МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент
_____Є. В. Сіденко
«____»_____202_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

МУЗИЧНИЙ СЕРВІС ІЗ НЕЙРОМЕРЕЖЕВОЮ СИСТЕМОЮ РЕКОМЕНДАЦІЙ ТА КРОСПЛАТФОРМНИМ КЛІЄНТОМ

Спеціальність 122 «Комп'ютерні науки»

122 – МКР – 601.1710126

Виконав студент 6-го курсу, групи 601
_____ В. О. Рудь
«16» лютого 2023 р.

Керівник: д.ф-м. наук, професор
_____ Е. А. Лисенков
«16» лютого 2023 р.

Миколаїв – 2023

Чорноморський національний університет ім. Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент

_____ Є. В. Сіденко

« » 20 р.

ЗАВДАННЯ

на магістерську кваліфікаційну роботу

Рудю Володимиру Олеговичу
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Музичний сервіс із нейромережевою системою рекомендацій та кросплатформним клієнтом».

Керівник роботи Лисенков Едуард Анатолійович, д.ф.-м.н, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «03» листопада 2022 р. № 199

2. Строк подання студентом роботи _16_ лютого 2023 р.

3. Вхідні (початкові) дані до роботи: користувацькі оцінки контенту який існує у сервісі, статистика щодо відтворювання музики. Очікуваний результат: система рекомендації на основі нейромережі, результатом роботи якої є рекомендаційний список музики на основі даних користувачів.

4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

– аналіз сучасного стану задачі розробки рекомендаційних систем на основі користувацької статистики та оцінок користувачів;

– огляд аналогів системи;

– огляд існуючих методів нейромережевих рекомендаційних систем;

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини:

_____

_____

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці		
Методична частина		

Керівник роботи __ д.ф.-м.н, професор Лисенков Е.А._____

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання ___Рудь В.О._____

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «_7_» _____листопада____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
Виконання магістерської кваліфікаційної роботи

Тема: Музичний сервіс із нейромережевою системою рекомендацій та кросплатформним клієнтом

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР	01.09.2022	20.10.2022	
2	Отримання завдання на виконання МКР	21.10.2022	10.11.2022	
3	Складання календарного плану на період виконання МКР	11.11.2022	15.11.2022	
4	Огляд літератури за темою дослідження	16.11.2022	27.11.2022	
5	Проходження переддипломної практики, збір та аналіз матеріалів до МКР	28.11.2022	18.12.2022	
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	19.12.2022	12.01.2023	
7	Опис фахової частини МКР, зокрема дослідження публікацій щодо створення систем рекомендацій на основі нейромережі	13.01.2023	25.01.2023	
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2023	02.02.2023	
9	Попередній захист МКР на засіданні комісії кафедри	03.02.2023	03.02.2023	
10	Корегування роботи за результатами попереднього захисту	04.02.2023	06.02.2023	
11	Остаточне оформлення пояснювальної записки та слайдів доповіді для захисту	015.02.2023	09.02.2023	
12	Подання МКР рецензенту	15.02.2023	10.02.2023	
13	Рецензування МКР	16.02.2023	12.02.2023	
14	Подання МКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	16.02.2023	16.02.2023	
15	Захист МКР перед екзаменаційною комісією (ЕК)	23.02.2023	23.02.2023	

Розробив студент _____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи _____ _____
(*наук. ступінь, вчене звання, прізвище та ініціали)* *(підпис)*

«_12_» _11_ 2022 р.

АНОТАЦІЯ

до магістерської кваліфікаційної роботи
студента групи 601м ЧНУ ім. Петра Могили
Рудя Володимира Олеговича
на тему: **“ МУЗИЧНИЙ СЕРВІС ІЗ НЕЙРОМЕРЕЖЕВОЮ СИСТЕМОЮ РЕКОМЕНДАЦІЙ ТА КРОСПЛАТФОРМНИМ КЛІЄНТОМ”**

Актуальність. Музичні сервіси стали важливою складовою життя багатьох людей у світі. Кількість підписників YouTube Music, Spotify та інших давно перейшла межі сотень мільйонів активних підписників.

Ці сервіси дають змогу людям слухати музику у зручному вигляді через зручні клієнтські застосунки або веб-клієнти, мати змогу одразу отримувати музику з онлайну без довгих пошуків необхідних файлів та дисків. Також ці сервіси допомагають знайти нову музику беручи до уваги смаки користувача, аналізуючи його плейлісти, історію прослуховувань виконавців, їх пісень.

Об’єктом дослідження є процес надання рекомендацій на основі рекомендаційної системи.

Предметом дослідження є методи розробки рекомендаційної системи.

Метою даного дослідження є покращення роботи рекомендаційних систем за допомогою нейромережі із використанням k-means алгоритму.

В якості засобу розроблення системи було обрано PyCharm та мову програмування Python. Інструментами розроблення були обрані бібліотека для імпорту та експорту даних Pickle та бібліотека для роботи з алгоритмами нейронних мереж sckit-learn, бібліотека для побудови графічних інтерфейсів Kivy..

Результатом роботи є повноцінна система для прослуховування музики з системою рекомендацій.

Робота складається зі вступу, 3 розділів, 2 спеціальних розділів, висновку, списку використаних джерел (28 найменувань). Робота містить 19 рисунків. Загальний обсяг становить 103 сторінок.

Ключові слова: *система рекомендацій, підбір рекомендацій, музичний плеєр, k-means.*

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Rud Volodymyr

“ MUSIC SERVICE WITH NEURAL NETWORK RECOMMENDATION SYSTEM AND CROSS-PLATFORM CLIENT”

Topicality. Music services have become an important part of the lives of many people in the world. The number of subscribers to YouTube Music, Spotify and others has long crossed the hundreds of millions of active subscribers.

These services allow people to listen to music in a convenient way through convenient client applications or web clients, to be able to immediately receive music from the Internet without long searches for the necessary files and discs. Also, these services help to find new music, taking into account the user's tastes, analyzing his playlists, listening history of artists, their songs.

The object of the research is the process of providing recommendations based on the recommendation system.

The subject of the research is the methods of developing a recommendation system.

The purpose of this research is to improve the performance of recommender systems using a neural network using the k–means algorithm.

PyCharm and the Python programming language were chosen as a system development tool. Pickle library for data import and export and sckit-learn library for working with neural network algorithms, Kivy library for building graphical interfaces were chosen as development tools.

The result is a full-fledged system for listening to music with a system of recommendations.

The work consists of an introduction, 3 chapters, 2 special chapters, a conclusion, a list of sources used (28 items). The work contains 19 drawings. The total volume is 103 pages.

Keywords: recommendation system, selection of recommendations, music player, k-means.

ЗМІСТ

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«МУЗИЧНИЙ СЕРВІС ІЗ НЕЙРОМЕРЕЖЕВОЮ СИСТЕМОЮ РЕКОМЕНДАЦІЙ ТА КРОСПЛАТФОРМНИМ КЛІЄНТОМ»

Спеціальність 122 «Комп’ютерні науки»

122 – МКР – 601.1710126

Виконав студент 6-го курсу, групи 601
_____ В. О. Рудь
«16» лютого 2023 р.

Керівник: д.ф.-м. наук, професор
_____Е. А. Лисенков
«16» лютого 2023 р.

Миколаїв – 2023

ВСТУП

Рекомендаційна система дозволяє людині позначити свої смаки і повертає результати, цікаві для нього, базуючись на оцінках інших користувачів і своїх припущеннях. На відміну від пошукових систем, для отримання від системи відповіді не потрібно чіткого завдання запиту. Замість цього користувачеві пропонується оцінити деякі об'єкти з колекції, і на підставі цих його оцінок і порівняння їх з оцінками попередніх користувачів будуються припущення про смаки користувача і повертаються мають найтісніший контакт з ним результати, формуючи для нього персоналізовану видачу. Як набір оцінюваних об'єктів можуть, наприклад, виступати: каталог фільмів, пісні, фото і т.п. У сферу застосування подібних систем входять і ситуації, коли користувач не шукає інформацію по конкретному ключовому слову, а, наприклад, хоче отримати список сучасних фільмів, схожих за тематикою на ті, які він переглядав до цього.

Метою даного дослідження є покращення роботи рекомендаційних систем за допомогою нейромережі із використанням k-means алгоритму:

– вибір найбільш ефективного рекомендаційного методу рекомендацій мультимедійних матеріалів;

– створення алгоритму рекомендацій мультимедійних матеріалів;

– створення рекомендаційної системи, яка буде пропонувати користувачу відповідно до встановленого режиму (фільми, музика і т.д.) відповідні матеріали, спираючись на те, які побажання він має.

Об’єктом для дослідження є процес надання рекомендацій на основі рекомендаційної системи.

Предметом дослідження є методи розробки рекомендаційної системи.

В ході дослідження належить розібратися в рекомендаційних методах, в їх особливостях, можливостях і доповненнях. Також вивчити основи мови програмування Python# та мови структурованих запитів SQL.

Робота складається з 5. розділів, 19 рисунків, 28. літературних джерел, загальний обсяг роботи 103 сторінки.

1 ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АНАЛОГІВ, МЕТОДІВ ТА СИСТЕМ

1.1 Огляд інтелектуальних рекомендаційних методів

Рекомендаційна система - це система, що рекомендує товари користувачам серед величезного потоку інформації, в залежності від їх потреб. Товари – елементи конкретного сервісу, до яких користувач має інтерес: фільми, ресторани, книги, статті та ін. Інтереси користувачів можуть бути представлені декількома способами: із застосуванням оцінок, які користувачі надають товарам або за допомогою ключових слів кожного товару. Щоб зберігати вподобання користувачів стосовно товарів, РС використовують профілі користувачів. У більшості РС профіль користувача містить набори оцінок та/або ключових слів (тегів). Оцінки, надані користувачами товарам, можуть належати різним проміжкам (0-1, 1-5, 1-10): чим вищий рейтинг, тим більше конкретний товар сподобався користувачу. Після кожного оцінювання все рейтинги користувача агрегуються через ряд обчислень, вимірюються схожість користувачів, а потім прогнозуються рекомендації для даного користувача. Ключові слова автоматично підвантажуються з текстів або товарів, які користувачі проглядали або оцінювали в минулому. Вони також можуть мати ваги в залежності від того, на скільки користувач оцінив конкретне слово, або більш значущі слова матимуть більшу вагу, ніж менш значущі (алгоритм TF-IDF) [1]. Після цього тексти (товари) зіставляються з профілем користувача та найбільш відповідні йому – рекомендуються. Рейтинги можуть бути явними та неявними. Явна оцінка – це оцінка, якою користувач показав зацікавленість даним товаром в межах своєї системи оцінювання. Неявні рейтинги вираховуються з історії покупок або поведінки користувачів. До їх переваг можна віднести зниження навантаження

на користувача оцінюванням товарів. Джерелом неявних рейтингів можуть бути час, витрачений на читання статті, посилання на товар в інших джерелах 14 (наприклад алгоритм ранжування сторінок Google). Інші індикатори поведінки перегляду, як рух курсору, ввід клавіатури та швидкість прокрутки сторінки, також були досліджені в якості неявних показників інтересу та показали непогані результати.

У більшості рекомендаційних систем використовується один з двох базових підходів: Колаборативна фільтрація (collaborative filtering) [2] та контентна фільтрація (content-based filtering). Також існує клас підходів, що базуються на поєднанні двох основних – гібридна фільтрація (hybrid filtering).

1.2 Колаборативна фільтрація

Метод прогнозу в рекомендаційних системах, який використовує відомі переваги (оцінки) групи користувачів для прогнозування невідомих переваг (оцінок) іншого користувача. За допомогою цього алгоритму будується певна таблиця користувачів, які групуються за схожістю, та прогнозуються результати для інших користувачів.

Колаборативна фільтрація прогнозує рекомендації, засновані на моделі попередньої поведінки користувача. Ця модель може бути побудована виключно на основі поведінки цього користувача або – що більш ефективно – з урахуванням поведінки інших користувачів з подібними характеристиками. У тих випадках, коли колаборативна фільтрація бере до уваги реакцію інших користувачів, вона використовує знання про групу (group knowledge) [3] для вироблення рекомендацій на основі схожості користувачів. По суті рекомендації базуються на автоматичній взаємодії множини користувачів і на виділені (методом фільтрації) тих користувачів, які демонструють схожі уподобання або шаблони поведінки. Наприклад, при створенні веб-блогу, на якому необхідно

запровадити рекомендаційну систему на основі інформації від багатьох користувачів, які дивляться та коментують фільми чи серіали, можна згрупувати цих користувачів за їх інтересами. Можна об'єднати в одну групу користувачів, які дивляться кілька однакових фільмів, серіалів і за цією інформацією ідентифікувати найпопулярніші стрічки серед тих, які дивляться учасники цієї групи. Потім конкретному користувачу з 16 цієї групи будуть рекомендуватися найпопулярніший фільм, серіал з тих, які він ще не бачив.

