
    list_of_wav_files = get_the_wav_files(path_to_wavs)
    list_of_vectors = import_audio_files(path_to_wavs, list_of_wav_files)

    X = confvert_to_mfcc(list_of_vectors, size)
    Y = set_the_target_label(list_of_wav_files)
    Y = np.array(Y)

    model = create_model(size)
    model = compile_the_model(model)
    reshaped_data = reshape_the_data(X, Y)

    model = train_and_evaluate(reshaped_data[0], reshaped_data[1],
reshaped_data[2], reshaped_data[3])
```

Лістинг 2.13 Головна функція програми

Також у даній функції визначені деякі константи, які потрібні для використання у інших функціях.

Першою функціє є функція get_the_wav_files() (Лістинг 2.14):

```
def get_the_wav_files(path_to_core_dir):
    return os.listdir(path_to_core_dir)
```

Лістинг 2.14 Функція отримання усіх аудіо записів

Дана функція приймає на вхід шлях де зберігаються усі аудіозаписи. На виході дана функція повертає список із усіх доступних аудіо записів.

Після чого слідує функція import_audio_files() (Лістинг 2.15):

```
def import_audio_files(path_to_wavs, all_wav_files):
    data = []
    for wav_file in all_wav_files:
        (rate, sig) = wav.read(f"{path_to_wavs}{wav_file}")
        data.append(sig)
    return data
```

Лістинг 2.15 Функція завантаження аудіозаписів

Дана функція на вхід приймає два параметри, це статичний шлях до папки з усіма аудіозаписами та список усіх доступних аудіозаписів.

Головне завдання даної функції – це зчитати аудіозапис та додати його до списку де вони зберігаються. Під час зчитування, аудіо-файл конвертується у масив даних. Приклад масиву який буде отримано на виході з функції можна побачити на рисунку 2.11:

```
[-1489  -962  -606 ... -1814 -1110   -15]
[ 36  18  63 ... -22 -35  15]
[-175 -509  293 ...  -93 -137  -89]
            ....
[  8   3   8 ... -12  -1  -6]
[12  4 15 ... 20 22 21]
[ -2  -9  -5 ... -10   2  -4]
```

Рис. 2.11 Масив із зчитаними даними

Також дана функція зчитує швидкість сигналу, але у даному випадку ця змінна не потрібна. Тому дані про швидкість сигналу не повертаються і ніде не зберігаються, окрім локальної змінної.

Після того, як усі аудіозаписи були зчитані та конвертовані у масив даних, відбувається виклик функції confvert_to_mfcc() (Лістинг 2.16):

```
def confvert_to_mfcc(data, size):
    X = []

    for i in range(len(data)):
        mfcc_feat = mfcc(data[i], nfft=512)
        mfcc_feat = np.resize(mfcc_feat, (size, 13))
        X.append(mfcc_feat)
    return np.array(X)
```

Лістинг 2.16 Функція перетворення у MFCC

На вхід дана функція приймає два параметр, а саме розмір та «data». Параметр розмір, або як можна побачити в коді «size», встановлений за замовчуванням і дорівнює 48. В свою чергу параметр «data», містить в собі той самий список масивів, які були створені з аудіозаписів.

Суть цієї функції складається в тому, щоб конвертувати уже раніше створений масив даних аудіозапису у масив MFCC. Конвертація відбувається за допомогою вбудованої функції mfcc у бібліотеку python_speech_features. Дана функція містить велику кількість різних параметрів, але був застосований лише один – nfft, даний параметр відповідає за розмір FFT.

Розмір FFT — це кількість вибірок у вікні. Тобто, частота вибірок визначає частоту Найквіста, найменшу частоту, яку може точно визначити користувач. Це частота, яка вдвічі перевищує частоту дискретизації.

Після того, як було застосовано функцію MFCC, отриманий масив потрібно привести до одного виду, для цього раніше було задано значення за замовчуванням, яке дорівнює 48.

Після того, як усі дані було перетворено, було отримано список із підсписками.

Так, як в подальшому нам потрібно працювати з масивами типу nympy, одразу перетворимо отриманий список у потрібний масив (рис. 2.12):

