НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

# ДИПЛОМНА РОБОТА

**на здобуття ступеня бакалавра**

з напряму підготовки
6.050101 “Комп’ютерні науки”

на тему: Алгоритмізація та реалізація алгоритму аналізу інформації щодо пропозицій покращення статусу гравця/команди гравців

Виконав: студент 4 курсу, групи ТР-52

Степаненко Владислав Олегович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н.Коваль Олександр Васильович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

# ЗАВДАННЯ
**на дипломну роботу студенту**
Степаненку Владиславу Олеговичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Алгоритмізація та реалізація алгоритму аналізу інформації щодо пропозицій покращення статусу гравця/команди гравців”

керівник роботи доцент, к.т.н. Коваль Олександр Васильович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”___”_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи персональний комп’ютер під керуванням операційної системи Microsoft Windows, мова програмування C#, мова програмування TypeScript.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі алгоритми рекомендацій та можливі засоби їх реалізації, обґрунтувати обрані алгоритми та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень) 1. Рекомендаційні системи. 2. Алгоритми аналізу інформації. 3. Математична модель задачі. 4.Колаборативна фільтрація. 5. Алгоритм матричної факторизації.

7.Апробація результатів роботи

6. Публікації:

Дата видачі завдання ”___”_____ 201__ р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Вивчення та аналіз задачі | | |
| 2. | Розробка архітектури та загальної структури системи | | |
| 3. | Розробка структур окремих підсистем | | |
| 4. | Підготовка матеріалів | | |
| 5. | Програмна реалізація системи | | |
| 6. | Захист програмного продукту | | |
| 7. | Оформлення пояснювальної записки | | |
| 8. | Передзахист | | |
| 9. | Захист | | |

Студент _____ Степаненко В.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Коваль О.В.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Метою роботи була розробка та реалізація алгоритму рекомендацій щодо покращення статусу команди гравців. Метод формування рекомендацій базується на алгоритмі аналізу даних з використанням матричної факторизації. Розроблена система складається з серверної та клієнтської частин. Створена рекомендаційна система дозволяє користувачу переглядати інформацію про команди, гравців та отримувати рекомендації щодо залучення нових гравців до команди.

Записка містить 73 сторінки,15 рисунків, 3 додатки і 29 посилань.

Ключові слова: РЕКОМЕНДАЦІЙНІ СИСТЕМИ, АНАЛІЗ ДАНИХ, МАТРИЧНА ФАКТОРИЗАЦІЯ, ASP.NET CORE.

# ABSTRACT

The purpose of the work was to develop and implement a recommendation algorithm to improve the status of the team players. The method of forming recommendations is based on the algorithm of data analysis using matrix factorization. The developed system consists of server and client parts. The created recommendation system allows the user to view information about teams, players and get recommendations for attracting new players to the team.

The note contains 73 pages, 15 figures, 3 attachments and 29 references.

Keywords: RECOMMENDATION SYSTEMS, DATA ANALYSIS, MATRIX FACTORIZATION, ASP.NET CORE.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СКБД — система керування базами даних;

РС — рекомендаційна система;

ШНМ — штучна нейронна мережа;

SVD — Singular Value Decomposition;

NNMF — Non negative matrix factorization;

XML — Extensible Markup Language;

JSON — JavaScript Object Notation;

TDD — Test Driven Development;

DLL — Dynamic Link Library;

REST — Representation State Transfer;

HTTP — Hyper Text Transfer Protocol;

# ВСТУП

Рекомендаційні системи — це програми і сервіси, які аналізують інтереси користувачів і намагаються передбачити, що саме буде найцікавішим для конкретного користувача в даний момент часу. Такі системи показують перевагу контенту для конкретного користувача на основі даних, зазначених користувачем явно або на основі його взаємодії з системою.

Рекомендаційна система зазвичай сфокусована на певному типі рекомендацій, тому її структура, графічний інтерфейс та алгоритм, що використовується для генерації рекомендацій, все це налаштовано для забезпечення ефективної роботи системи у конкретній предметній області.

Рекомендаційні системи повинні мати наступні властивості:

— система повинна адаптуватися під конкретного користувача, так як переваги можуть значно відрізнятися у різних людей;

— система повинна враховувати поточні переваги користувача, підлаштовуючись під нього згодом;

— система повинна постійно знаходити нові галузі інформації і пропонувати їх користувачеві.

Все це робить ресурси, засновані на рекомендаційних механізмах, привабливими для користувача. З іншого боку, подібні системи цікаві і власникам самих ресурсів, на яких розміщуються рекомендаційні системи, так як за допомогою подібних інструментів підвищується прибутковість самого ресурсу.