Переваги: швидка робота алгоритмів (K-based та ін.) [4] – мала кількість ітерацій; прості в реалізації.

Недоліки: не вирішені проблеми холодного старту, шахрайства, нема що рекомендувати новим або нетиповим користувачам; розріджені матриці оцінок (іноді неможливо зробити прогноз).

1.3 Контентна фільтрація

Тематична фільтрація формує рекомендацію на базі поведінки користувача. Наприклад, цей підхід може використовувати ретроспективну інформацію про перегляди або ж прослуховування мультимедійних матеріалів. Якщо який-небудь користувач зазвичай дивиться фільми, серіали одного жанру, або регулярно залишає коментарі під матеріалами іншого жанру, то тематична фільтрація може використовувати цю ретроспективну інформацію для виявлення подібного контенту і пропозиції такого контенту як рекомендованого для цього користувача. Цей контент може бути визначений в ручному режимі або завантажений автоматично на базі інших методів подібності.

Переваги: більш точний результат; немає проблеми холодного старту, оскільки рекомендації базуються на моделі об’єкта, а не на попередніх оцінках користувачів.

Недоліки: “затратне” створення моделі (її побудова досить складна), невисока швидкодія алгоритмів (багато обчислень) та втрата точності при скороченні параметрів моделі.

1.4 Гібридні алгоритми фільтрації

Гібридні підходи, які поєднують колаборативну і контентну фільтрацію, також підвищують ефективність (і складність) рекомендаційних систем. Об'єднання результатів колаборативної і контентної фільтрації потенційно дозволяє підвищити точність рекомендації. Гібридний підхід може бути корисний, якщо застосування колаборативної фільтрації відбувається на сильно розріджених даних (приклад холодного старту). Гібридний підхід дозволяє спочатку зважувати результати згідно контентної фільтрації, а потім зміщувати ці ваги у напрямку до колаборативної фільтрації (в міру "визрівання" доступного набору даних для конкретного користувача).

Тип	Характеристика
Зважена	Оцінки, отримані методами колаборативної та контентної фільтрації, агрегуються для отримання єдиної рекомендації.
Комутуюча	Система використовує критерії, щоб перемикатися між методами контентної та колаборативної фільтрації.
Каскадна	Рекомендація такої моделі з'являється за рахунок удосконалення рекомендації інших систем.
Комбінаційна	Критерії з різних джерел рекомендацій вкидаються в єдиний рекомендаційний алгоритм.
Нарощувальна	Вихідні дані, отримані з однієї системи, використовуються як вхідні для іншої.
Змішана	Рекомендації з різних рекомендаційних систем показуються одночасно
Мета-рівень	Навчена модель з однієї рекомендаційної системи використовується в якості системних даних для іншої системи.

Рисунок 1.1 – Моделі рекомендаційних систем на базі гібридних алгоритмів

Переваги: велика швидкодія; кращі результати.

Недоліки: дуже дорога розробка рекомендаційної системи, оскільки реалізація цього типу алгоритмів дуже складна; важко підтримувати, оскільки навіть незначні зміни в роботі призводять до змін роботи алгоритму.

1.5 Огляд інтелектуальних рекомендаційних систем

LinkedIn – це соціальна мережа для бізнес-спільноти. Заснована у 2003 році, онлайновий сайт – це місце для професіоналів, які можуть спілкуватися з минулими та нинішніми колегами, збільшувати кількість ділових зв'язків, мережу у своїй галузі, обговорювати бізнес-ідеї, шукати роботу та шукати нових робочих місць. Користувачі LinkedIn створюють професійні профілі, подібні до резюме, які дозволяють іншим учасникам сайту більше дізнатися про свій бізнес, спеціалісти та групи чи організації, до яких вони належать. Коли користувач створює свій профіль, він може додати інших користувачів до своєї мережі. Нещодавно LinkedIn запустив LinkedIn Learning [5]. Раніше їхні рекомендації базувалися на загальних та рандомізованих підходах, проте вони вирішили представити персоналізовані рекомендації, й саме вони допомогли їм покращити зацікавитись до свого ресурсу на 58%.

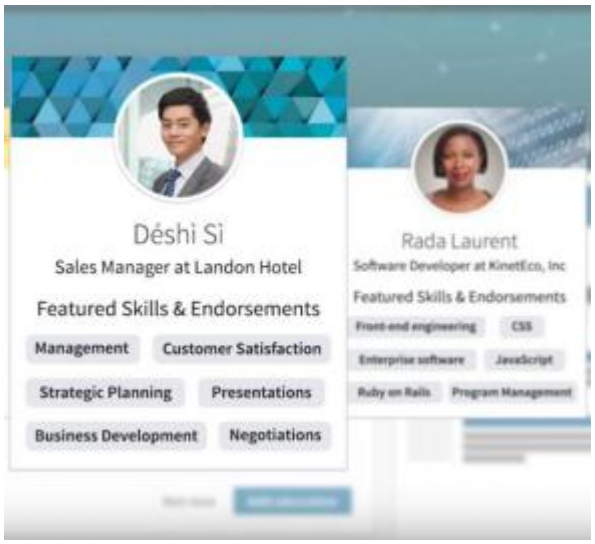

Рисунок 1.2 – Профіль користувача зі списком навичок та інтересів

При створені нового система LinkedIn нічого про нього не знає й через це не може нічого порекомендувати, отож щоб уникнути цю проблему, після створення користувача йому пропонується відповісти на ряд питань,

спрямованих на визначення його інтересів, а у випадку відмови, система буде використовувати дані фреймворків з якими працював користувач, зазначеними у його профілі.

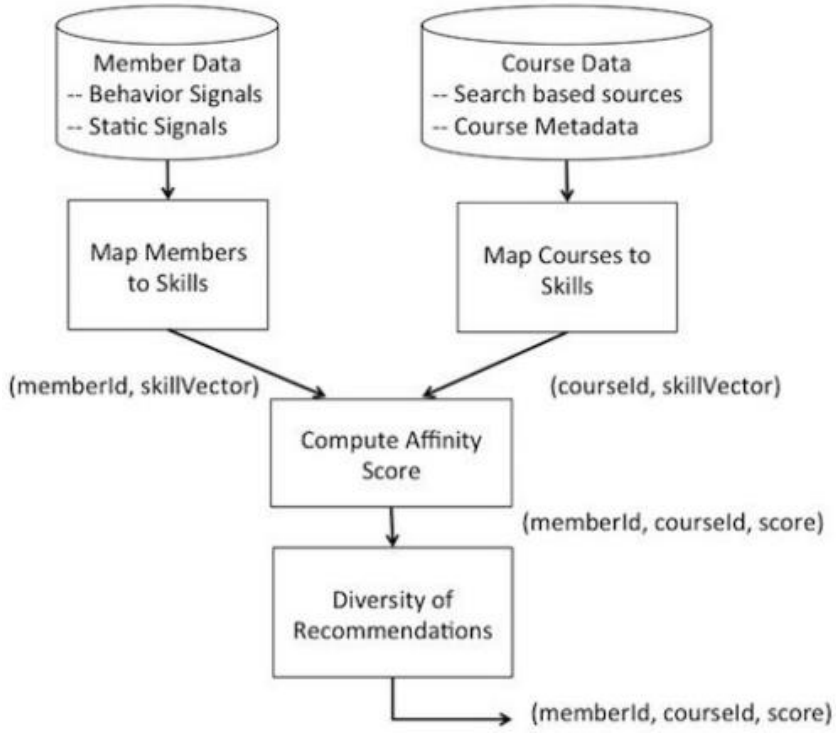

Рисунок 1.3 – Загальна архітектура рекомендаційної системи в мережі LinkedIn

Вони використовують два джерела даних: БД курсів та інформацію про користувачів. Їхня задача полягає у використанні навичок в якості властивостей для представлення користувачів та курсів. Курси можуть бути представлений векторами навичок, які можуть бути здобуті проходячи цей курс. Користувачі також можуть бути представлені в цьому простору навичок, використовуючи різні джерела інформації.

Рисунок 1.4 – Рекомендації у мережі LinkedIn, які були згенеровані використовуючи інтереси

В той час, як в профілі користувача явно задані навички, існують й інші джерела, що можуть використовуватися для виявлення не явних навичок, наприклад, може бути використані професійні зв'язки або активність користувачів у мережі. Як тільки користувачі та курси утворили простір навичок, система оцінює спорідненість між учасниками та параметрами курсів. Результуючі рекомендації для користувачів зберігаються в онлайновий та офлайнових сховищах [6]. Навички мають різну популярність. Для того щоб переконатися, що навичка, на якій базується рекомендований курс бере до уваги унікальні навички користувача, система визначає “зворотну частоту документу” (IDF) для кожної навички.

The soft skills companies need most – and how to learn them

First, let's start with the skills all professionals should learn, regardless of what they do. These are "soft" skills, although in practice they are anything but: 57 percent of leaders say soft skills are more important than hard skills.

1. Leadership

Recommended Courses: Body Language for Leaders, Strategic Thinking, Leading Without Formal Authority

2. Communication

Recommended Courses: Communicating with Confidence, Influencing Others,Giving and Receiving Feedback

3. Collaboration

Recommended Courses: Effective Listening, Building Business Relationships,Finding Your Introvert/Extrovert Balance in the Workplace

Рисунок 1.5 – Згенерований список популярних навичок станом на 2018 рік у мережі LinkedIn

Для підрахунку цієї частоти, користувацький профіль розглядається як вибірка слів, в цьому документі. Потім показник IDF розраховується як $IDF(s) = \log(1 + M/N(s))$, де M це загальна кількість користувачів, а N(s) – це кількість коритсувачів, що мають навичку s у своєму профілі. Навички, що зустрічаються дуже рідко ($N(s) < 100$) не розглядаються. Вектору навичок, що представляє певного користувача, надається вага, яка розраховується за допомогою IDF, перед тим як використовуватися у кінцевому продукті, забезпечуючи механізм

обчислення зваженої подібності. Таким чином, користувач отримає рекомендовані курси на основі власних унікальних навичок.

Перед тим як представити рекомендації користувачу вони проходять додаткову обробку, що використовується для диверсифікації запропонованих курсів, щоб унеможливити присутність декількох схожих курсів, що розташовуються у верхній позиції списку. В якості прикладу, скажімо, що є список курсів, їх чотири, які мають набір навичок, позначених тегами (можуть бути представленні оновлені версії старшого курсу або той самий на іншому рівні складності). В усіх випадках показник схожості всіх чотирьох курсів буде однаковим для учасника, котрий має декілька або всі ті навички, зазначені в його профілі. У випадку, коли найвищий показник співвідношення відповідає цим курсам, то чотири основні рекомендації для користувача будуть чотирма курсами. Проблема включення різноманітності пропозицій являється класичною. В результаті використовують круговий алгоритм, щоб рекомендації були відмінними. Скажімо, у чотирьох курсах однакові навички й вони збігаються з поточними навичками користувача; в цьому випадку об'єднаємо ці курси в одну вибірку й додамо ознаку кожному курсу – скажімо, дату створення. Через це, отримавши навички, для котрих необхідна рекомендація з курсу, з вибірки береться останній курс. Після зроблення цієї вибірки, маємо одну рекомендацію, створюється наступна вибірка з курсів, що має інший перелік навичок. Коли система дійшла до останньої вибірки, й взяла з неї останній курс, повертаємося до першої вибірки та беремо з неї наступний курс. Таким чином система уникає подібних курсів, рекомендованих по порядку.

«YouTube.Music» перед тим як запропонувати музику, створює представлення о смаках користувача. Найпростіший спосіб зробити це – подивитися, які треки на «YouTube.Music» він вже послухав.

Це найважливіша інформація для рекомендаційної системи; з історії прослуховувань можна встановити, яких виконавців і які жанри людина вважає за краще. Однак щоб скласти більш повну картину, непогано ще розуміти, що йому подобається більше, а що – менше.

Для цього ми використовуємо додаткові дані. Одне з джерел таких даних – оцінки «Подобається» і «Не подобається», які ставлять користувачі.

Як правило, люди оцінюють музику, яка викликала у них сильний емоційний відгук – неважливо, позитивний або негативний. Тому оцінки досить точно відображають уподобання людини. Але одних оцінок недостатньо: по-перше, люди ставлять їх далеко не завжди, а по-друге, у шкалі не вистачає півтонів – є тільки або «добре» («Подобається»), або «погано» («Не подобається»).

Тому, крім оцінок і прослуховувань, ми звертаємо увагу і на інші дії користувача: пропуски треків і додавання треків в плейлисти. Всі дії ми поділяємо на позитивні і негативні. Позитивні – прослуховування, оцінка «Подобається», додавання в плейлист – говорять про те, що музика подобається користувачеві, а негативні – пропуск і оцінка «Не подобається» – навпаки. Тому кожній дії ми присвоюємо вага: у оцінки «Подобається» він максимальний, а у пропуску – мінімальний.