```
[[[ 2.12147153e+01 -2.45523332e+01  1.95296625e+01 ... -1.42497425e+01
    8.20962768e+00 -1.37571754e+01]
  [ 2.16052657e+01 -2.21508407e+01  2.78346822e+01 ... -1.73068131e+01
    5.53447402e-01 -2.20201388e+01]
                .....
  [ 1.53994028e+01 -8.83798439e+00 -3.09336523e+01 ... -9.94804678e+00
   -9.12028601e+00 -1.00069202e+01]
  [ 1.56885661e+01 -9.94021639e+00 -3.71360826e+01 ... -8.49527363e+00
   -9.40348171e+00 -1.73713281e+01]]]
```

Рис. 2.12 Фінальний масив отриманий із застосуванням MFCC

Для більш наглядної демонстрації використання функції MFCC, виведемо мел-спектрограму отриману з аудіозапису «0_jackson_0.wav». Даний аудіозапис містить одну із можливих альтернатив вимови слова «zero» (рис. 2.13):

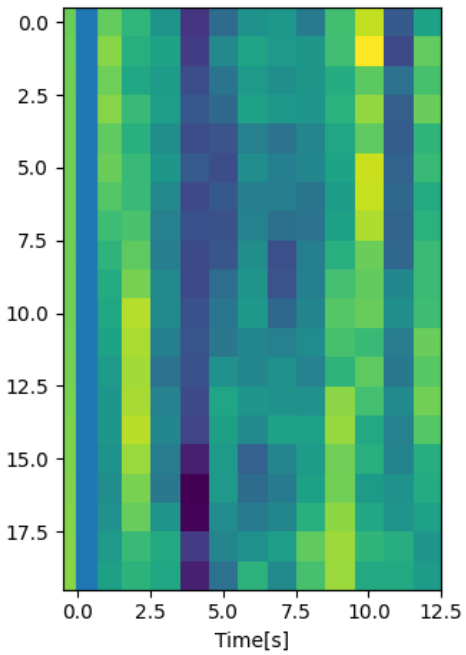

Рис. 2.13 Мел-спектрограма аудіозапису з вимовою слова «zero»

Далі використовується функція set_the_target_label() (Лістинг 2.17):

```
def set_the_target_label(soundfiles):
    y = [i[0] for i in soundfiles]
    return pd.get_dummies(y)
```

Лістинг 2.17 Функція встановлення цільових міток для прогнозування

Дана функція на вхід приймає лише один параметр – список усіх доступних аудіо-файлів за допомогою звичайного генератора, який заміняє цикл, та вбудованої функції бібліотеки pandas, а саме функції get_dumies() яка створює фіктивні змінні з об'єктів Pandas на Python.

На рисунку 2.14 можна побачити результат виконання даної функції:

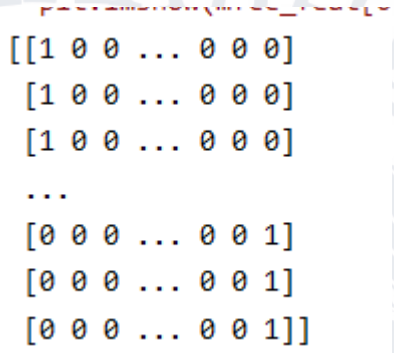

Рис. 2.14 Масив із встановленими цільовими мітками

Як можна побачити на даному рисунку, результатом є масив даних, який фактично можна назвати картою відповідності. Тобто кожен із підмасивів має загалом 10 елементів, в основному ці елементи складаються із 0, але у кожному підмасиві є одна цифра 1, якою визначено якому саме числу відповідає даний аудіозапис.

Тобто спираючись на рисунок 2.14, перші 3 завантажених файли, відповідають цифрі 0, а останні 3 цифрі 9.

Таким чином, зміна X складається з 3000 матриць 48 x 13, що відповідають функціям, які були отримані за допомогою функції MFCC для кожного запису голосу.

Після того, як цільові мітки проставлені викликається на виконання функція create_model() (Лістинги 2.18, А.1):

```
def create_model(size):
    model = Sequential()

    model.add(Conv2D(8, (3, 3), activation=”relu”, input_shape = (size, 13,
1)))
    model.add(Conv2D(8, (3, 3), activation=”relu”))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.1))
          ……