Рекомендаційні системи знайшли своє застосування в багатьох сферах життєдіяльності людини: пошуку фільмів і наукових статей, роздрібній торгівлі, соціальних мережах, електронній комерції, онлайн-банкінгу, спортивних середовищах, тощо.

Подібна методика може бути застосована і до рекомендацій в області кіберспорту. У наш час він розвивається досить стрімко, та залучає все більше

ігрових дисциплін. За цими дисциплінами все частіше проводяться найперспективніші турніри, які мають найбільшу кількість гравців по всьому світу.

У першому розділі описується постановка задачі рекомендацій щодо покращення статусу команди.

У другому розділі наводиться історія розвитку рекомендаційних систем та аналізуються вже відомі підходи до реалізації рекомендацій.

У третьому розділі вказуються засоби розробки програмного продукту.

У четвертому розділі надається опис структури програмного продукту, функціональні схеми, опис ключових модулів, наведено діаграми класів.

У п’ятому розділі описано системні вимоги до робочих станцій, наведено вказівки до інсталяції та запуску програмного продукту, а також його подальшого використання.

# 1 ЗАДАЧА ОТРИМАННЯ РЕКОМЕНДАЦІЙ ЩОДО ПОКРАЩЕННЯ СТАТУСУ КОМАНДИ ГРАВЦІВ

Мета цього проекту полягає в тому, щоб дати змогу користувачеві оцінити існуючу ситуацію в команді, та надати рекомендації щодо нових гравців, залучення яких підвищить злагодженість та результативність команди загалом.

Використовуючи теоретичні відомості надані керівником дипломної роботи та базу даних, надану співробітником бази проходження практики, потрібно розробити та реалізувати алгоритм рекомендацій щодо покращення статусу команди гравців. Потенційними користувачами системи кіберспортивні аналітики для вирішення своїх прикладних задач.

Програмний продукт повинен підтримувати наступні функції:

— надавати перегляд інформації про команди;

— надавати перегляд інформації про гравців;

— надавати можливість перегляду списку рекомендованих гравців до кожної команди.

Сутність рекомендацій у рамках даної дипломної роботи полягає у тому, щоб система пропонувала заданій команді список гравців, які при переході до даної зможуть підвищити її рейтинг та підтримувати високий відсоток злагодженості з існуючими гравцями в команді. При цьому слід враховувати певні показники, що характерні для предметної області кіберспорту, тобто такі як:

— загальний рейтинг команди;

— рейтинг рекомендованого гравця;

— географія місцезнаходження гравців команди;

— мова спілкування гравців у команді.

Стратегія для формування рекомендацій, яку було обрано для вирішення даної задачі, заснована на підходах колаборативної фільтрації, а саме матричній факторизації.

Для розробки серверної частини програмного продукту була використана платформа .NET Core. Для розробки інтерфейсу користувача було використано платформу для веб-застосунків Angular. В якості сховища даних було обрано реляційну СКБД PostgreSQL.

# 2 АНАЛІЗ ПРОБЛЕМИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ ТА ПРИНЦИПИ ЇХ ПОБУДОВИ

На даний момент часу існують веб-сайти, на яких можна переглянути результати кіберспортивних матчів, статистику команд та гравців, можна здійснити пошук команд за обраними критеріями. Ці веб-сайти використовують інформацію з профілів гравців, за їх дозволом. Також є системи керування командами, які надають змогу тренеру або власнику команди. Наразі, існує досить мало прикладів рекомендаційних систем в області кіберспорту, проте можна розглянути підходи до проблем рекомендацій серед інших областей даних та реалізувати їх для розв’язку своєї задачі.

## 2.1 Історія розвитку рекомендаційних систем

Незважаючи на те, що проблема рекомендаційних систем постала відносно недавно, її теоретична база в області машинного навчання існує вже багато років. В середині 50-х років було сформульовано математичні підходи та описано моделі, які є базою для нині існуючих алгоритмів.

На початку 1990-х років колаборативна фільтрація стала використовуватись як інструмент для боротьби з надлишковою інформацією в інтернеті [1]. Експериментальний поштовий сервіс Tapestry [2] став однією із перших систем, які використовували даний підхід. Користувач мав змогу створювати власноруч запити, які базувалися на діях других користувачів. Такі маніпуляції вимагали від користувача певної активності, проте, з іншого боку це надавало змогу користувачам визначати актуальність отриманої інформації.