Алгоритм аналізує профіль користувача (тобто дані про його музичні уподобання) і передбачає, які треки і виконавці можуть йому сподобатися. Крім того, алгоритм вміє навчатися в режимі реального часу. Кожен раз, коли ви робите нову дію – слухаєте трек або додаєте його в плейлист, – профіль оновлюється, і прогноз будується заново. Це дозволяє швидко підлаштовуватися під смаки і пропонувати музику, яка відповідає сьогоднішньому настрою.

Роблячи прогноз, алгоритм також враховує інформацію про те, як пов'язані один з одним об'єкти з каталогу «YouTube.Music»: треки, альбоми, виконавці, жанри. Завдяки цим даним можна порадити людині нових виконавців у його

улюбленому жанрі. Крім того, система порівнює профілі всіх користувачів «YouTube.Music». Це робиться для того, щоб виявити людей зі схожими музичними вподобаннями: то, що подобається одному, може сподобатися і іншому.

1.6 Постановка задачі

Метою даної роботи є створення десктопного додатку, який полегшить процес вибору мультимедійних матеріалів. Кожна людина має різні вподобання, й постійно зайнята пошуком мультимедії у напрямку, що її цікавить. Небагато веб сервісів можуть виконати усі побажання користувача, не говорячи про десктопні додатки, що змогли б підбирати мультимедію до вподобань людини.

Першим етапом задачі є вибір методу рекомендацій, який краще за всіх підходить для досягнення мети та вирішення задачі, бо велика кількість нині існуючих веб-сайтів використовують різні рекомендаційні методи, для рекомендації контенту своїм користувачам. Існує багато рекомендаційних методів й кожен із них підходить для вирішення певних задач, тому необхідно визначити, який саме метод краще всього підійде для вирішення поставленої задачі.

Другий етап полягає у тому, щоб визначити алгоритм рекомендацій мультимедійних матеріалів у системі та визначити у якому вигляді будуть представлені матеріали й те, як користувачі взаємодіють з ним, тобто, яка буде підходити для вирішення задачі, та визначити, як вона буде використовуватися для генерування рекомендацій.

Третій етап задачі полягає у створенні системи, яка буде пропонувати користувачу відповідно до встановленого режиму (фільми, музика тощо) відповідні матеріали, спираючись на те, які побажання він має. Крім цього рекомендації системи повинні бути адаптивними, які порекомендують

користувачу саме той контент, який потрібен йому даний момент часу. В систему будуть вноситися дані користувачем, які засвідчують його побажання. Враховуючи побажання, система буде рекомендувати мультимедійні матеріали, по проходженню яких користувач матиме змогу оцінювати та доповнювати їх й ця оцінка буде використовуватися при формуванні рекомендацій для інших юзерів. Після генерації рекомендацій, система може запропонувати декілька варіантів, й при виборі його користувачем, враховувати цей вибір для наступної генерації.

Висновки до розділу 1

У першому розділі було визначено, що предметом дослідження являються рекомендаційні системи. За допомогою проведення аналізу існуючих рішень було виявлено основні сучасні рекомендаційні системи та проведений їхній аналіз. Була сформована задача, яка буде проаналізована під час роботи з магістерською дисертацією. Було вказано призначення розробки рекомендаційної системи, цілі та задачі, які повинні бути досягнуті під час роботи з системою рекомендацій мультимедійних матеріалів. Наступний розділі буде присвячений вибору інструментів для розробки.

2 ПРОЕКТУВАННЯ ІНТЕЛЕКТУАЛЬНИХ РЕКОМЕНДАЦІЙНИХ СИСТЕМ МУЛЬТИМЕДІА РЕСУРСІВ

2.1 Опис предметної області

Дослідження систем музичних рекомендацій (MRS) поступово розвиваються [7]. Так само і дослідження глибокого навчання (DL). Незважаючи на свій потенціал, архітектури нейронних мереж все ще дивно рідко використовуються для MRS, хоча кількість відповідних публікацій збільшується.

Історично дослідження MPC виникли у двох окремих спільнотах, тобто системах пошуку музичної інформації (MIP) та системах рекомендацій (PC), з різними напрямками, перспективами та термінологіями.

MIR [8] бере свій початок із бібліотекознавства та обробки сигналів [8]. Тому він протягом тривалого часу сильно зосереджувався на підходах, що ґрунтуються на змісті, де «зміст» відноситься до інформації, витягнутої з фактичного аудіосигналу (порівняно із загальним значенням цього терміна у дослідженнях RS нижче). Дослідження MIR створили захоплюючі інструменти та програми, наприклад, музичні партитури за [9], інтелектуальні інтерфейси перегляду музики [9] або автоматичну класифікацію музики (наприклад, за жанрами [9–10] або афективні категорії, такі як настрій [11, 12]), лише деякі з них. Однак, хоча багато досліджень у MIR стосувалося теми подібності аудіо [13, 14], що є передумовою для створення змістової MRS, напрочуд мало досліджень спільноти MIR було присвячено спеціально музичним рекомендаціям [15].

2.2 Вибір метода колаборативної фільтрації на основі пам’яті

Дослідження систем рекомендацій у музичній сфері, що використовує DL, як правило, використовує глибокі нейронні мережі (DNN) для отримання уявлень про пісні або виконавців (вбудовані чи приховані фактори) з аудіовмісту або текстових метаданих, таких як біографії виконавців або теги, створені користувачами. Ці приховані фактори позиції потім або безпосередньо використовуються у системах CBF, таких як рекомендації найближчого сусіда, інтегруються у підходи до факторизації матриці, або використовуються для створення гібридних систем, найчастіше інтегруючи методи CBF та CF.

Ймовірно, найдавнішою роботою, яка використовує DL для контентної MRS, є Ван ден Оорд та ін., Які приймають згорткову нейронну мережу (CNN) з використанням випрямлених лінійних одиниць (ReLU) і не випадають для представлення кожної пісні за допомогою 50 прихованих факторів з аудіофункцій [16]. В якості вхідних даних вони використовують короткі музичні фрагменти музики, отримані з 7digital5 для треків у наборі даних мільйонів пісень (MSD) [17]. Потім навчання CNN виконується на стиснених журналом спектрограмах Мела (128 діапазонів частот, розмір вікна 23 мс, перекриття вікон 50%), обчислюється з 3-х секундних фрагментів аудіо фрагментів, що діють випадково. Досліджуються два алгоритмічні варіанти: мінімізація середньоквадратичної помилки (MSE) та мінімізація зваженої помилки прогнозування (WPE) як цільової функції. Експерименти проводяться на 382 тис. Пісень та 1 млн користувачів MSD. Підрахунок відтворення для пар "користувач -пісня" перетворюється на двійкові неявні дані зворотного зв'язку (тобто 1, якщо користувач прослухав елемент і незалежно від частоти прослуховування; 0 – інакше). Враховуючи 500 прогнозів на користувача, експерименти показують, що CNN, що використовує MSE як цільову функцію, працює найкраще (AUC:

0,772). Лінійна регресія з використанням загального подання аудіо-слів (векторно-квантовані MFCC) виявилася значно гіршою (AUC: 0,645).

У випадку даної роботи використано, так званий, item-based метод.

У випадку item-based підходу користувач A характеризується об'єктами objsA, які він переглянув або оцінив. Для кожного об'єкта з objsA визначається m об'єктів-сусідів, тобто знаходяться m найбільш схожих об'єктів з точки зору переглядів / оцінок користувачів. При побудові RS для фільмів, m приймає значення від 10 до 30. Всі об'єкти-сусіди об'єднуються в безліч з якого виключаються об'єкти, переглянуті або оцінені користувачем A. A з останків безлічі будується top-k рекомендацій. Таким чином, при item-based підході у створенні рекомендацій беруть участь всі користувачі, яким сподобався той чи інший об'єкт з objsA.

Дана методологія забезпечує максимально точні рекомендації для кожного користувача, які базуються виключно на його персональних вподобаннях і поведінці.

2.3 Вибір метода колаборативної фільтрації на основі моделі

У цьому підході моделі розробляються з використанням різних методів інтелектуального аналізу даних, алгоритмів машинного навчання для прогнозування рейтингу користувачів не оцінених елементів. Існує багато алгоритмів CF на основі моделей. Байєсові мережі, моделі кластеризації, латентні семантичні моделі, такі як розкладання сингулярних значень, ймовірнісний латентний семантичний аналіз, множинний мультиплікативний коефіцієнт, прихований розподіл Діріхле та моделі марковського процесу прийняття рішень [18].

Завдяки цьому підходу методи зменшення розмірності здебільшого використовуються як доповнююча техніка для покращення надійності та точності підходу на основі пам’яті. У цьому сенсі такі методи, як розкладання сингулярних значень, аналіз головних компонентів, відомі як моделі прихованих факторів, стискають матрицю елементів користувача у низьковимірне представлення з точки зору прихованих факторів. Однією з переваг використання цього підходу є те, що замість матриці високого розміру, що містить велику кількість відсутніх значень, ми будемо мати справу з набагато меншою матрицею в просторі нижчих розмірів. Скорочена презентація може бути використана для алгоритмів сусідства на основі користувачів або елементів, представлених у попередньому розділі. У цієї парадигми є кілька переваг. Він краще справляється з розрідженістю оригінальної матриці, ніж на основі пам'яті. Порівняння подібності в отриманій матриці набагато більш масштабоване, особливо при роботі з великими розрідженими наборами даних [19].

В даному випадку використано власну варіацію алгоритму k-means. В загальному вигляді k-means — це метод векторного квантування, спочатку з обробки сигналу, який має на меті розділити n спостережень на k кластерів, в яких кожне спостереження належить до кластера з найближчим середнім

значенням (центри кластера або центроїд кластера), що служить прототипом кластер. Це призводить до поділу простору даних на осередки Вороного. k-означає кластеризація мінімізує дисперсії в межах кластера (квадратні евклідові відстані), але не регулярні евклідові відстані, що було б складнішою проблемою Вебера: середнє оптимізує квадратичні помилки, тоді як лише геометрична медіана мінімізує евклідові відстані. Наприклад, кращі євклідові рішення можна знайти за допомогою k-медіан та k-медоїдів.

Проблема обчислювальна (NP-важка); проте ефективні евристичні алгоритми швидко сходяться до локального оптимуму. Зазвичай вони подібні до алгоритму максимізації очікувань для сумішей гаусових розподілів за допомогою підходу ітераційного уточнення, що використовується як для k-середніх, так і для моделювання суміші Гауса. Вони обидва використовують кластерні центри для моделювання даних; однак, кластеризація k-засобів має тенденцію знаходити кластери порівнянного просторового розміру, тоді як модель суміші Гауса дозволяє кластерам мати різну форму.

Алгоритм некерованого k-засобу має слабке відношення до класифікатора k-найближчого сусіда, популярної методики машинного навчання під наглядом для класифікації, яку часто плутають з k-засобами через назву. Застосування класифікатора 1 найближчого сусіда до центрів кластерів, отриманих за допомогою k-засобів, класифікує нові дані у існуючі кластери. Це відоме як найближчий центроїдний класифікатор або алгоритм Роккіо.

У даному випадку створення власної реалізації інтелектуальних рекомендаційних систем мультимедіа ресурсів даний алгоритм реалізований в наступному вигляді: у якості центроїда приймаються 2 найчастіших варіанта по кожному фактору (характеристиці) музичного твору і на базі отриманих центроїдів усі наявні музичні твори діляться на 3 окремих кластери, 2 з яких потрапляють до фінального списку рекомендацій, а 3-й вважається кластером «непідходящих під вподобання користувача» результатів.

Висновки до розділу 2

У другому розділі було визначено, що робота буде базуватися на моделі Item-based, а для безпосереднього пошуку рекомендацій буде використано власну спрощену реалізацію алгоритму k-means для пошуку найближчих сусідів центроїда, створеного на базі вподобань користувача. Окрім цього було проведено короткий опис предметної області, який дав загальне розуміння специфіки теми. Наступний розділі буде присвячений проектуванню і розробці власної програмної реалізації інтелектуальної рекомендаційної системи мультимедіа ресурсів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНИХ РЕКОМЕНДАЦІЙНИХ СИСТЕМ МУЛЬТИМЕДІА РЕСУРСІВ

3.1 Опис бази даних

База даних проекту складається з однієї таблиці, яка містить інформацію щодо музичних композицій і їх характеристик, за якими ведеться підбір рекомендацій. Структура таблиці бази даних зображена на рисунку 3.1.