    model.add(Dense(10))
    model.add(Activation(“softmax”))
    model.summary()
    return model
```

Лістинг 2.18 Функція створення моделі

Дана функція відповідає за створення та налаштування нейронної мережі.

Для даної програми була обрана нейрона мережа – Sequential, із бібліотеки keras. Дану нейрону мережу доцільно використовувати для звичайного стека шарів, де кожен шар має рівно один вхідний тензор і один вихідний тензор.

У якості шарів дано нейронної мережі, було використано наступні шари:

- Conv2d;
- MaxPooling2D;
- Dropout;
- Flatten;
- Dense;
- Activation.

Conv2D – даний шар створює ядро згортки, яке об'єднане з вхідним шаром для отримання тензора виходів. У якості функції активації була використана функція «relu», ця функція повертає значення 0, якщо отримує від'ємний вхід, але для будь-якого позитивного значення x повертає це значення назад.

MaxPooling2D – даний шар виконує операцію максимального об'єднання для 2D просторових даних. Він знижує роздільну здатність входу вздовж його

просторових розмірів (висоти та ширини), приймаючи максимальне значення над вхідним вікном для кожного каналу входу.

Dropout – даний шар випадковим чином встановлює вхідні одиниці до 0 з частотою на кожному кроці під час тренування, це допомагає запобігти перенавантаженню. Також даний шар застосовується лише тоді, коли встановлено значення True таким чином, що під час виведення значення не втрачаються.

Наступним використаним шаром є Flatten. Він використовується для того, щоб звести усі тензори у єдиний вимір.

Dense – це правильний глибоко з'єднаний шар нейронної мережі. Це найбільш поширений і часто використовуваний шар. Щільний шар робить нижню операцію на вході і повертає вихід.

На вихідну форму шару Dense буде впливати кількість нейронів, зазначених у шарі Dense. Наприклад, якщо вхідна фігура має вигляд (8,), а кількість одиниць дорівнює 16, то вихідна форма дорівнює (16,). Весь шар матиме пакетний розмір, як перший розмір, і тому вхідна фігура буде представлена (None, 8), а вихідна фігура – як (None, 16).

Шар активації або Activation layer, можна використовувати або через шар, або через аргумент, підтримуваний усіма прямими шарами. Тобто даний шар підтримується усіма іншими шарами у якості додаткового аргументу.

Даний шар володіє такими функціями як:

- relu – застосовує функцію активації випрямленої лінійної одиниці;
- sigmoid – застосовує функцію активації сигмовидної форми. Для малих значень (<-5) повертає значення, близьке до нуля, а для великих значень (>5) результат функції наближається до 1;
- softmax – перетворює вектор значень в розподіл ймовірностей;
- softplus – функція активації софтплюса;
- softsign – функція активації softsign;
- than – гіперболічна дотична функція активації;

- selu – масштабована експоненціальна лінійна одиниця. Значення і вибираються таким чином, щоб середнє і дисперсія входів зберігалися між двома послідовними шарами до тих пір, поки ваги ініціалізуються правильно, а кількість одиниць введення є "досить великою";
- elu – мають від'ємні значення, що наближає середнє значення активацій до нуля. Середні активації, які наближаються до нуля, дозволяють швидше навчатися, наближаючи градієнт до природного градієнта;
- exponential – функція експоненціальної активації.

Результатом роботи даної функції є готова модель, яка готова до навчання (Рис. 2.15):

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 11, 8)         80

 conv2d_1 (Conv2D)           (None, 44, 9, 8)          584

 max_pooling2d (MaxPooling2D  (None, 22, 4, 8)         0
 )

 dropout (Dropout)           (None, 22, 4, 8)          0

 flatten (Flatten)           (None, 704)               0

 dense (Dense)               (None, 64)                45120

 dropout_1 (Dropout)         (None, 64)                0

 activation (Activation)     (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 dropout_2 (Dropout)         (None, 64)                0

 activation_1 (Activation)   (None, 64)                0

 dense_2 (Dense)             (None, 10)                650

 activation_2 (Activation)   (None, 10)                0

=================================================================
Total params: 50,594
Trainable params: 50,594
Non-trainable params: 0
```

Рис. 2.15 Головна інформація готової моделі

Після того, як модель була створена, потрібно її зкомпілювати, для чого і була написана функція compile_the_model() (Лістинг 2.19):

```
def compile_the_model(model):
    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss="binary_crossentropy", optimizer = sgd, metrics =
["accuracy"])
    return model
```

Лістинг 2.19 Функція компіляції моделі

На вхід дана функція приймає уже готову модель, яку потім компілює.

Кожного разу, коли нейронна мережа завершує проходження пакету через мережу та генерування результатів прогнозування, вона повинна вирішити, як використовувати різницю між отриманими результатами та значеннями, які вона знає як істинні, щоб налаштувати вагові коефіцієнти на вузлах. Алгоритм, який визначає цей крок, відомий, як алгоритм оптимізації.

У якості цього оптимізатора було обрано SGD оптимізатор.

Даний оптимізатор приймає наступні параметри:

- learning_rate – позначає швидкість навчання, за замовчуванням дорівнює 0,01;
- momentum – позначає прискорення градієнтного спуску у відповідному напрямку і гасить коливання, за замовчуванням дорівнює 0;
- Nesterov – позначає чи варто застосовувати нестеровський імпульс, за замовчуванням дорівнює False;
- name – даний параметр не є обов'язковим і позначає лише ім'я операцій, створених під час застосування градієнтів.

Після того, як був заданий оптимізатор можемо запустити компілювання моделі за допомогою команди «.compile». Дана функція приймає наступні параметри:

- optimizer – даний параметр приймає назву оптимізатора, а точніше його об'єкт, який раніше був заданий;
- loss – визначає функцію втрат, метою яких є обчислення величини, яку модель повинна прагнути мінімізувати під час навчання;

- metrics – список показників, які повинні бути оцінені моделлю під час навчання та тестування. Кожен з них може бути рядком (ім'ям вбудованої функції), функцією або екземпляром;
- loss_weight – список або словник що вказує скалярні коефіцієнти для зважування внесків втрати різних виходів моделі;
- weighted_metrics – список показників, які підлягають оцінці та зваженню під час навчання або тестування;
- steps_per_execution – кількість пакетів, які потрібно виконати під час кожного виклику;
- jit_compile – скомпілюйте модель навчального кроку за допомогою XLA.

Після того, як функція була скомпільована, нам потрібно створити дві вибірки, одну для навчання, іншу для тестування. Для створення подібних вибірок, була написана функція reshape_the_data() (Лістинг 2.20):

```
def reshape_the_data(X, Y):
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.25)
    x_train = x_train.reshape(-1, size, 13, 1)
    x_test = x_test.reshape(-1, size, 13, 1)
    return [x_train, x_test, y_train, y_test]
```

Лістинг 2.20 Функція створення вибірок для навчання та тестування

На вхід дана функція приймає два масиви X та Y:

- X – містить масиви чисел, отриманих із зчитування та обробки аудіозапису;
- Y – містить масиви чисел із цільовими мітками.