Пізніше з’явились системи фільтрації, які автоматично визначали для користувача інформацію, спираючись на думку інших учасників системи. Ця інформація узагальнювалась для представлення рекомендацій. Програмний додаток GroupLens [3] використовував даний метод для знаходження статей в мережі Usenet, які могли бути цікавими для конкретного користувача. Система збирала інформацію про оцінки статей користувачами для надання персоналізованих результатів.

В той же момент рекомендаційні системи стали предметом багатьох обговорень в області інформаційного пошуку та машинного навчання. Як наслідок, рекомендаційні алгоритми все частіше стали з’являтися в сфері маркетингу для збільшення числа продажів.

Перші комерційні рекомендаційні системи з’явились наприкінці 1990-х. Відома компанія Amazon однією з перших інтегрувала подібні технології. Базуючись на історії переглянутих товарів, система робила припущення щодо продуктів, які могли б бути цікавими користувачу. Після успіху компанії Amazon інші представники сегменту електронної комерції звернули увагу на рекомендаційні рішення.

У той же час компанія Google представила свою систему персоналізації новин Google News [4]. Вона працювала на основі історії кліків користувачів. В даному випадку, новини розглядаються як об'єкти інтересу, а кліки користувачів як позитивна оцінка до статті. Алгоритм колаборативної фільтрації застосовувався до зібраних рейтингів. На основі цього, система робила висновок про персоналізовану видачу статей для конкретного користувача.

Особливу увагу рекомендаційні алгоритми привернули до себе у 2006 році, коли компанія Netflix почала змагання Netflix Prize, метою якого було створення алгоритму, який зміг би покращити результат вже існуючого алгоритму CineMatch хоча б на 10%. Цей конкурс викликав численні дискусії, як в академічних колах, так і серед аматорів.

Бурхливий розвиток соціальних мереж у середині 2000-х років минув не без участі рекомендаційних систем. Найбільша соціальна мережа Facebook запровадила алгоритм рекомендацій потенційних соціальних зв’язків. Такі рекомендації

забезпечували швидкий розвиток та зв’язність мережі, тому природа цього алгоритму дещо відрізнялась від стандартних рекомендаційних алгоритмів.

## 2.2 Функції рекомендаційних систем

Наведемо головні причини впровадження рекомендаційних сервісів у бізнес-середовищі:

— підвищення рівню задоволеності користувачів. Правильно спроектована рекомендаційна система може справити добре враження на користувача при роботі з нею. Користувач буде вважати рекомендації цікавими, актуальними та ефективними.

— покращити рівень розуміння користувачів. Наступною важливою функцією рекомендаційних систем є опис уподобань користувача, які збираються явно або передбачаються системою. Власник сервісу може потім прийняти рішення про повторне використання цієї інформації для ряду інших цілей.

— надання можливості отримувати рекомендації у вигляді послідовності. Ідея полягає в тому, щоб замість однієї рекомендації отримувати послідовний список пунктів, які задовольнятимуть потребам користувача. Типовими прикладами є рекомендації книг, серіалів, друзів у соціальних мережах та нових гравців у команду.

— збільшення кількості проданих предметів. Є досить важливою функцією для комерційних рекомендаційних систем. Мається на увазі можливість продавати додатковий набір предметів разом з тими товарами, що продаються без використання рекомендацій. Ця мета може бути досягнута, тому що рекомендовані предмети можуть задовольнити потребам користувача.

— формування джерела надійних рекомендацій. Деякі користувачі не мають довіри до рекомендаційних систем, тому вони усіляко випробовують систему, щоб перевірити наскільки правильними є рекомендації. Тому деякі системи можуть

надавати змогу користувачам переглянути інформацію про те, на чому засновані отримані рекомендації.

## 2.3 Підходи для формування рекомендацій

Для реалізації своєї основної функції — визначення корисних рекомендацій для користувача, рекомендаційна система повинна передбачити, що даний предмет можна порекомендувати. Для цього система повинна вміти прогнозувати вагомість для користувача щодо цих предметів, або принаймні порівнювати вагомість елементів між собою та вирішувати, які з них рекомендувати на основі цих порівнянь[5].

Слід зазначити, що вагомість рекомендованого елемента для користувача може залежати від його обізнаності в даній предметній області, часу, місцезнаходження, мови спілкування, тощо. Отже, рекомендації повинні бути адаптовані до певного набору деталей і, в результаті, стає все важче оцінити правильність рекомендацій.