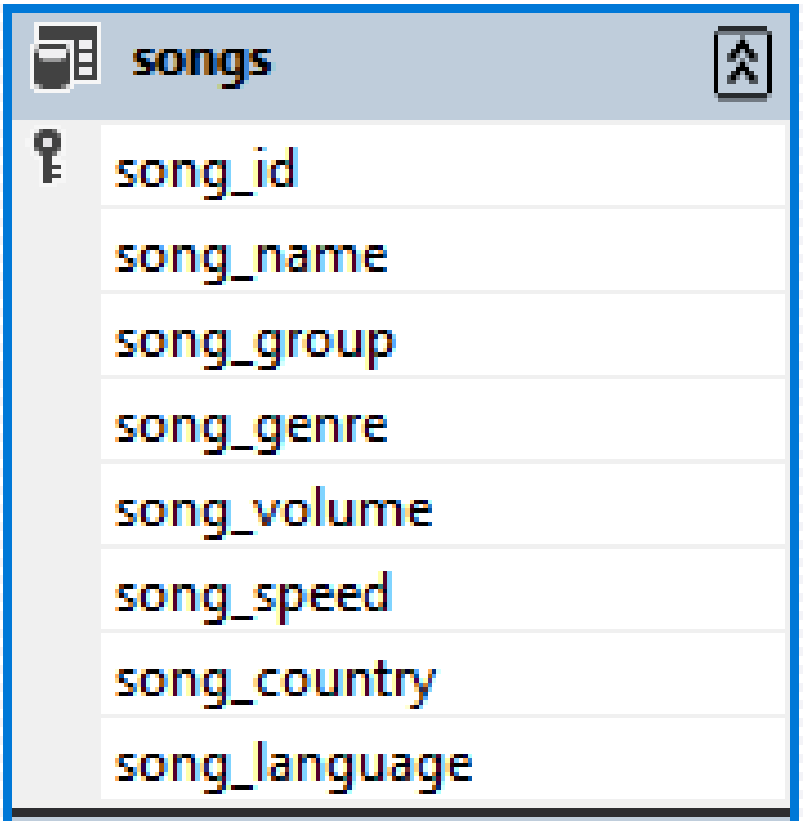

Рисунок 3.1 – Структура таблиці «songs»

Дана таблиця містить наступні поля:

– поле ідентифікатора композиції;
– поле назви композиції;
– поле назви групи, яка виконує дану композицію;

- поле назви жанру, до якого належить композиція;
- поле гучності композиції;
- поле швидкості виконання композиції;
- поле країни походження групи, яка виконує композицію;
- поле мови, на якій виконується дана композиція.

Дані характеристики є вичерпними для будь-якої композиції і їх кількість і глибина опису композиції забезпечує широкі можливості для підвищення точності підбору рекомендацій в ході роботи інтелектуальних рекомендаційних систем мультимедіа ресурсів на основі даних характеристик.

3.2. Мова програмування Python

Python — це універсальна мова програмування, яка розшифровується на значному рівні. Розуміння плану Python підкреслює узгодженість коду, використовуючи величезний простір. Його мовна конструкція та підхід до впорядкованості елементів призначені для того, щоб допомогти розробникам у створенні ідеального, узгодженого коду для все більших підприємств.

Python поступово збирає сміття. Він підтримує декілька ідеальних моделей програмування, включаючи організоване (враховуючи процедурне), об’єктне та корисне програмування. Оскільки Python має загальну стандартну бібліотеку, його зазвичай зображують як мову, що розрахована на батареї.

Будучи наступником мови програмування ABC, Гвідо ван Россум почав досліджувати Python в останній половині 1980-х років і вперше представив мову під назвою Python 0.9.0 у 1991 році [22]. Python 2.0 було представлено в 2000 році, представляючи нові моменти. , наприклад, сприйняття списків і структура асортименту сміття з підрахунком посилань, і була вилучена у варіанті 2.7.18 у 2020 році [23]. Python 3.0 був представлений у 2008 році та є значною модифікацією мови. Мова не є повністю життєздатною у зворотному напрямку, і більшість коду Python 2 не може працювати належним чином у Python 3.

Python завжди був одним із найвідоміших діалектів програмування.

Python було створено в останній половині 1980-х років Гвідо ван Россумом з нідерландського Centrum Wiskunde and Informatica (CWI). Це заміна мови програмування ABC, мотивованої SETC, яка може мати справу з особливими випадками та взаємодіяти з робочою структурою одноклітинного створіння. Його реалізація почалася в грудні 1989 року. Будучи провідним дизайнером підприємства, Ван Россум опікувався цим заходом до 12 липня 2018 року, коли

він оголосив про свою «вічну подію» з титулом «Хороший деспот життя» Python. Це відображено у своїй довгій історії групою людей Python. Зобов'язання бути головним адміністратором завдань. Наразі він твердо стоїть на ініціативній позиції як особа з адміністративної ради з п'яти осіб. У січні 2019 року дизайнери динамічного центру Python обрали Бретта Кеннона, Скретча Коглана, Баррі Варшау, Гімна Віллінга та Ван Россума як осіб із ради з п'яти осіб. Відтоді Гвідо ван Россум відкликав свою заявку на участь у правлінні 2020 року.

Python 2.0 було представлено 16 жовтня 2000 року з багатьма основними новинками, включаючи коло, що розпізнає спеціаліста зі сміття, і підтримку Unicode.

Python 3.0 було випущено 3 грудня 2008 року. Це суттєва корекція мови, яка не є повністю життєздатною для іншої сторони. Значну кількість центральних елементів було перенесено на Python 2.6.x і 2.7.x. Розповсюдження Python 3 містить утиліту 2to3, яка відповідно (безсумнівно, дещо) повністю змінює код Python 2 на Python 3.

Спочатку Python 2.7 було завершено у 2015 році, а потім перенесено на 2020 рік через занепокоєння, що багато існуючого коду не можна буде швидко перенести на Python 3. Більших виправлень безпеки та інших удосконалень не буде. Наближаючись до кінця свого життєвого циклу, Python 2 лише підтримує Python 3.6.x і пізніших версій.

Оскільки всі форми Python мають проблеми з безпекою, Python 3.9.2 і 3.8.8 були придушені, що призвело до ймовірного віддаленого виконання коду та пошкодження веб-резерву.

Python — це мова програмування з різними світоглядами. Він повністю підтримує об'єктно-організоване програмування та організоване програмування, і велика кількість його елементів підтримує корисне програмування та програмування з перспективою (враховуючи план метапрограми та метаоб'єкт

(техніка зачарування)). Розширення підтримує численні різні ідеальні моделі, включаючи контрактний план і обґрунтування програмування.

Python використовує динамічне створення та поєднання спеціалістів із підрахунку посилань і сміття. Спеціаліст зі сміття характеризує коло для спостереження за пам'яттю. Він також має ціль динамічного імені (пізнє обмеження), щоб зв’язати імена техніки та змінних під час виконання.

Python покликаний запропонувати деяку допомогу для утилітарного програмування на основі Drawl. Використання можливостей відсіву, показу та зменшення; фіксується розуміння твірних, посилань на слова, множин і артикуляцій. Стандартна бібліотека має два модулі (itertools і functools) для виконання утилітарних пристроїв, отриманих від Haskell і Standard ML.

Основні принципи мови викладені в документі Zen of Python (PEP 20), який включає такі сентенції:

- красиво-краще, ніж потворно;
- явне краще, ніж неявне;
- просте – краще, ніж складне;
- комплекс краще, ніж комплекс;
- розрахунок читабельності.

Python не має своїх елементів, інтегрованих у весь центр, але призначений як винятково розширюваний (за допомогою модулів). Ця консервативна виміряна якість робить його особливо відомим як метод додавання програмованих точок підключення до існуючих програм. Бачення Ван Россума маленьких центральних діалектів, величезних стандартних бібліотек і ефективно розширюваних інтерпретаторів походить від його розчарування ABC, який виступає за протилежну методологію. Python планує покращити та зменшити хаос пунктуації та структури речень, дозволивши інженерам вибирати свої стратегії кодування. Зовсім не схожий на дотепність Perl: «Існує більше одного

методу, щоб це сталося», Python охоплює міркування плану: «Повинен бути один — в ідеалі — лише один — чіткий метод, щоб це зробити» [23]. Алекс Мартеллі, співробітник Python Programming Establishment і автор книги про Python, висловився: «Зображення чогось як «геніального» не розглядається як похвала культурі Python».

Інженери Python швидко тримаються подалі від передчасного прогресу та відмовляються виправляти небазові частини еталонного виконання CPython. Ці виправлення завдають шкоди чіткості, отже незначно розвиваючи швидкість. У той момент, коли швидкість є значною, розробники програмного забезпечення Python можуть перенести можливості базування часу на доповнення, написані діалектами, як-от C, або використовувати компілятор PyPy, що підтримує час. Також доступний Cython, який змінює вміст Python на C і надає рішення щодо інтерфейсу програмування на рівні C безпосередньо до транслятора Python.

Важлива мета дизайнерів Python — забезпечити клієнту блаженство. Це відображається на користь мови (визнання англійської сатиричної групи Monty Python), а в деяких випадках і в захоплюючих навчальних вправах і довідкових стратегіях, наприклад, стандартні foo і такти.

Клієнтів і шанувальників Python, особливо тих, хто вважається кваліфікованим або досвідченим, часто називають Pythonistas. Python має бути простою для розуміння мовою. Його конфігурація зовні не вводить в оману, і він часто використовує англійські крилаті фрази, тоді як різні діалекти використовують знаки акцентуації. На відміну від багатьох різних діалектів, він не використовує хвилястих опор для ізоляції перешкод і дозволяє крапки з комою після пояснень, але крапки з комою використовуються рідко. Він має менше синтаксичних винятків і екстраординарних випадків, ніж C або Pascal.

Python використовує пробіли, а не хвилясті опори чи ключові слова для ізоляції блоків. Простір буває після конкретних адміністраторів; Зменшення відступу майже гарантує загибель поточного блоку. Таким чином, візуальний

дизайн програми точно відображає семантичну конструкцію програми. Цей елемент час від часу називають «зовнішнім» правилом, і кілька різних діалектів також мають цей стандарт, однак у багатьох діалектах пробіл не має семантичного значення. Рекомендований розмір місця – чотири місця.Рекомендований розмір відступу – чотири пробіли.

Оператори Python включають (серед іншого):

– використовуйте знак рівності = оператор присвоєння;

– оператор if, який використовується разом з else та elif (скорочення від else–if) для умовного виконання блоку коду;

– оператор for, який фіксує кожен елемент у локальній змінній для використання вкладеним блоком для перегляду об'єкта, що ітерацію;

– поки оператор, доки умова є істинним, виконується блок коду;

– оператор try, який дозволяє захоплювати та обробляти витяги, що містяться у вкладених блоках коду (крім речень); він також гарантує, що незалежно від того, як блок виходить, код очищення в блоці завжди буде виконаний в кінцевому підсумку;

– оператор підвищення застосовується для отримання зазначеного винятку або повторного підняття спійманого винятку;

– оператори класів, які виконують блоки коду та приєднують свій локальний простір імен до класу для використання в об'єктно–орієнтованому програмуванні;

– оператор Def, який визначає функцію або метод;

– оператор with з Python 2.5, випущений у вересні 2006 р. [23], який охоплює блок коду в диспетчері контексту (наприклад, отримання блокування перед запуском кодового блоку, а потім звільнення блокування або відкриття файлу. Потім закрийте його), дозволяють поведінку подібних ресурсів ініціалізувати (RAII) та замінюють загальне використання try / final;

– оператор Break, вихід із циклу;

– оператор continue пропускає цю ітерацію та переходить до наступного пункту;

– оператор del видалить змінну, що означає, що посилання на ім'я значення буде видалено, а спроби використання цієї змінної призведуть до помилки. Ви можете перепризначити видалені змінні;

– виконувати функції NOP через операторів. Синтаксично це необхідно для створення порожнього блоку коду;

– заява про затвердження, що використовується для перевірки умов, що застосовуються під час налагодження;

– оператор Yield, який повертає значення з функції генератора. У Python 2.5 yield також є оператором. Ця форма використовується для реалізації спільного плану;

– оператор return використовується для повернення значень із функцій;

– оператор import використовується для імпорту модулів, функції чи змінні яких можна використовувати в поточній програмі. Існує три способи використання імпорту: імпорт <ім'я модуля> [як <псевдонім>] або <ім'я модуля> імпорт * або <ім'я модуля> для імпорту <визначення 1> [як <псевдонім 1>], <визначення 2 > [Як <псевдонім 2>].

Роль оператора присвоєння (=) полягає у прив'язуванні імені як посилання на окремий динамічно розподілений об'єкт. Потім ви можете будь-коли відхилити змінну до будь-якого об’єкта. У Python ім'я змінної є спільним власником посилання, і з ним не пов’язаний фіксований тип даних. Однак у цей час змінна посилається на об’єкт із типом. Це називається динамічним набором тексту і на відміну від статично набраних мов програмування, де кожна змінна може містити лише певний тип значення.

Python не підтримує оптимізацію хвостових викликів або першокласні розширення, і за словами Гвідо ван Россума, це ніколи не буде. Однак краща підтримка функцій, подібних до корутину, передбачена в 2.5, розширюючи тим самим генератор Python. Ледачий ітератор має максимум 2,5 генератора. Інформація передається в одному напрямку від генератора. За допомогою Python 2.5 ви можете передавати інформацію назад до функції генератора, тоді як за допомогою Python 3.3 ви можете передавати інформацію через кілька рівнів стека.