За допомогою вбудованої функції train_test_split, яка належить до бібліотеки sklear, а саме класу model_selection, було створено 4 масиви:

- x_train – масиви чисел отриманих з зчитування та обробки аудіозапису для навчання;

- x_test – масиви чисел отриманих з зчитування та обробки аудіозапису для тестування;
- y_train – масиви чисел із цільовими мітками для навчання;
- y_test – масиви чисел із цільовими мітками для тестування.

У якості вибірки для тестування, було прийнято рішення узяти лише 25% вибірки, так як це є найбільш оптимальним варіантом. Тобто у даному випадку, довжина вибірки для навчання дорівнює 2250 аудіо записів, а для тестування 750.

Для порівняння роботи, під час тестування величина вибірки для тестування буде збільшена, а величина вибірки для тренування зменшена.

У якості результату роботи функції, вона повертає вибірки для навчання та тестування.

Після того, як вибірки були створені, потрібно провести тренування моделі та оцінку. Для виконання цих операцій було створено окрему функцію train_and_evaluate() (Лістинг 2.21):

```
def train_and_evaluate(x_train, x_test, y_train, y_test):
    history = model.fit(
        x_train,
        y_train,
        epochs=18,
        batch_size=32,
        validation_split=0.2,
        shuffle=True
    )
    model.evaluate(x_test, y_test, verbose=2)
    return model
```

Лістинг 2.21 Функція тренування та оцінки моделі

На вхід дана функція приймає декілька параметрів. А саме два параметри для тренування та два для оцінки моделі.

Тренування моделі виконується за допомогою вмонтованої функції «.fit». Даний метод приймає на вхід наступні методи:

- x – вхідні дані;
- y – цільові дані;
- batch_size – кількість зразків на одне оновлення градієнта;

- epoch – кількість епох для навчання моделі;
- callbacks – список екземплярів зворотних дзвінків для застосування під час навчання;
- validation_split – виділення частки навчальних даних на яких система не буде тренуватись, а буде проводити оцінку втрат та будь-яких інших метрик;
- validation_data – дані, за якими можна оцінити втрати та будь-які модельні метрики в кінці кожної епохи;
- shuffle – дана функція застосовується для боротьби з обмеженням даних;
- class_weight – необов'язкове словникове зіставлення індексів класів зі значенням ваги, що використовується для зважування функції втрати;
- sample_weight – масив ваг Numpy для тренувальних зразків;
- initial_epoch – епоха, в яку потрібно починати тренування;
- steps_per_epoch – загальна кількість кроків до оголошення однієї епохи закінченою і початком наступної епохи;
- validation_steps – загальна кількість кроків, які потрібно виконати перед зупинкою при виконанні валідації в кінці кожної епохи;
- validation_batch_size – кількість зразків на одну партію валідації;
- validation_freq – вказується, скільки навчальних епох потрібно виконати перед виконанням нового пробігу перевірки;
- max_queue_size – максимальний розмір для черги генератора;
- workers – максимальна кількість процесів, що виконуються при використанні багатопроцесного навчання;
- use_multiprocessing – параметр, який визначає чи потрібно використовувати бегатопроцесорність.

Після того, як процес тренування моделі був успішно виконаний, потрібно провести оцінку. Оцінка проводиться за допомогою вбудованої функції «.evaluate». Дана функція приймає на вхід наступні параметри:

- x – вхідні дані, але з вибірки для тестування;
- y – цільові дані із вибірки для тестування;
- batch_size – кількість вибірок на партію обчислень;
- sample_weight – необов'язковий масив ваг Numpy для тестових зразків, що використовується для зважування функції втрати;
- steps – загальна кількість кроків (партій зразків) до оголошення раунду оцінки закінченим;
- callbacks – список зворотних дзвінків для подачі заявки під час оцінки;
- max_queu_size – список зворотних дзвінків для подачі заявки під час оцінки;
- workers – максимальна кількість процесів, що виконуються при використанні багатопроцесного навчання;
- use_multiprocessing – параметр, який визначає чи потрібно використовувати бегатопроцесорність.
- return_dict – якщо, втрати та метричні результати повертаються як словник, при цьому кожен ключ є назвою метрики.

У якості результату функція evaluate повертає скалярні тестові втрати (якщо модель має один вихід і немає показників) або список скалярів (якщо модель має кілька виходів або показників).

Результатом роботи цієї функції є натренована модель, яка готова до використання.

На основі цієї моделі проведемо декілька тестів.

Для початку візьмемо стандартні значення, а саме:

- x (розмір вибірки для навчання) – 75% (2250 аудіозаписів);
- y (розмір вибірки для тестування) – 25% (750 аудіозаписів);
- epochs (кількість епох) – 2.