У загальноприйнятій класифікації рекомендаційних систем[6], розрізняють шість різних видів, а саме:

— РС, що базуються на фільтрації змісту: система навчається рекомендувати елементи, подібні до тих, які користувачу подобались у минулому. Наприклад, якщо користувач оцінив якийсь фільм або залишив позитивний відгук про гравця з яким провів матч, тоді система може навчитись рекомендувати елементи подібні до цих.

— колаборативна фільтрація: найпростіша реалізація цього підходу полягає у тому, щоб рекомендувати користувачам елементи, які в минулому подобалися іншим користувачам зі схожими смаками. Обрахування схожості користувачів засноване на історії їх оцінок.

— РС, що базуються на демографічному підході: цей тип системи рекомендує елементи на основі демографічної інформації про користувача. Передбачається, що

для різних демографічних ніш повинні бути представлені різні рекомендації. Наприклад, користувачі відправляються на певні веб-сайти на основі мови. Хоча цей підхід є досить популярним у маркетинговій області, існує відносно мало інформації про дослідження рекомендаційних систем на демографічному підході.

— РС, що базуються на знаннях: такі системи рекомендують елементи, що засновані на специфічних знаннях про предметну область. У загальному випадку, вони знають про те, як певні функції елементів відповідають потребам користувачів.

— РС, що використовують громадську думку: цей тип систем видає рекомендації користувачу, що засновані уподобань його друзів. Досліди свідчать про те, що люди, як правило, більше покладаються на думку своїх друзів, а ніж на думку сторонніх осіб. Даний підхід широко використовується у соціальних мережах.

— гібридні РС: ці системи базуються на комбінації вищезгаданих підходів. Методи комбінування гібридної системи намагаються використовувати переваги одного методу для компенсації недоліків іншого. Тобто, методи доповнюють один одного.

## 2.4 Огляд відомих систем, що використовують рекомендації

Наразі рекомендаційні системи вже інтегровані до багатьох веб-застосунків, якими кожен день користуються мільйони людей. Розглянемо приклади ресурсів, які використовують рекомендаційні алгоритми.

Веб-портал DreamTeam — електронна мережа для пошуку гравців та керування кіберспортивною командою[7]. Ця мережа дозволяє користувачеві знаходити гравців, команди, тренерів та приймати участь в змаганнях. Ефективно керувати командою, формувати аналітичні звіти. Також тут присутній модуль рекомендацій щодо підбору нових гравців до команди, який враховує рейтинг команди, статистику та географічне розташування гравців.

Бізнес-орієнтована соціальна мережа LinkedIn використовує рекомендаційний механізм на основі колаборативної фільтрації. Він пропонує користувачеві вакансії, які могли б його зацікавити та групи, до яких він міг би долучитися. Цей алгоритм працює з використанням мережевої платформи Apache Hadoop.

Сервіс Last.fm рекомендує музичні композиції користувачам, збираючи та аналізуючи інформацію про його вподобання. Цей сервіс рекомендує записи, яких немає в бібліотеці користувача, проте вони прослуховувались іншими користувачами зі схожими інтересами. Цей алгоритм також є прикладом колаборативної фільтрації, оскільки він використовує поведінку користувачів.

## 2.5 Огляд алгоритмів аналізу даних для рекомендаційних систем

Процес інтелектуального аналізу даних, як правило складається з трьох кроків, які виконуються послідовно: попередня обробка даних, аналіз даних, інтерпретація результатів. Схему процесу наведено на рисунку (2.1)