Деякі вирази Python подібні до таких, що зустрічаються в таких мовах, як C та Java, а інші – ні:

– додавання, віднімання та множення однакові, але поведінка ділення інша. У Python є два типи розділів. Вони є базовим діленням (або цілочисельним поділом) // та плаваючою комою / діленням. Python також використовує оператори ** для просування;

– у Python 3.5 було введено нове твердження @infix. Він використовується бібліотеками, такими як NumPy, для множення матриць;

– у Python 3.8 введено синтаксис: =, який називається "моржовим оператором". Він присвоює значення змінним як частину більшого виразу; [24]

– у Python == порівнює з Java за значенням, Java порівнює числові значення за значенням, а об'єкти порівнює за посиланням. (Порівняння значень об'єктів у Java може бути здійснено за допомогою методу equals ().) Оператор is is is is може бути використаний для порівняння ідентифікаторів об'єктів (порівняння посилань). У Python можна порівняти порівняння, наприклад <= b <= c;

– python використовує слово і, або не для своїх булевих операторів, а не для символів &&, ||! Використовується в Java та C.Python має тип виразу, що

називається розумінням списку, а також більш загальний вираз, який називається генераторським виразом;

– анонімні функції реалізовані з використанням лямбда-виразів, однак вони обмежені тим фактом, що тіло може бути лише виразом;

– якщо с інакше у, умовний вираз у Python буде записаний як x (відмінний від порядку операнда оператора c? X: y), що є однаковим у багатьох інших мовах;

– python розрізняє списки та кортежі. Список записаний як [25], він був змінений і не може використовуватися як ключ словника (ключ словника повинен бути постійним у Python). Кортеж записується як (1, 2, 3) і є незмінним, тому, поки всі елементи кортежу є незмінними, він може використовуватися як ключ словника. Оператор + може використовуватися для об'єднання двох кортежів, що безпосередньо не змінює їх вміст, але створює новий кортеж, що містить елементи двох передбачених кортежів. Отже, враховуючи, що змінна t спочатку дорівнює (1,2,3), коли виконується t = t + (4, 5), спочатку обчисліть t + (4, 5), щоб отримати (1,2, 3) . 3, 4, 5), а потім призначити його назад до t, тим самим ефективно "змінюючи вміст t", одночасно відповідаючи інваріантній природі об'єкта кортежу [26]. У чіткому контексті кортежі не потребують дужок;

– python має функцію декомпресії [27] послідовності, де кілька виразів (кожен вираз обчислює весь вміст, який може бути призначений) (змінні, атрибути запису тощо) мають однакову форму з текстом, що становить кортеж, і, як правило, розміщуються зліва Одна сторона знака рівності у твердженні про присвоєння. Оператор очікує ітерабельний об’єкт праворуч від знака рівності. При сортуванні об’єкт [28] видасть таку ж кількість значень, що й даний вираз, і відфільтрує їх, присвоюючи кожне створене значення лівій стороні відповідного виразу;

– python має оператор% "рядковий формат". Це схоже на рядок printf у C, наприклад, "спам =% s яйце =% d"% ("бла", 2) оцінюється як "спам = бла–яйця = 2". У Python 3 та 2.6+ метод str () формату () доповнив це, наприклад "спам = {0} яйце = {1}". Формат ("бла", 2). Python 3.6 додав "f–рядки": blah = "blah"; яйця = 2; f'spam = {blah} яйця = {яйця};

– рядки в Python можна поєднувати шляхом "додавання" (те саме твердження, що і додавання цілих чи значень з плаваючою комою). Наприклад, "спам" + "яйце" повертає "спамера". Навіть якщо ваш рядок містить цифри, вони все одно будуть додані як рядки замість цілих чисел. Наприклад, "2" + "2" повертає "22".

Python має різні типи рядкових літералів:

– рядки, укладені в одинарні або подвійні лапки. На відміну від Unix Shell, мови Perl та Perl впливають на Perl, а одинарні та подвійні лапки працюють однаково. Обидва типи рядків використовують зворотні скісні риски (\) як вихідні символи. Інтерполяція рядків стала "відформатованим рядковим літералом" у Python 3.6.Рядки з подвійними лапками, які починаються та закінчуються серією з трьох одинарних або подвійних лапок. Вони можуть охоплювати кілька рядків і функціонувати, як тут документи в оболонках, Perl і Ruby;

– початковий рядок, представлений префіксом рядкового літералу перед r. Ескейп-послідовності не інтерпретуються; тому необроблені рядки корисні в місцях, де поширені зворотні скісні риски літер, наприклад, регулярні вирази та шляхи у стилі Windows. Порівняйте "@ –citation" у C #.

На відміну від таких мов, як Normal Stutter, Plan або Ruby, Python робить розумну кваліфікацію серед артикуляцій і проголошень. Це спонукає до дублювання певних можливостей.

Python зазвичай використовується в створених людьми свідомості та проектах ШІ з бібліотеками, наприклад TensorFlow, Keras, Pytorch і Scikit-learn. Python зазвичай використовується для обробки звичайної мови як мова попередньої обробки з особливим дизайном, базовою мовною структурою та офісами обробки тексту високого рівня. Python було ефективно застосовано в багатьох елементах попереднього програмування, включаючи обмежене програмування компонентів, наприклад, Abaqus, 3D-параметричні моделі, наприклад, FreeCAD, 3ds Max, Blender, Film 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage , розширені візуалізації Nuke, аранжувальник, програми для двовимірних зображень (наприклад, GIMP, Inkscape, Scribus і Paint Shop Master) і програми пояснення (наприклад, Creator і Gathering). Налагоджувач GNU використовує Python як гідний принтер для показу складних конструкцій, таких як відсіки C++.

Esri стверджує, що Python є найбільш ідеальним рішенням для попереднього налаштування в ArcGIS. Він також використовувався в кількох комп'ютерних іграх і був першим із трьох доступних діалектів програмування, які використовував Google Application Motor, два інших — Java і Go.

Багато робочих фреймворків використовують Python як стандартну частину. Він супроводжує більшість версій Linux, AmigaOS 4 (використовує Python 2.7), FreeBSD (у комплекті), NetBSD, OpenBSD (у комплекті) та macOS, і його можна використовувати з рядка замовлення (термінал). Численні розповсюдження Linux використовують інсталятори, написані на Python: Ubuntu використовує інсталятор Omnipresence, тоді як Red Cap Linux і Fedora використовують інсталятор Boa constrictor. Gentoo Linux використовує Python у своєму пакеті Portage, інфраструктурі плати.

Python зазвичай використовується для захисту даних, у тому числі для вдосконалення експлойтів.

Переважна більшість програм Sugar PC XO, створених Sugar Labs, написані на Python. У проекті одноплатної програми для ПК Raspberry Pi використано Python як основну клієнтську мову програмування.

LibreOffice містить Python і планує замінити Java на Python. Його постачальник попередньої підготовки Python є важливим компонентом у варіанті 4.0 7 лютого 2013 року.

Виходячи з обсягу та потужних особливостей мови Python, описаних вище, можна зробити висновок, що вона універсальна. Ось чому для виконання цієї роботи було обрано саме цю мову програмування.

3.3 PyCharm

PyCharm – це інтегроване середовище розробки (IDE), що використовується в комп'ютерному програмуванні, зокрема для мови Python. Він розроблений чеською компанією JetBrains. Він забезпечує аналіз коду, графічний налагоджувач, інтегрований тестер одиниць, інтеграцію із системами контролю версій (VCSes), а також підтримує веб-розробку з Django, а також аналіз даних з Anaconda.

PyCharm є кросплатформним, з версіями для Windows, macOS та Linux. Видання Community випускається під ліцензією Apache, а також існує Professional Edition з додатковими функціями – випущено під власною ліцензією.

Бета-версія була випущена в липні 2010 року, а версія 1.0 з’явилася через 3 місяці. Версія 2.0 була випущена 13 грудня 2011 року, версія 3.0 – 24 вересня 2013 року, а версія 4.0 – 19 листопада 2014 року

PyCharm Community Edition, версія з відкритим кодом PyCharm, стала доступною 22 жовтня 2013 р.

3.4 Структура програми

З точки зору структурної будови — програма складається з 8 класів, кожний з яких відповідає за окрему частину функціоналу програми.

Для представлення структури програми в сприйнятому вигляді використано діаграму класів (рисунок 3.2).

У програмній інженерії діаграма класів Unified Modeling Language (UML) – це статична структурна діаграма, яка описує структуру системи та відображає системні класи, їх атрибути, операції (або методи) та взаємозв'язки між об'єктами.

Діаграми класів – це основні будівельні блоки об’єктно-орієнтованого моделювання. Він використовується для загального концептуального моделювання структури програми та детального моделювання для перетворення моделі в програмний код. Діаграми класів також можна використовувати для моделювання даних. Класи на схемі класів представляють основні елементи, взаємодії в програмі та класи програмування.

На схемі класи представлені вікнами, які містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква – мала.

При розробці системи виявлення багатьох класів та згрупування їх у схему класів це допомагає визначити статичний зв’язок між ними. При детальному

моделюванні категорії концептуальних об’єктів зазвичай поділяються на кілька підкатегорій.

Залежність – це семантичний зв’язок між залежними елементами та незалежними елементами моделі. Якщо зміна визначення одного елемента (сервера або цілі) може змінити іншого (клієнта або джерела), воно існує між двома елементами. Ця асоціація однобічна. Актуальність представлена пунктирною стрілкою, де відкрита стрілка вказує від замовника до постачальника.

Для подальшого опису поведінки системи ці діаграми класів можуть бути доповнені діаграмами стану або автоматами стану UML.

Асоціація пропонує безліч посилань. Бінарні асоціації (з двома кінцями) зазвичай виражаються у вигляді рядків. Асоціації можуть пов’язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціацію можна назвати, а кінець асоціації можна змінити, використовуючи такі атрибути, як ім'я ролі, індикатор атрибутів, кратність та видимість.

Існує чотири різні типи асоціацій: двосторонні, односторонні, агрегаційні (включаючи комбіновану агрегацію) та рефлексивні. Найпоширенішими є двосторонні та односторонні асоціації.

Наприклад, клас польоту асоціюється з класом двостороннього літака. Асоціація – це статичний зв’язок, що використовується спільно між об’єктами двох класів.

Агрегація є варіантом взаємозв'язку "має"; агрегація є більш конкретною, ніж асоціація. Це асоціація, яка представляє частину або частину відносин. Як показано на малюнку, професор "має" клас для викладання. Як асоціація, агрегація може бути названа та мати ті ж модифікації, що і асоціація. Однак агрегація не може включати більше двох категорій; вона повинна бути бінарною асоціацією. Крім того, майже немає різниці між агрегуванням та асоціацією в

процесі реалізації, а планування може повністю пропустити взаємозв'язок агрегування. [26]

Коли клас є колекцією або контейнером інших класів, відбувається агрегування, але розміщений клас не сильно залежить від життєвого циклу контейнера. Коли контейнер знищений, вміст контейнера все ще існує.

В UML він зображений графічно у вигляді порожнистого діаманта на хост-–класі, що з'єднує його з рядком хост-класу. Агрегація – це семантично розширений об’єкт, який розглядається як одиниця в багатьох операціях, хоча фізично складається з декількох менших об’єктів.

Наприклад: бібліотека та студенти. Тут студенти можуть існувати без бібліотеки, і між студентом і бібліотекою існує ціле.

Це показує, що один із двох пов’язаних класів (підкласів) вважається спеціалізацією іншого (супертип), а суперклас - узагальненням підкласів. На практиці це означає, що будь-який екземпляр підтипу є також екземпляром супертипу. Типове дерево узагальнення цієї форми можна знайти в біологічній систематиці: людина – це підгрупа мавп, мавпа – підгрупа ссавців тощо. Цей зв’язок найкраще зрозуміти з фрази «А – це В» (люди – це ссавці, а ссавці – тварини).

Графічне зображення узагальнення UML – це форма порожнистого трикутника в кінці лінії суперкласу (або дерева ліній), що зв’язує його з одним або кількома підтипами.

Відносини узагальнення також називають спадкоємством або відносинами "є".

Суперклас (базовий клас) у відносинах узагальнення також називають "батьківським класом", суперкласом, базовим класом або базовим типом.

Підтипи у відносинах спеціалізації також називаються "дочірніми" підкласами, похідними класами, похідними типами, успадкованими класами або успадкованими типами.

Зверніть увагу, що ці стосунки повністю відрізняються від біологічних стосунків між батьком та дитиною: використання цих термінів дуже поширене, але може ввести в оману.

A – це тип B

Наприклад, "дуб – це дерево", "машина – транспортний засіб"

Короткий зміст може відображатися лише на схемі класів та діаграмі використання.

При моделюванні UML взаємозв'язок реалізації – це взаємозв'язок між двома елементами моделі, де один елемент моделі (замовник) реалізує (реалізує або виконує) поведінку, визначену іншим елементом (постачальником) моделі.