Проведемо тренування моделі та на основі вибірки для тестування, за допомогою функції «predict» спробуємо визначити число.

У якості цільового масиву візьмемо наступний – [0 0 0 0 1 0 0 0 0 0]

Тобто згідно з цим масивом очікуваним числом має бути число 4.

За заданих значень, наша модель вивела нам наступні результати (рис. 2.16):

```
Predicted sound 4
Predicted probability array:
[[0.         0.01457894]
 [1.         0.12972586]
 [2.         0.02519098]
 [3.         0.02685035]
 [4.         0.29835984]
 [5.         0.28878349]
 [6.         0.03895581]
 [7.         0.01962179]
 [8.         0.03467064]
 [9.         0.12326223]]
```

Рис. 2.16 Результат тестування

Як можна побачити на рисунку 2.16, система визначила число правильно. Але якщо подивитись на ймовірності, то дані ймовірності є дуже низькими, адже від 0 до 1, найвищою ймовірністю є лише 0.298.

Також, якщо проаналізувати інші результати, то можна побачити, що доволі близько за ймовірністю підібралась п'ятірка з ймовірністю 0.288.

Тому, хоча система вірно розпізнала потрібне значення, існує доволі велика ймовірність того, що можна отримати інше число.

Проведемо ще одне тестування, але з іншими параметрами, а саме:

- x (розмір вибірки для навчання) – 75% (2250 аудіозаписів);
- y (розмір вибірки для тестування) – 25% (750 аудіозаписів);
- epochs (кількість епох) – 6.

Як можна побачити головною зміною є кількість епох. Проведемо повторне тренування моделі на основі більшої кількості епох.

У якості цільового масиву візьмемо наступний – [0 0 1 0 0 0 0 0 0 0].

Тобто згідно із цим масивом очікуваним числом має бути число 2.

Опираючись на нові початкові значення для тренування моделі, отримаємо наступний результат (рис. 2.17):

```
Predicted sound 2
Predicted probability array:
[[0.00000000e+00 1.14509508e-01]
 [1.00000000e+00 1.29128220e-02]
 [2.00000000e+00 5.57430565e-01]
 [3.00000000e+00 1.75049696e-02]
 [4.00000000e+00 2.01658495e-02]
 [5.00000000e+00 1.30267339e-02]
 [6.00000000e+00 9.46174711e-02]
 [7.00000000e+00 3.47514749e-02]
 [8.00000000e+00 1.30589575e-01]
 [9.00000000e+00 4.49106283e-03]]
```

Рис. 2.17 Результат тестування

Як можемо побачити, тепер ймовірності значно виросли, і було отримано правильне число, а саме цифру два. З доволі великою ймовірністю, а саме 0.557.

Якщо проаналізувати всі отримані дані, побачимо, що жодне число навіть близько не наблизилось за ймовірністю до двійки.

Проведемо останнє тестування, при якому використаємо наступні параметри:

- x (розмір вибірки для навчання) – 75% (2250 аудіозаписів);
- y (розмір вибірки для тестування) – 25% (750 аудіозаписів);
- epochs (кількість епох) – 18.

Головною зміною знову ж таки є збільшення кількості епох, але цього разу більш ніж у двічі. Якщо модель працює вірно, то маємо отримати ймовірність близьку до 1.

У якості цільового масиву візьмемо наступний – [0 0 0 0 0 0 0 0 8 0].

Тобто згідно з цим масивом очікуваним числом має бути число 8.

Оскільки, було змінено кількість епох, потрібно знову ж таки перетренувати модель. Після збільшення кількості епох до 18, було отримано наступний результат (рис. 2.18):

```
Predicted sound 8
Predicted probability array:
[[0.00000000e+00 1.67257582e-07]
 [1.00000000e+00 3.69253115e-07]
 [2.00000000e+00 1.02846080e-07]
 [3.00000000e+00 2.07484860e-04]
 [4.00000000e+00 2.44463005e-09]
 [5.00000000e+00 1.69018737e-08]
 [6.00000000e+00 2.45746305e-05]
 [7.00000000e+00 1.79504593e-06]
 [8.00000000e+00 9.99758065e-01]
 [9.00000000e+00 7.42925477e-06]]
```

Рис. 2.18 Результат передбачення враховуючи навчання у 18 епох

Отже, як можна побачити, модель вірно передбачила число. Але у даному випадку нам цікавіше ймовірність з якою вона це зробила, а саме 0.999.

Тобто, як і очікувалось, збільшення епох значно покращило розпізнавання і наблизило його до ідеального.

2.4 Висновки до розділу 2

У другому розділі було розглянуто декілька доволі важливих аспектів, які використовують у розпізнавані мовлення.

За результатами другого модуля, було створено декілька програмних реалізацій, які наглядно показали деякі підходи до реалізації тієї чи іншої частини голосового асистенту. А саме було реалізовано:

- Програмний засіб для відображення теореми вибірки та частоти дискретизації;
- На практиці було реалізовану базову мовну модель та наведено приклади її використання;
- Було реалізовано модель переводу мовлення у текст з використанням глибокого навчання, на основі розпізнавання чисел.