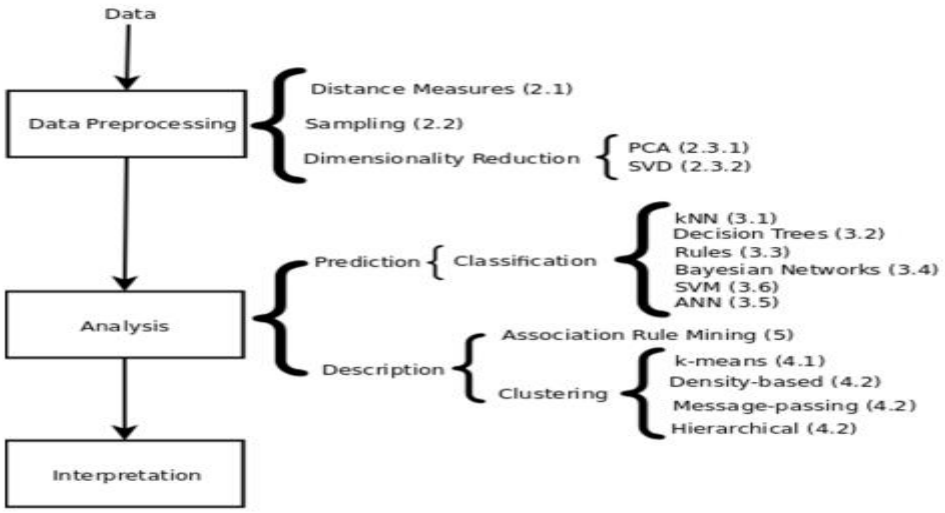


Рисунок 2.1 — Схема процесу інтелектуального аналізу даних

### 2.5.1 Алгоритм, що використовує коефіцієнт кореляції

Нехай маємо матрицю $R$ таку, що рядки матриці є векторами уподобань для кожного користувача, а стовпчики є векторами оцінок для кожного предмета рекомендацій. Насамперед залишимо в цих векторах тільки ті елементи, для яких нам відомі значення для обох векторів, тобто слід залишити тільки такі предмети, які оцінили обидва користувачі. В результаті необхідно визначити, наскільки схожі два вектори дійсних чисел. Для цього необхідно порахувати коефіцієнт кореляції

$$w_{i,j} = \frac{\sum_a (r_{i,a} - \bar{r}_i)(r_{j,a} - \bar{r}_j)}{\sqrt{\sum_a (r_{i,a} - \bar{r}_i)^2} \sqrt{\sum_a (r_{j,a} - \bar{r}_j)^2}} \quad (2.1)$$

де $\bar{r}i$ — середній рейтинг, виставлений користувачем.

Іноді використовують так звану “косинусну схожість”, використовуючи косинус кута між векторами

$$w_{i,j} = \frac{\sum_a r_{i,a} r_{j,a}}{\sqrt{\sum_a r_{i,a}{}^2} \sqrt{\sum_a r_{j,a}{}^2}} \quad (2.2)$$

Для того, щоб скористатися цими коефіцієнтами, необхідно наблизити новий рейтинг як середній рейтинг даного користувача плюс зважене відхилення від середнього рейтингу інших користувачів

$$r_{i,a} = \bar{r}_i + \frac{\sum_j (r_{j,a} - \bar{r}_j) w_{i,j}}{\sum_j |w_{i,j}|} \quad (2.3)$$

Перевагою даного алгоритму є те, що він достатньо простий у реалізації. Проте він вважається застарілим, наразі існують інші алгоритми, які швидше виконують свою задачу та видають більш точні результати.

### 2.5.2 Алгоритм, що використовує Баєсовий класифікатор

Баєсовий класифікатор є методом вирішення проблеми класифікації на основі ймовірнісних підходів. Він ґрунтується на визначенні умовної ймовірності та теоремі Баєса.

Теорема Баєса — це одна з основних теорем у теорії ймовірностей. Вона визначає ймовірність настання події в умовах, коли на основі спостережень відома лише деяка інформація про подію. Відповідно до [8] умовна ймовірність $P(A \mid H)$ події $A$ за умови настання події події $H$ обчислюється за формулою

$$P(A \mid H) = \frac{P(A, H)}{P(H)} \tag{2.4}$$

де $P(A)$ та $P(H)$ — це ймовірності настання кожної події окремо,

$P(A, H)$ — ймовірність одночасного настання подій $A$ та $H$.

Виходячи з теореми про множення ймовірностей

$$P(A, H) = P(A \mid H) * P(H) = P(H \mid A) * P(A) \tag{2.5}$$

Враховуючи (2.4) та (2.5), теорема Баєса матиме наступний вигляд

$$P(A \mid H) = \frac{P(H \mid A) * P(A)}{P(H)} \tag{2.6}$$

Перепишемо формулу (2.6) через інші позначення

$$p(\theta \mid D) = \frac{p(\theta) * p(D \mid \theta)}{p(D)} \tag{2.7}$$

де $p(\theta \mid D)$ — це розподіл ймовірностей параметрів моделі після того, як було прийнято до уваги дані, тобто апостеріорна ймовірність.

Напряму її знайти, як правило, не виходить, тому потрібно використовувати теорему Баєса. $p(D \mid \theta)$ — це так звана правдоподібність, тобто ймовірність даних за умови зафіксованих параметрів моделі. Зазвичай знайти це число легко, тому що будова моделі передбачає задання функції правдоподібності. Функція $p(\theta)$ — це апріорна ймовірність, яка є математичною формалізацією нашого інтуїтивного бачення предмету, тобто формалізацією того, що ми знали раніше, до проведення експериментів.