Графічне представлення реалізації UML – це порожнистий трикутник, який підключений до кінця пунктирного інтерфейсу (або дерева рядків) одного або кількох реалізаторів. Проста стрілка використовується в кінці пунктирного інтерфейсу, щоб підключити її до користувача. Схема компонентів використовує графічну умову «кульова розетка» (продавець розміщує кульку або льодяник, а користувач відображає кульову розетку). Реалізація може відображатися лише в режимі класу або компонента. Реалізація – це взаємозв'язок між класами, інтерфейсами, компонентами та пакетами, що з'єднують елементи замовника з елементами постачальника. Зв'язок реалізації між класом / компонентом та інтерфейсом вказує на те, що клас / компонент реалізує операцію.

Залежність – це слабша форма спілкування, яка вказує на те, що один клас залежить від іншого класу, оскільки він використовує його в певний момент часу. Якщо незалежний клас залежить від змінної параметра або локальної змінної методу, то один клас залежить від іншого класу. Це відрізняється від асоціації, де атрибут залежного класу є екземпляром незалежного класу. Іноді відносини між цими двома класами слабкі. Вони взагалі не реалізовані зі змінними-членами. Натомість вони можуть бути реалізовані як параметри функцій-членів.

Інші стосунки часто описують як "А має В" (у самки кота є кошеня і кошка).

Представлення асоціації UML – це лінія, що з'єднує два пов'язаних класи. У кожному кінці рядка є зайві символи. Наприклад, ми можемо використовувати кінчик стрілки, щоб вказати, що кінчик видно з хвоста стрілки. Ми можемо вказати право власності, поставивши кульку на цьому кінці, яку роль відіграє елемент, і вказати ім'я ролі та кілька екземплярів сутності (з іншого боку, діапазон об'єктів, що беруть участь в асоціації).

Довгострокова інформація, що обробляється системою моделювання сутності, і іноді поведінка, пов'язана з інформацією, моделюється. Вони не повинні розпізнаватися як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Або їх можна намалювати як звичайні заняття з "суттєвим" стереотипом щодо назви класу.

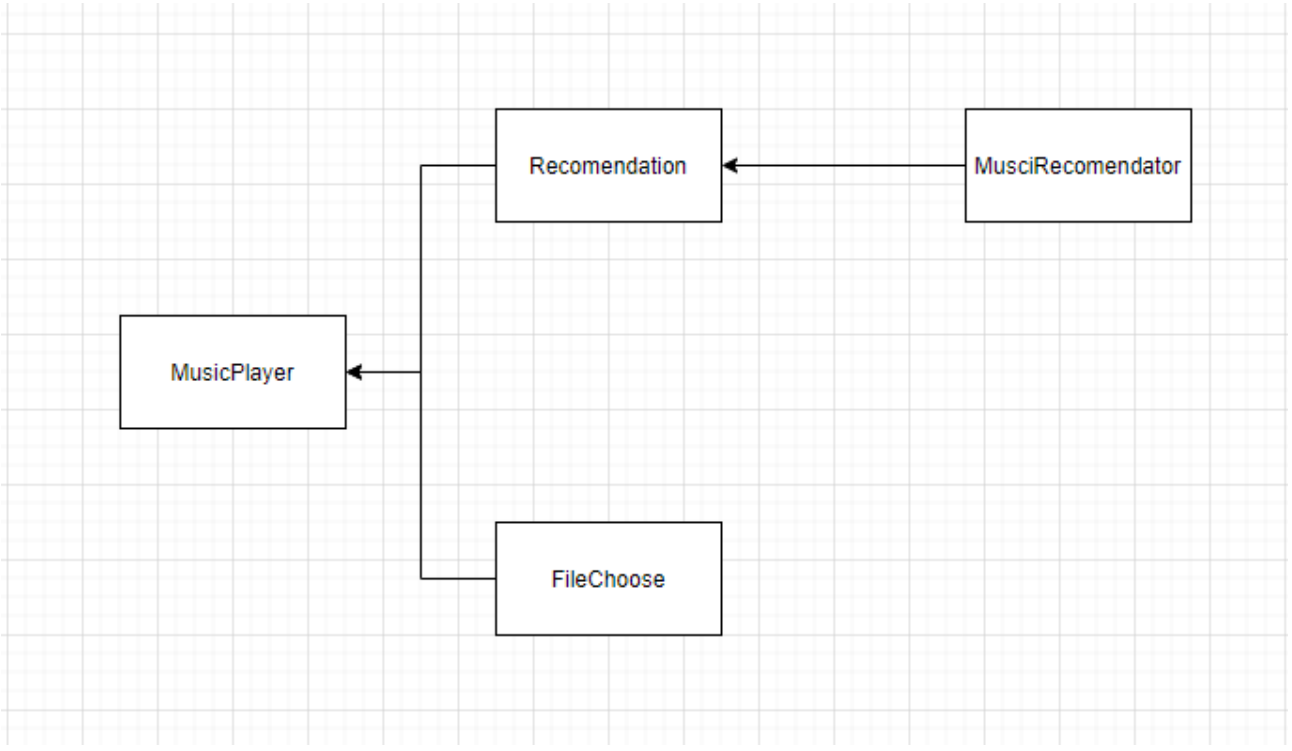

Рисунок 3.2 – Діаграма класів програмного продукту

3.5 Опис інтерфейсу програми

Графічний інтерфейс системи було створено з використанням Python-бібліотеки Kivy, яка розрахована не тільки на зручне створення графічних інтерфейсів, а й на забезпечення кросплатформенності додатків.

Графічний інтерфейс користувача складається з одного графічного вікна і двох поп-апів (спливаючих вікон), які з'являються за необхдності.

На рисунку 3.2 показано інтерфейс головного вікна користувача.

Рисунок 3.2 — Головне вікно програми

На рисунку 3.3 зображено інтерфейс поп-апу для вибору директорії для прослуховування.

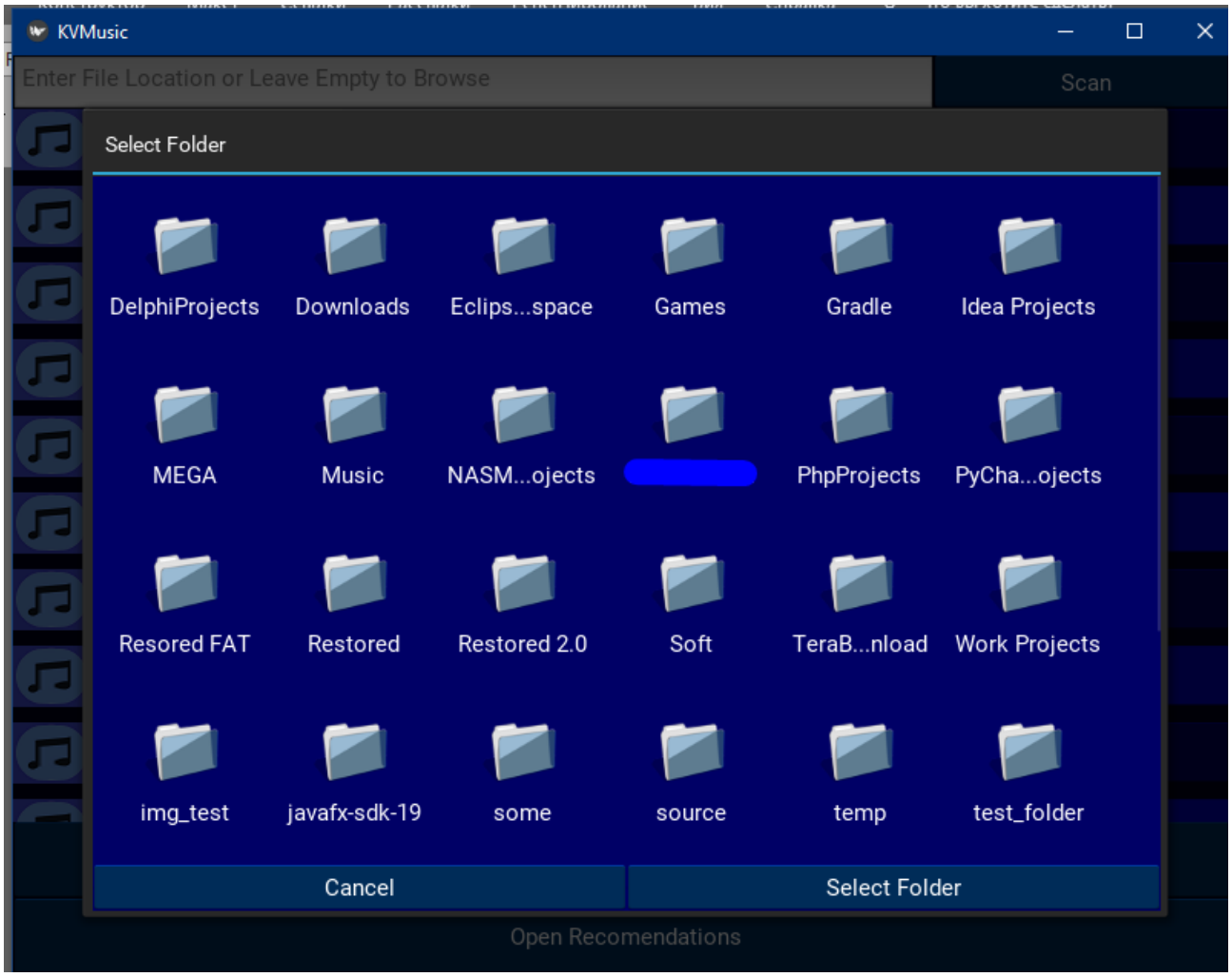

Рисунок 3.3 — Поп-ап вибору директорії

На рисунку 3.4 зображено поп-ап з рекомендаціями для користувача.

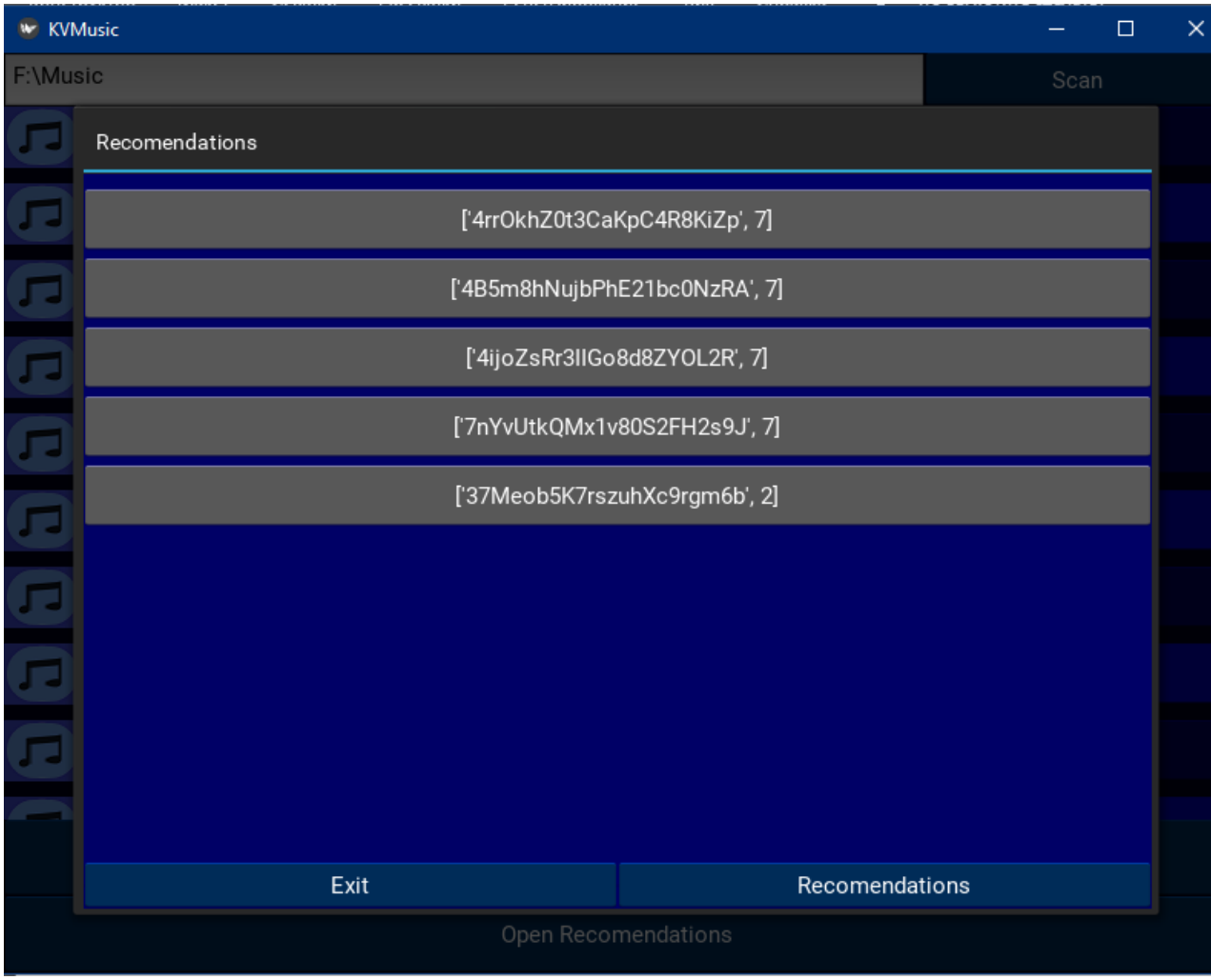

Рисунок 3.4 — Поп-ап з рекомендаціями

3.6 Опис основних алгоритмів програми

Основним алгоритмом системи є створення нейронної мережі для підбору рекомендацій для користувачів.