Аналізуючи усе зроблене у другому модулі, можна прийти до висновку, що на сьогоднішній день безліч технологій доступні до використання. І більш того вони реалізовані у різних бібліотеках, які також є у вільному доступі. З

кожним днем стає все більше практичного матеріалу про те, як використовувати ті чи інші бібліотеки і як результат, ті чи інші технології.

Не зважаючи на те, що ці технології несучасні, вони досі часто використовуються у великих компаніях.

Сьогодні ці технології доступні кожному для приватно використання.

ВИСНОВОК

Темою даної магістерської роботи було «Дослідження голосового асистента в автомобілях». Згідно з цим було досліджено:

- Загальні положення про мовлення та розпізнання мови;
- Загальну структуру голосового асистенту, що використовується у більшості компаній, які працюють над створенням голосових асистентів;
- Патерни та підходи до автоматичного розпізнання мовлення (Automated speech recognition);
- Головні поняття та принципи роботи моделі NLU;
- Роботу перетворення тексту у мовлення за рахунок штучної мови;
- Причини доволі повільного розвитку нових підходів до розпізнавання мовлення;
- Наведено приклади реалізації програмного забезпечення використовуючи різні підходи.

Загалом, можна зробити висновок про те, що голосові асистенти все частіше зустрічаються у нашому житі, та все частіше ми розпочинаємо ними користуватись. А голосові асистенти у автомобілях з кожним роком розвиваються все більше і більше та стають більш розповсюдженішими.

Не дивлячись на популярність голосових асистентів і теми розпізнавання людського мовлення у цілому, найчастіше використовуються підходи, які були започатковані десятки років тому. Сьогодні вони лише покращуються за рахунок певних нових технологій, найяскравіший приклад – це нейрона мережа, яку зараз активно намагаються використовувати у розпізнаванні голосу.

В свою чергу нові підходи теж не стоять на місці, але нажаль кількість досліджень та інформації доступної за цими підходами поки що доволі обмежена, власне це і є головною проблемою чому новітні підходи досі стоять на місці та не отримують активного розвитку.

Також з огляду на сформовані підходи, можна зробити висновок, що все ж більшу перспективу мають підходи основані на глибокому навчанні, при чому незалежно від того де саме вони використовуються. Адже не зважаючи на те що подібні підходи потребують великої кількості даних для навчання результати які можна отримати використовуючи дані підходи, є набагато кращими ніж при використані традиційних підходів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розпізнавання мовлення — Вікіпедія URL: https://uk.wikipedia.org/wiki/Розпізнавання_мовлення
2. Обробка мовлення URL: https://ewikiuk.top/wiki/Speech_processing
3. Автоматичне розпізнавання мови (ASR) — визначення та огляд процесу URL: https://ru.shaip.com/blog/audio-data-collection-for-automatic-speech-recognition/
4. What is Automatic Speech Recognition? | NVIDIA Technical Blog URL: https://developer.nvidia.com/blog/essential-guide-to-automatic-speech-recognition-technology/
5. What is Speech Recognition? | Rev URL: https://www.rev.com/blog/speech-to-text-technology/what-is-speech-recognition
6. Speech synthesis - Wikipedia URL: https://en.wikipedia.org/wiki/Speech_synthesis
7. What is Text Analysis? A Beginner's Guide URL: https://monkeylearn.com/text-analysis/
8. Jonathan Harrington The Phonetic Analysis of Speech Corpora, 2012. 14 с.
9. Chapter 4. Prosodic Analysis with Praat – Phonetics and PhonologyURL: https://corpus.eduhk.hk/english_pronunciation/index.php/chapter-4-prosodic-analysis-with-praat/
10. Waveform Coding - an overview | ScienceDirect Topics URL: https://www.sciencedirect.com/topics/computer-science/waveform-coding
11. A Hybrid Speech Enhancement Algorithm for Voice Assistance Application – PMC URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8588137/
12. Acoustic model - Wikipedia URL: https://en.wikipedia.org/wiki/Acoustic_model
13. Speech Recognition — Acoustic, Lexicon & Language Model | by Jonathan Hui | Medium URL: https://jonathan-hui.medium.com/speech-recognition-acoustic-lexicon-language-model-aacac0462639

14. Language model - Wikipedia URL: https://en.wikipedia.org/wiki/Language_model
15. Hidden Markov model - Wikipedia URL: https://en.wikipedia.org/wiki/Hidden_Markov_model
16. Markov Chains | Brilliant Math & Science Wiki URL: https://brilliant.org/wiki/markov-chains/
17. IBM Research advances in end-to-end speech recognition at INTERSPEECH 2019 | IBM Research Blog URL: https://www.ibm.com/blogs/research/2019/10/end-to-end-speech-recognition/
18. What is natural language understanding (NLU)? URL: https://www.techtarget.com/searchenterpriseai/definition/natural-language-understanding-NLU
19. Natural language processing – Wikipedia URL: https://en.wikipedia.org/wiki/Natural_language_processing
20. An Introduction to Natural Language Processing (NLP) | Built In URL: https://builtin.com/data-science/introduction-nlp
21. NLP vs NLU: What's The Difference? – BMC Software | Blogs URL: https://www.bmc.com/blogs/nlu-vs-nlp-natural-language-understanding-processing/
22. Natural Language Processing: Was ist das? | SAS URL: https://www.sas.com/de_at/insights/analytics/what-is-natural-language-processing-nlp.html
23. What You Need to Know About Data Preprocessing and Linguistic Annotations for Natural Language Processing (NLP) | by Kevin C Lee | Towards Data Science URL: https://towardsdatascience.com/what-you-need-to-know-about-data-preprocessing-and-linguistic-annotations-for-natural-language-439d42f2f355
24. Tokenization for Natural Language Processing | by Srinivas Chakravarthy | Towards Data Science URL: https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4?gi=77de17193135