### 2.5.3 Кластерний аналіз

Основною проблемою для масштабування класифікатора колаборативної фільтрації є кількість операцій, що беруть участь у обчисленні відстаней наприклад, для пошуку найближчих сусідів[5].

Можливим виходом із даної ситуації, є зменшення розмірності. Але, навіть якщо зменшити розмірність особливостей користувача та предмету, ми матимемо ще багато об'єктів для обчислення відстаней. Саме тут можуть допомогти алгоритми кластеризації. Такі ж самі підходи є справедливими і для рекомендаційних систем, що базується на змісті, де для отримання подібних користувачів або об’єктів обчислюються відстані між ними.

У такому випадку, кластеризація підвищить ефективність, оскільки кількість операцій обчислення зменшується. Проте, на противагу до інших методів зменшення розмірностей, вона навряд чи зможе підвищити точність роботи моделі. Таким чином, використовувати кластеризацію у рекомендаційних системах потрібно з обережністю, підтримуючи компроміс між підвищенням ефективності та можливим зниженням точності.

Кластеризація — це алгоритм машинного навчання без вчителя. Він використовується для розбиття множини елементів на підмножини (кластери) таким

чином, щоб елементи в одній і тій самій підмножині були більш схожими за обраним критерієм, ніж елементи в різних підмножинах.

Кластеризацію роблять з метою:

— класифікація об'єктів. Спроба зрозуміти залежності між об'єктами шляхом виявлення можливих груп даних (кластерів). Таке розділення вибірки спрощує подальшу обробку даних і прийняття рішень, дозволяє застосувати до кожного кластеру свій метод аналізу. В даному випадку, для того, щоб узагальнити закономірності, прагнуть зменшити число кластерів;

— стиснення даних. Можна скоротити розмір вхідної вибірки, взявши один або кілька найбільш типових представників з кожного кластера. У такому випадку кількість кластерів не є важливим критерієм, набагато важливішим є найбільш точно окреслити межі для кожного кластера;

— виявлення новизни (або виявлення шуму). Виділення серед множини об'єктів тих, які не підходять за заданими критеріямидо жодного кластеру. Виявлені об'єкти в подальшому слід обробляти окремо.

Для оцінки якості кластеризації постановку задачі потрібно переформулювати в рамках задачі оптимізації. Необхідно поставити у відповідність об’єктам вхідної множини мітки кластерів таким чином, щоб значення вибраного функціоналу якості приймало найкраще значення. Серед таких функціоналів можуть бути максимум середньої міжкластерної відстані.

### 2.5.4 Штучні нейронні мережі

Штучна нейронна мережа (ШНМ) — це математична модель, а також пристрій паралельних обчислень, що представляють собою систему з’єднаних і взаємодіючих між собою штучних нейронів[9].

Поняття нейронних мереж виникло при дослідженні процесів в області головного мозку. Групування в мозку людини відбувається так, що інформація

обробляється динамічним, інтерактивним і самоорганізуючим шляхом. Біологічні нейроні мережі створені в тривимірному просторі з мікроскопічних компонентів і здатні до різноманітних з'єднань, а для створеної людиною мережі існують фізичні обмеження[9].

Якщо розглянути структуру ШНМ, то можна сказати про те, що вона складається з набору з’єднаних між собою штучних нейронів. ШНМ має фіксовані передаючі функції для всіх нейронів та ваги, які корегуються у процесі навчання останньої. Перший шар нейронів має зовнішній вхід мережі, і відповідно, останній шар нейронів має зовнішній вихід мережі. Подаючи вектор вхідних значень на входи нейронної мережі, можна отримати вихідний вектор на виході мережі. На рисунку (2.2) зображено структуру простої нейромережі