Система кожного разу кластеризує отримані на вхід дані і алгоритмом k-means підбирає схожі композиції.

Кластеризація k-середніх — це метод векторного квантування, спочатку з обробки сигналів, який має на меті розділити n спостережень на k кластерів, у яких кожне спостереження належить до кластера з найближчим середнім (центри кластерів або центроїд кластерів), що служить прототипом кластер. Це призводить до поділу простору даних на комірки Вороного. Кластеризація k-середніх мінімізує дисперсії в межах кластера (квадрати евклідових відстаней), але не регулярні евклідові відстані, що було б складнішою проблемою Вебера: середнє оптимізує квадратичні помилки, тоді як лише геометрична медіана мінімізує евклідові відстані. Наприклад, кращі евклідові рішення можна знайти за допомогою k-медіан і k-медоїдів.

Проблема обчислювально складна (NP-складна); однак ефективні евристичні алгоритми швидко збігаються до локального оптимуму. Зазвичай вони подібні до алгоритму очікування-максимізації для сумішей гаусових розподілів через ітераційний підхід уточнення, який використовується як для k-середніх, так і для моделювання сумішей за Гаусом. Вони обидва використовують центри кластерів для моделювання даних; однак кластеризація k-середніх має тенденцію знайти кластери порівнянного просторового розміру, тоді як модель суміші Гауса дозволяє кластерам мати різні форми.

Алгоритм неконтрольованих k-середніх має вільний зв’язок із класифікатором k-найближчих сусідів, популярним методом контрольованого машинного навчання для класифікації, який часто плутають із k-середніми через назву. Застосування класифікатора 1 найближчого сусіда до центрів кластерів,

отриманих за допомогою k-середніх, класифікує нові дані в існуючі кластери. Це відомо як класифікатор найближчого центроїда або алгоритм Роккіо.

Нижче представлено програмний код реалізації алгоритму.

```
      def music_recommender(userPreferences):
          raw_data = pd.read_csv('genres_v2.csv', dtype={'song_name': 'str'})
          print(raw_data.shape)
          pd.set_option('display.max_columns', None)
          raw_data.info()
          nulls = raw_data.isnull().sum()
          training_data    =    raw_data.drop(['type',    'uri',    'track_href',
'analysis_url',
                                             'song_name',   'Unnamed:   0',   'title',
'genre'], axis=1, inplace=False)
          global_scalar = MinMaxScaler()
          id_column = training_data['id']
          training_data.drop(['id'], axis=1, inplace=True)
          global_scalar.fit(training_data)
          training_data = pd.DataFrame(global_scalar.transform(training_data),
                                       index=training_data.index,
                                       columns=training_data.columns)
          training_data['id'] = id_column
          training_data.hist()
          training_data.info()
          corr = training_data.corr()
          sns.heatmap(corr[corr > 0.1], cmap="Blues", annot=True)
      wcss = []
          for i in range(1, 20):
              kmeans = KMeans(i)
              kmeans.fit(training_data.drop(['id'], axis=1, inplace=False))
              wcss_iter = kmeans.inertia_
              wcss.append(wcss_iter)
          number_clusters = range(1, 20)
          kmeans = KMeans(n_clusters=10)
          training_data_clustered = kmeans.fit(training_data.drop(['id'], axis=1,
inplace=False))
          training_data["cluster"] = training_data_clustered.labels_
          centroids = training_data_clustered.cluster_centers_
          print(training_data.head())
          #                                                                    making
output.....................................................................
.................

userPreferences.drop(userPreferences.columns.difference(["danceability",
"energy", "key", "loudness", "mode",

"speechiness", "acousticness", "instrumentalness",
                                                                    "liveness",
"valence", "tempo", "duration_ms",

"time_signature"]), 1, inplace=True)
          # input normalizing
          userPreferences                                                        =
pd.DataFrame(global_scalar.transform(userPreferences),
                                       index=userPreferences.index,
                                       columns=userPreferences.columns)
```

```
        fields = ["id", "cluster"]
        # single playlist
        single_playlist = []
        for i in range(5):
            cluster_index                                                    =
(training_data_clustered.predict(userPreferences.iloc[[i]]))[0]
            print(cluster_index)
            cluster_songs     =     training_data[training_data.cluster     ==
cluster_index]
            cluster_songs.drop(cluster_songs.columns.difference(["id",
"cluster"]), 1, inplace=True)

single_playlist.append((cluster_songs.sample()).values.flatten().tolist())
            print(single_playlist[i])
        filename = "single_playlist.csv"
        # writing to csv file
        with open(filename, 'w') as csvfile:
            # creating a csv writer object
            csvwriter = csv.writer(csvfile)
            # writing the fields
            csvwriter.writerow(fields)
            # writing the data rows
            csvwriter.writerows(single_playlist)
        # 5 playlists
        for i in range(5):
            ith_playlist = []
            filename = "pl" + str(i + 1) + ".csv"
            cluster_index                                                    =
(training_data_clustered.predict(userPreferences.iloc[[i]]))[0]
            cluster_songs     =     training_data[training_data.cluster     ==
cluster_index]
            cluster_songs.drop(cluster_songs.columns.difference(["id",
"cluster"]), 1, inplace=True)
            for j in range(5):

ith_playlist.append((cluster_songs.sample()).values.flatten().tolist())
            with open(filename, 'w') as csvfile:
                # creating a csv writer object
                csvwriter = csv.writer(csvfile)

                # writing the fields
                csvwriter.writerow(fields)
                # writing the data rows
                csvwriter.writerows(ith_playlist)
        return single_playlist
```

Висновки до розділу 3

У третьому розділі було проведено і описано розробку власної програмної реалізації інтелектуальної рекомендаційної системи мультимедіа ресурсів. Було описано базу даних, яка складається з однієї таблиці для зберігання композицій, описано інструментарій розробки, а саме мову програмування Python, середовище розробки PyCharm та систему керування базами даних SQLite. Сформовано загальне уявлення про структуру програмного продукту, шляхом побудови діаграми класів програмного забезпечення. Окрім цього було розроблено і описано графічний інтерфейс користувача, описано основні віхи програмного коду.

ВИСНОВКИ

В ході роботи над даним проектом було виконано основну задачу, а саме розробку власної програмної реалізації інтелектуальної рекомендаційної системи мультимедіа ресурсів, яка покликана створити список рекомендацій на базі вподобань користувача.

Окрім виконання основної задачі було виконано такі периферійні завдання:

– вибір найбільш ефективного рекомендаційного методу рекомендацій мультимедійних матеріалів;

– створення алгоритму рекомендацій мультимедійних матеріалів;

– створення рекомендаційної системи, яка буде пропонувати користувачу відповідно до встановленого режиму (Фільми, музика і т.д.) відповідні матеріали, спираючись на те, які побажання він має;

– розібратися в рекомендаційних методах, в їх особливостях, можливостях і доповненнях;

– вивчити основи мови програмування Python та мови структурованих запитів SQL.

Завдяки чіткому виконання поставлених завдань та слідуванню плану результатом роботи є створена повнофункціональна інтелектуальна рекомендаційна система мультимедіа ресурсів, яка в повній мірі виконує свій функціонал, здатна підбирати рекомендації на основі вподобань користувача та готова до використання в реальних задачах за умови деяких модифікацій і дороблень.

В якості перспективи розвитку можна розглянути варіант додавання до програмного продукту декількох режимів пошуку, яке призведе до розширення функціоналу за рахунок появи можливості створення рекомендаційного списку

не тільки для музичних композицій, але й для фільмів, книг, серіалів, мультфільмів, або, навіть архітектурних пам’яток.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schedl M, Zamani H, Chen CW, Deldjoo Y, Elahi M. Current challenges and visions in music recommender systems research. Int J Multi Inform Ret. (2018) 7:95–116. doi: 10.1007/s13735–018–0154–2
2. Schedl M, Gómez E, Urbano J. Music information retrieval: recent developments and applications. Found Trends Inform Ret. (2014) 8:127–261. doi: 10.1561/1500000042
3. Downie JS. Music information retrieval. Ann Rev Inform Sci Techn. (2003) 37:295–340. doi: 10.1002/aris.1440370108
4. Dorfer M, Henkel F, Widmer G. Learning to listen, read, and follow: score following as a reinforcement learning game. In: Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR 2018), Paris (2018). p. 784–91.
5. Chou PW, Lin FN, Chang KN, Chen HY. A simple score following system for music ensembles using chroma and dynamic time warping. In: Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval (ICMR 2018), Yokohama: ACM (2018). p. 529–32. doi: 10.1145/3206025.3206090
6. Goto M, Dannenberg RB. Music interfaces based on automatic music signal analysis: new ways to create and listen to music. IEEE Signal Proc Mag. (2019) 36:74–81. doi: 10.1109/MSP.2018.2874360
7. Schedl M. Intelligent user interfaces for social music discovery and exploration of large–scale music repositories. In: Proceedings of the 22nd ACM International Conference on Intelligent User Interfaces (IUI 2017): Workshop on Theory–Informed User Modeling for Tailoring and Personalizing Interfaces (HUMANIZE 2017). Limassol: ACM (2017). p. 7–11. doi: 10.1145/3039677.3039678

8. Oramas S, Nieto O, Barbieri F, Serra X. Multi–label music genre classification from audio, text and images using deep features. In: Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR), Suzhou (2017). p. 23–30.
9. Mayer R, Rauber A. Music genre classification by ensembles of audio and lyrics features. In: Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011). Miami, FL (2011). p. 675–80.
10. Sturm BL. Classification accuracy is not enough: on the evaluation of music genre recognition systems. J Intell Inform Syst. (2013) 41:371–406. doi: 10.1007/s10844–013–0250–y
11. Yang YH, Chen HH. Machine recognition of music emotion: a review. Trans Intell Syst Techn. (2013) 3:40. doi: 10.1145/2168752.2168754
12. Huq A, Bello JP, Rowe R. Automated music emotion recognition: a systematic evaluation. J New Music Res. (2010–11) 39:227–44. doi: 10.1080/09298215.2010.513733
13. Knees P, Schedl M. Music similarity and retrieval — an introduction to audio– and web–based strategies. Berlin; Heidelberg: Springer (2016).
14. Karydis I, Lida Kermanidis K, Sioutas S, Iliadis L. Comparing content and context based similarity for musical data. Neurocomputing. (2013) 107:69–76. doi: 10.1016/j.neucom.2012.05.033
15. Schedl M, Knees P, McFee B, Bogdanov D, Kaminskas M. Music recommender systems. In: Ricci F, Rokach L, Shapira B, Kantor PB, editors. Recommender Systems Handbook, 2nd ed. Boston, MA: Springer (2015). p. 453–92.
16. van den Oord A, Dieleman S, Schrauwen B. Deep content–based music recommendation. In: Burges C, Bottou L, Welling M, Ghahramani Z, Weinberger K, editors. Advances in Neural Information Processing Systems 26 (NIPS). Lake Tahoe, NV: Curran Associates, Inc. (2013). p. 2643–51.