25. Text preprocessing: Stop words removal | Chetna | Towards Data Science URL: https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a
26. Explained: Stemming vs lemmatization in NLP URL: https://analyticsindiamag.com/explained-stemming-vs-lemmatization-in-nlp/
27. NLP: POS (Part of speech) Tagging & Chunking | by Suneel Patel | Medium URL: https://suneelpatel18.medium.com/nlp-pos-part-of-speech-tagging-chunking-f72178cc7385
28. Повний посібник з аналізу настроїв - techukraine.net URL: https://goo.su/oJ7Tw
29. Синтаксис — Вікіпедія URL: https://goo.su/FulUvFy
30. Syntactic Analysis | Guide to Master Natural Language Processing(Part 11) URL: https://www.analyticsvidhya.com/blog/2021/06/part-11-step-by-step-guide-to-master-nlp-syntactic-analysis/
31. Semantic Features Analysis Definition, Examples, Applications URL: https://goo.su/e4WOr8
32. Understanding NLU benchmarks for intent detection and named-entity recognition in call centre conversations – Artefact URL: https://www.artefact.com/blog/nlu-benchmark-for-intent-detection-and-named-entity-recognition-in-call-center-conversations/
33. What Is Object Detection? - MATLAB & Simulink URL: https://nl.mathworks.com/discovery/object-detection.html
34. Частота дискретизації — Вікіпедія URL: https://goo.su/Op4vPxS
35. Передавання даних — Вікіпедія URL: https://goo.su/bncz0x
36. Теорема вибірки - LibreTexts – Ukrayinska URL: https://goo.su/WvD8P9s
37. Python — Википедия URL : https://ru.wikipedia.org/wiki/Python
38. NumPy URL: https://numpy.org/
39. Matplotlib documentation — Matplotlib 3.6.2 documentation URL: https://matplotlib.org/stable/index.html

40. Advancing Neural Language Modeling in Automatic Speech Recognition URL: https://goo-gl.me/AVJoL
41. NLTK :: Natural Language Toolkit URL: https://www.nltk.org/
42. Python's collections: A Buffet of Specialized Data Types – Real Python URL: https://realpython.com/python-collections-module/
43. Что такое преобразование речи в текст? – Руководство для начинающих специалистов по расшифровке – AWS URL: https://goo-gl.me/t5BIv
44. pandas documentation — pandas 1.5.1 documentation URL: https://pandas.pydata.org/docs/
45. scikit-learn documentation — DevDocs URL: https://devdocs.io/scikit_learn/
46. Documentation scikit-learn: machine learning in Python — scikit-learn 0.21.3 documentation URL: https://scikit-learn.org/0.21/documentation.html
47. SciPy documentation — SciPy v1.9.3 Manual URL: https://docs.scipy.org/doc/scipy/
48. Welcome to python_speech_features's documentation! — python_speech_features 0.1.0 documentation URL: https://python-speech-features.readthedocs.io/en/latest/
49. API Documentation | TensorFlow v2.10.0 URL: https://www.tensorflow.org/api_docs
50. Getting started URL: https://keras.io/getting_started/