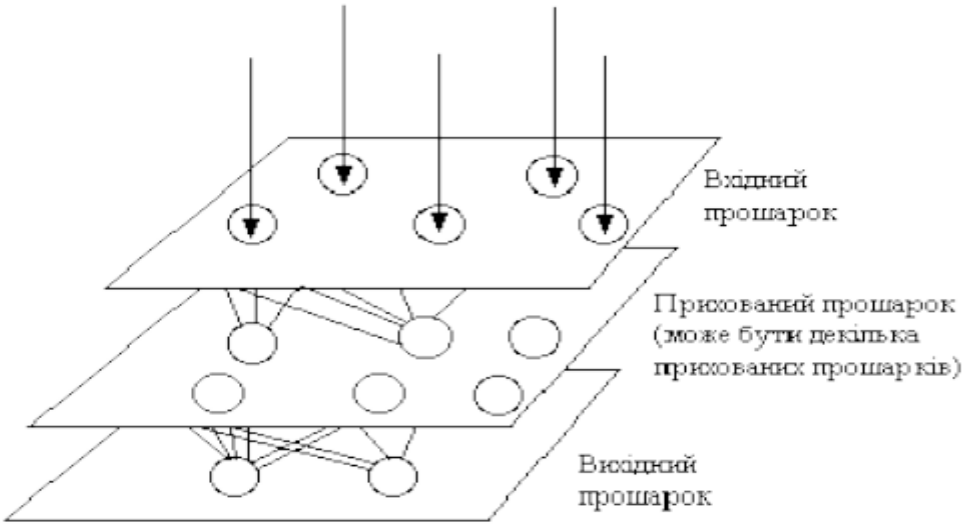


Рисунок 2.2 — Будова простої нейронної мережі

Таким чином, робота ШНМ складається в перетворенні вхідного вектора у вихідний вектор, причому це перетворення задається вагами мереж[9].

Історично першою публікацією, що заклала підвалини для створення штучних нейронів та нейронних мереж, вважають роботу Уоррена С. Мак-Каллока та Вальтера Піттса[10]. У цій роботі було започатковано теорію, в основі якої лежав той факт, що всі аспекти нервової діяльності можна моделювати за допомогою

мережі елементів, які мають два стійких стани[11]. Така модель дістала назву найпростішого персептрону. Його структуру наведено на рисунку (2.3)

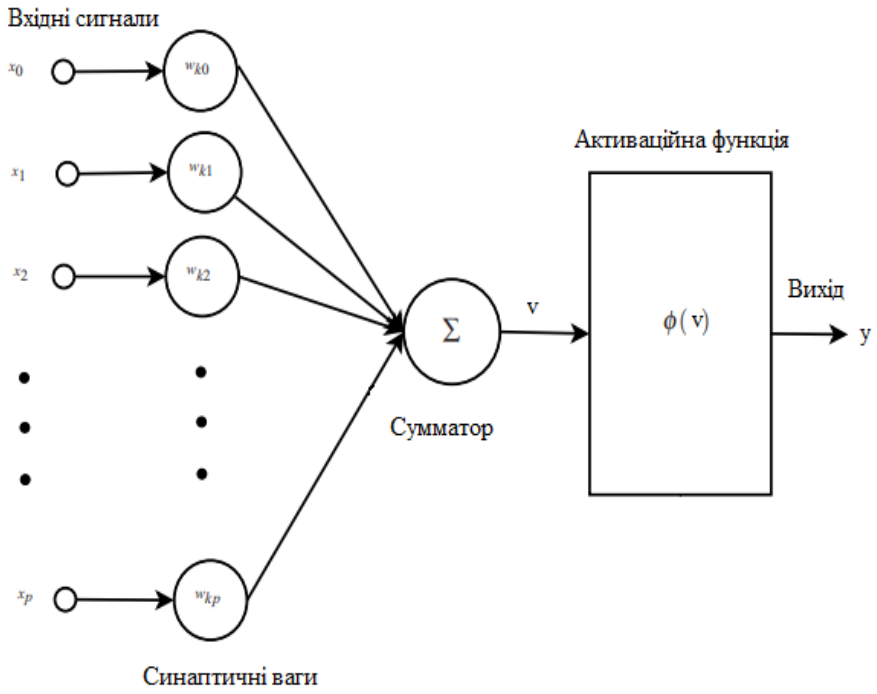


Рисунок 2.3 — Найпростіший персептрон

Шляхом перемноження вектору вхідних сигналів на вектор синаптичних вагових коефіцієнтів відбувається моделювання функцій синапсів. Отримані сигнали потрапляють на вхід до суматора, який виконує подальшу обробку за формулою

$$v = \sum_{i=0}^{n} w_{ki} x_i \tag{2.8}$$

Отримане значення змінної $v$ є аргументом активаційної функції:

$$y = \phi(v) \tag{2.9}$$

Перша активаційна функція була запропонована у роботі Уоррена С. Мак-Каллока та Вальтера Піттса[10] та мала такий вигляд:

$$y = \begin{cases} 1 \, при \, v \le 0, \\ 0 \, при \, v > 0 \end{cases} \tag{2.10}$$

З розвитком ШНМ, в області аналізу даних з’явилась велика кількість парадигм, що підтримують різні активаційні функції. Найбільшого поширення з них набула сигмоїдальна функція, що існує в дискретному та аналоговому варіантах.