17. Bertin–Mahieux T, Ellis DPW, Whitman B, Lamere P. The million song dataset. In: Proceedings of the 12th International Society for Music Information Retrieval Conference. Miami, FL (2011). p. 591–6.
18. Xiaoyuan Su, Taghi M. Khoshgoftaar, A survey of collaborative filtering techniques, Advances in Artificial Intelligence archive, 2009.
19. Recommender Systems – The Textbook | Charu C. Aggarwal | Springer. Springer. 2016. ISBN 9783319296579.
20. "virtual (C# Reference)". docs.microsoft.com.
21. "Installation guidance for SQL Server on Linux". December 21, 2017. Retrieved February 1, 2018.
22. "SQL Server 2008 R2 Express Database Size Limit Increased to 10GB"
23. "What's up with SQL Server 2008 Express editions"
24. Kalen Delaney (2007). Inside Microsoft SQL Server 2005: The Storage Engine
25. "Pages and Extents" (http://msdn.microsoft.com/en–us/library/ms190969.aspx)
26. "Table and Index Organization" (https://docs.microsoft.com/en–us/previous–versions/sql/sql–server–2008–r2/ms189051(v=sql.105)?redirectedfrom=MSDN)
27. "Buffer Management" (https://docs.microsoft.com/en–us/previous–versions/sql/sql–server–2008–r2/aa337525(v=sql.105)?redirectedfrom=MSDN)
28. "Transact–SQL Reference" (https://docs.microsoft.com/en–us/previous–versions/sql/sql–server–2005/ms189826(v=sql.90)?redirectedfrom=MSDN)

Додаток А Лістинг програмного коду

```
import sys
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
import os
import csv


def music_recommender(userPreferences):
    raw_data = pd.read_csv('genres_v2.csv', dtype={'song_name': 'str'})
    print(raw_data.shape)
    pd.set_option('display.max_columns', None)
    raw_data.info()

    # data cleaning ———————————————————————————————————————————

    nulls = raw_data.isnull().sum()
    print(nulls)

    training_data = raw_data.drop(['type', 'uri', 'track_href', 'analysis_url',
                                   'song_name', 'Unnamed: 0', 'title', 'genre'], axis=1, inplace=False)

    print(training_data.shape)
    training_data = training_data[training_data.key != –1]
```

```
print("after dropping some rows:\n", training_data.shape)
print(training_data.head())

print(training_data.shape)
print(training_data.duplicated().any())
training_data.drop_duplicates(inplace=True)
print(training_data.shape)

training_data.hist()

# this global scalar will be fitted on training data and will be used for both training and test data
global_scalar = MinMaxScaler()
id_column = training_data['id']
training_data.drop(['id'], axis=1, inplace=True)
global_scalar.fit(training_data)
training_data = pd.DataFrame(global_scalar.transform(training_data),
                             index=training_data.index,
                             columns=training_data.columns)
training_data['id'] = id_column

training_data.hist()

training_data.info()

corr = training_data.corr()
sns.heatmap(corr[corr > 0.1], cmap="Blues", annot=True)
```

```
        # clustering ————————————————————————————————————————————————————————————————————————

        wcss = []
        for i in range(1, 20):
            kmeans = KMeans(i)
            kmeans.fit(training_data.drop(['id'], axis=1, inplace=False))
            wcss_iter = kmeans.inertia_
            wcss.append(wcss_iter)

        number_clusters = range(1, 20)

        kmeans = KMeans(n_clusters=10)
        training_data_clustered = kmeans.fit(training_data.drop(['id'], axis=1, inplace=False))
        training_data["cluster"] = training_data_clustered.labels_
        centroids = training_data_clustered.cluster_centers_
        print(training_data.head())

        # making output.....................................................................................
        userPreferences.drop(userPreferences.columns.difference(["danceability", "energy", "key", "loudness", "mode",
                                                                 "speechiness", "acousticness", "instrumentalness",
                                                                 "liveness", "valence", "tempo", "duration_ms",
```

```
                                                    "time_signature"]), 1, inplace=True)

        # input normalizing
        userPreferences = pd.DataFrame(global_scalar.transform(userPreferences),
                                       index=userPreferences.index,
                                       columns=userPreferences.columns)

        fields = ["id", "cluster"]

        # single playlist
        single_playlist = []
        for i in range(5):
            cluster_index =
(training_data_clustered.predict(userPreferences.iloc[[i]]))[0]
            print(cluster_index)
            cluster_songs = training_data[training_data.cluster == cluster_index]
            cluster_songs.drop(cluster_songs.columns.difference(["id",  "cluster"]),  1,
inplace=True)
            single_playlist.append((cluster_songs.sample()).values.flatten().tolist())
            print(single_playlist[i])

        filename = "single_playlist.csv"

        # writing to csv file
        with open(filename, 'w') as csvfile:
            # creating a csv writer object
            csvwriter = csv.writer(csvfile)
```

```
            # writing the fields
            csvwriter.writerow(fields)

            # writing the data rows
            csvwriter.writerows(single_playlist)

        # 5 playlists
        for i in range(5):
            ith_playlist = []
            filename = "pl" + str(i + 1) + ".csv"
            cluster_index =
(training_data_clustered.predict(userPreferences.iloc[[i]]))[0]
            cluster_songs = training_data[training_data.cluster == cluster_index]
            cluster_songs.drop(cluster_songs.columns.difference(["id", "cluster"]), 1,
inplace=True)
            for j in range(5):
                ith_playlist.append((cluster_songs.sample()).values.flatten().tolist())

            with open(filename, 'w') as csvfile:
                # creating a csv writer object
                csvwriter = csv.writer(csvfile)

                # writing the fields
                csvwriter.writerow(fields)

                # writing the data rows
```

```
            csvwriter.writerows(ith_playlist)
    return single_playlist
# If on Windows to avoid fullscreen, use the following two lines of code
import os
import sys

import pandas as pd
from musicRecommender import music_recommender
from kivy.config import Config

Config.set('graphics', 'fullscreen', '0')

from kivy.app import App
from kivy.lang import Builder
from kivy.uix.popup import Popup
from kivy.uix.button import Button
from kivy.uix.widget import Widget
from kivy.core.audio import SoundLoader
from kivy.properties import ObjectProperty
from kivy.uix.gridlayout import GridLayout
from kivy.uix.floatlayout import FloatLayout

from os import listdir, path

Builder.load_string('''
<MusicPlayer>:
```

```
canvas.before:
    Color:
        rgba: 0, 0, .1, 1
    Rectangle:
        pos: self.pos
        size: self.size

TextInput:
    id: direct
    pos: 0,root.top–35
    size: root.width–200,35
    hint_text: 'Enter File Location or Leave Empty to Browse'
Button:
    id: searchBtn
    text: 'Browse'
    size: 200,35
    background_color: 0,.5,1,1
    pos: root.width–200, root.top–35
    on_release: root.getSongs()

ScrollView:
    size_hint: None, None
    size: root.width, root.height–135
    pos: 0, 100
    GridLayout:
        id: scroll
        cols: 2
```

```
        spacing: 10
        size_hint_y: None
        row_force_default: True
        row_default_height: 40

    GridLayout:
        rows: 1
        pos: 0, 50
        size: root.width, 50
        Button:
            id: 'prevSong'
            text: '<—'
            background_color: 0,.5,1,1
            on_release: root.prevSong()
        Button:
            id: 'pause'
            text: '||'
            background_color: 0,.5,1,1
            on_release: root.pauseSong()
        Button:
            id: 'nextSong'
            text: '—>'
            background_color: 0,.5,1,1
            on_release: root.nextSong()
    Button:
        id: nowplay
        text: 'Now Playing'
```

```
        pos: 0,0
        size: root.width, 50
        background_color: 0,.5,1,1

    Button:
        id: 'openRecomend'
        text: 'Open Recomendations'
        pos: 0,0
        size: root.width, 50
        background_color: 0,.5,1,1
        on_release: root.openRecomend()

    Label:
        id: status
        text: ''
        center: root.center

<ChooseFile>:

    canvas.before:
        Color:
            rgba: 0, 0, .4, 1
        Rectangle:
            pos: self.pos
            size: self.size
    BoxLayout:
        size: root.size
```

```
        pos: root.pos
        orientation: "vertical"
        FileChooserIconView:
            id: filechooser

        BoxLayout:
            size_hint_y: None
            height: 30
            Button:
                text: "Cancel"
                background_color: 0,.5,1,1
                on_release: root.cancel()

            Button:
                text: "Select Folder"
                background_color: 0,.5,1,1
                on_release: root.select(filechooser.path)

<Recomendations>:
    canvas.before:
        Color:
            rgba: 0, 0, .4, 1
        Rectangle:
            pos: self.pos
            size: self.size
    BoxLayout:
        size: root.size
```

```
pos: root.pos
orientation: "vertical"

ScrollView:
    size_hint: None, None
    size: root.width, root.height–40
    pos: 0, 0
    GridLayout:
        id: scroll_recomend
        cols: 1
        spacing: 5
        size_hint_y: None
        row_force_default: True
        row_default_height: 40

BoxLayout:
    size_hint_y: None
    height: 30
    Button:
        text: "Exit"
        background_color: 0,.5,1,1
        on_release: root.cancel()

    Button:
        text: "Recomendations"
        background_color: 0,.5,1,1
        on_release: root.doRecomend()
```

```
'')


class Recomendations(FloatLayout):
    select = ObjectProperty(None)
    cancel = ObjectProperty(None)

    def doRecomend(self):
        file_name = 'input_tracks.csv'
        if not os.path.isfile(file_name):
            print("File does not exist")
            sys.exit()
        else:
            userPreferences = pd.read_csv(file_name)
            playlist =  music_recommender(userPreferences)
            for el in playlist:
                btn = Button(text=str(el))
                self.ids.scroll_recomend.add_widget(btn)
```

```
class ChooseFile(FloatLayout):
    select = ObjectProperty(None)
    cancel = ObjectProperty(None)



class MusicPlayer(Widget):
    directory = "  # location of songs folder
    nowPlaying = "  # Song that is currently playing
    songID = 0
    songs = []
    state = 'pause'

    def openRecomend(self):
        content = Recomendations(select=self.select,
                                 cancel=self.dismiss_popup)

        self._popup = Popup(title="Recomendations", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()



    def getpath(self):
        try:
            f = open("sav.dat", "r")
            self.ids.direct.text = str(f.readline())
            f.close()
            self.ids.searchBtn.text = "Scan"
```

```
            self.getSongs()
        except:
            self.ids.direct.text = ''

    def savepath(self, path):
        f = open("sav.dat", "w")
        f.write(path)
        f.close()

    def dismiss_popup(self):
        self._popup.dismiss()

    def fileSelect(self):
        content = ChooseFile(select=self.select,
                             cancel=self.dismiss_popup)

        self._popup = Popup(title="Select Folder", content=content,
                            size_hint=(0.9, 0.9))
        self._popup.open()

    def select(self, path):
        self.directory = path
        self.ids.direct.text = self.directory
        self.ids.searchBtn.text = "Scan"
        self.savepath(self.directory)
        self.getSongs()
        self.dismiss_popup()
```

```
    def nextSong(self):
        if self.songID + 1 < len(self.songs):
            self.songID += 1
            try:
                self.nowPlaying.stop()
            except:
                pass
            finally:
                self.nowPlaying = SoundLoader.load(self.directory + self.songs[self.songID])
                self.nowPlaying.play()
                self.ids.nowplay.text = self.songs[self.songID][:–4]
                self.state = 'play'

    def prevSong(self):
        if self.songID – 1 >= 0:
            self.songID –= 1
            try:
                self.nowPlaying.stop()
            except:
                pass
            finally:
```

```
                self.nowPlaying = SoundLoader.load(self.directory + self.songs[self.songID])
                self.nowPlaying.play()
                self.ids.nowplay.text = self.songs[self.songID][:–4]
                self.state = 'play'

        def pauseSong(self):
            if self.state == 'play':
                self.nowPlaying.stop()
                self.state = 'pause'
            elif self.state == 'pause':
                self.nowPlaying.play()
                self.state = 'play'

        def getSongs(self):

            # List to hold songs from music directory
            self.directory = self.ids.direct.text  # Directory entered by the user

            if self.directory == '':
                self.fileSelect()

            # To make sure that the directory ends with a '/'
            if not self.directory.endswith('/'):
                self.directory += '/'

            # Check if directory exists
```

```
if not path.exists(self.directory):
    self.ids.status.text = 'Folder Not Found'
    self.ids.status.color = (1, 0, 0, 1)

else:

    self.ids.status.text = ''

    self.ids.scroll.bind(minimum_height=self.ids.scroll.setter('height'))

    # get mp3 files from directory
    for fil in listdir(self.directory):
        if fil.endswith('.mp3'):
            self.songs.append(fil)

    # If there are no mp3 files in the chosen directory
    if self.songs == [] and self.directory != '':
        self.ids.status.text = 'No Music Found'
        self.ids.status.color = (1, 0, 0, 1)

    self.songs.sort()
    for song in self.songs:

        def playSong(bt):
            try:
                self.nowPlaying.stop()
            except:
```

```
                    pass
                finally:
                    self.nowPlaying = SoundLoader.load(self.directory + bt.text +
'.mp3')
                    self.nowPlaying.play()
                    self.ids.nowplay.text = bt.text
                    self.state = 'play'
                    self.songID = self.songs.index(bt.text + '.mp3')
            btn = Button(text=song[:–4], on_press=playSong)
            icon = Button(size_hint_x=None, width=50,
background_down="ico.png", background_normal="ico.png")

            # Color Buttons Alternatively
            if self.songs.index(song) % 2 == 0:
                btn.background_color = (0, 0, 1, 1)
            else:
                btn.background_color = (0, 0, 2, 1)

            self.ids.scroll.add_widget(icon)  # Add icon to layout
            self.ids.scroll.add_widget(btn)  # Add btn to layout


    class KVMusicApp(App):

        def build(self):
            music = MusicPlayer()
            music.getpath()
```

```
        return music

    def on_pause(self):
        return True

    def on_resume(self):
        pass


if __name__ == "__main__":
    KVMusicApp().run()
```