ДОДАТОК А

Лістинги

```
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from IPython.display import HTML
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
from scipy.io import wavfile as wav
from python_speech_features import mfcc

import tensorflow as tf

from keras.models import Sequential
from keras.layers import Dense, Flatten, Activation, Dropout, LSTM
from keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D
from keras.optimizers import SGD
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"


def get_the_wav_files(path_to_core_dir):
    return os.listdir(path_to_core_dir)


def import_audio_files(path_to_wavs, all_wav_files):
    data = []
    for wav_file in all_wav_files:
        (rate, sig) = wav.read(f"{path_to_wavs}{wav_file}")
        data.append(sig)
    return data


def confvert_to_mfcc(data, size):
    X = []
    for i in range(len(data)):
        mfcc_feat = mfcc(data[i], nfft=512)
        mfcc_feat = np.resize(mfcc_feat, (size, 13))
        X.append(mfcc_feat)
    return np.array(X)


def set_the_target_label(soundfiles):
    y = [i[0] for i in soundfiles]
    return pd.get_dummies(y)


def create_model(size):
    model = Sequential()

    model.add(Conv2D(8, (3, 3), activation="relu", input_shape = (size, 13, 1)))
    model.add(Conv2D(8, (3, 3), activation="relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.1))
```

```
    model.add(Flatten(input_shape=(size, 13, 1)))

    model.add(Dense(64))
    model.add(Dropout(0.16))
    model.add(Activation("relu"))

    model.add(Dense(64))
    model.add(Dropout(0.12))
    model.add(Activation("relu"))

    model.add(Dense(10))
    model.add(Activation("softmax"))
    model.summary()
    return model


def compile_the_model(model):
    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(loss="binary_crossentropy", optimizer = sgd, metrics = ["accuracy"])
    return model


def reshape_the_data(X, Y):
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.25)
    x_train = x_train.reshape(-1, size, 13, 1)
    x_test = x_test.reshape(-1, size, 13, 1)
    return [x_train, x_test, y_train, y_test]


def train_and_evaluate(x_train, x_test, y_train, y_test):
    history = model.fit(
        x_train,
        y_train,
        epochs=18,
        batch_size=32,
        validation_split=0.2,
        shuffle=True
    )
    model.evaluate(x_test, y_test, verbose=2)
    return model


if __name__ == '__main__':
    path_to_wavs = "recordings/"
    size = 48
    list_of_wav_files = get_the_wav_files(path_to_wavs)
    list_of_vectors = import_audio_files(path_to_wavs, list_of_wav_files)

    X = confvert_to_mfcc(list_of_vectors, size)
    Y = set_the_target_label(list_of_wav_files)
    Y = np.array(Y)

    model = create_model(size)
    model = compile_the_model(model)
    reshaped_data = reshape_the_data(X, Y)

    model = train_and_evaluate(reshaped_data[0], reshaped_data[1], reshaped_data[2],
reshaped_data[3])

    print("\n\n"+ "="*100)
    sound_index = 10
```

```
    pred = model.predict(reshaped_data[1][sound_index].reshape(-1, size, 13, 1))

    available_numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    print(f"Sound:\n{reshaped_data[3][sound_index]}\n" + "="*100)
    print(f"Predicted sound {pred.argmax()}")
    print("Predicted probability array:")
    print(np.column_stack((available_numbers, list(pred[0]))))
```

Лістинг А.1 Speech to text

```
from nltk.corpus import reuters
from nltk import bigrams, trigrams, download
from collections import Counter, defaultdict


def store_the_model():
    return defaultdict(lambda: defaultdict(lambda: 0))


def transform_counts_to_prob(model):
    for word_1_word_2 in model:
        total_count = float(sum(model[word_1_word_2].values()))
        for word_3 in model[word_1_word_2]:
            model[word_1_word_2][word_3] /= total_count
    return model


def trigram_model(model):
    for sentence in reuters.sents():
        for word_1, word_2, word_3 in trigrams(sentence, pad_right=True,
pad_left=True):
            model[(word_1, word_2)][word_3] += 1
    return model


def download_data(data_list):
    [download(data) for data in data_list]


def basic_model():
    download_data(["reuters", "punkt"])
    model = store_the_model()
    model = trigram_model(model)
    model = transform_counts_to_prob(model)
    return model


if __name__ == '__main__':
    download_data(["reuters", "punkt"])
    b_model = basic_model()
```

Лістинг А.2 Basic language model

```
import random
import basic_language_model


def create_random_text(model, text):
    sentence_finished = False
    while not sentence_finished:
        r = random.random()
        accumulator = .0
        for word in model[tuple(text[-2:])].keys():
            accumulator += model[tuple(text[-2:])][word]
            if accumulator >= r:
                text.append(word)
                break
        if text[-2:] == [None, None]:
            sentence_finished = True
    return text


if __name__ == '__main__':
    text = ["today", "the"]
    basic_model = basic_language_model.basic_model()
    random_text = create_random_text(basic_model, text)
    print()
    print(' '.join([t for t in random_text if t]))
```

Лістинг А.3 Basic language model testing

```
from numpy import cos, pi, ceil, floor, arange, linspace
from matplotlib import pyplot


def generate_sampling_rate(rate):
    T = 1 / rate
    n_min = ceil(t_min / T)
    n_max = floor(t_max / T)
    n = arange(n_min, n_max)
    x = cos(2 * pi * n * T) + cos(2 * pi * f * n * T)
    return [x, n, T]


if __name__ == '__main__':
    f = 30
    t_min = -0.2
    t_max = 0.2
    s_range = 300
    t = linspace(t_min, t_max, s_range)
    x = cos(2*pi*t) + cos(2*pi*f*t)
    pyplot.xlabel("t")
    pyplot.ylabel("X")
    pyplot.plot(t, x)
    x1 = generate_sampling_rate(60)
    pyplot.plot(x1[1] * x1[2], x1[0], "bo")
    x2 = generate_sampling_rate(25)
    print(x2[0])
    pyplot.plot(x2[1] * x2[2], x2[0], "-r", markersize=6)
    pyplot.axis([-0.2, 0.2, 1.8, 2.6])
    pyplot.show()
```

Лістинг А.4 Sempling theorem