Дискретна сигмоїдальна функція з параметром або функція Хевісайда є прикладом активаційної функції та виглядає наступним чином:

$$y = \begin{cases} 1 \, при \, v \le a, \\ 0 \, при \, v > a \end{cases} \tag{2.11}$$

Якщо аргумент активаційної функції Хевісайда не перевищує за значенням параметр $a$, то нейрон знаходиться у пасивному стані. Якщо відбувається перевищення параметра, тоді активаційна функція видає фіксоване значення, що приймається за логічну одиницю.

Аналоговий варіант сигмоїдальної функції є нелінійною активаційною функцією з параметрами $b, c, d$ та задається наступним виразом:

$$y = \frac{b}{c + e^{dv}} \tag{2.12}$$

Для спрощення параметрам $b, c, d$ задають такі значення:

$$\begin{cases} b = 1, \\ c = 1, \\ d = -1 \end{cases} \tag{2.13}$$

Враховуючи (2.14) та (2.15) активаційна функція матиме такий вигляд:

$$y = \frac{1}{1 + e^{-v}} \tag{2.14}$$

При використанні даної активаційної функції, амплітуда вихідного сигналу нейрона залежить від амплітуди вхідного. На рисунку (2.4) представлено графік сигмоїдальної функції активації

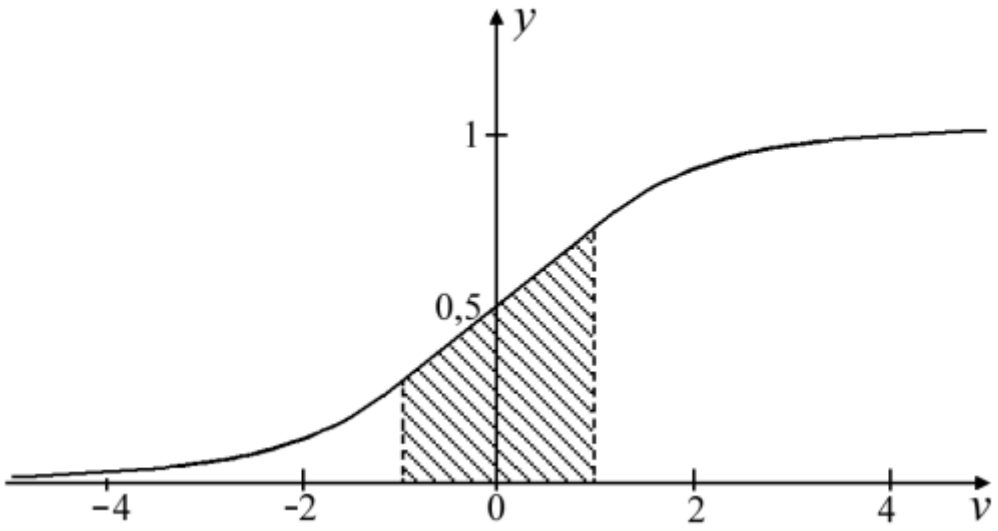


Рисунок 2.4 — Нелінійна сигмоїдальна фунція

Для випадку використання вхідних сигналів з від’ємного діапазону чисел, існує подібна до функції Хевісайда, функція гіперболічного тангенса (рисунок 2.5)

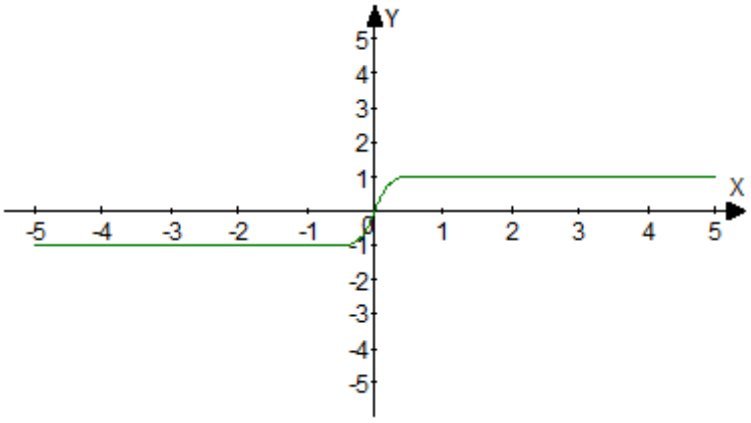


Рисунок 2.5 — Гіперболічний тангенс

У параметризованому вигляді активаційна функція гіперболічного тангенсу має вигляд

$$y = k\ th(mv) \tag{2.15}$$

Спростивши параметри, можна отримати найбільш поширений варіант даної функції активації

$$y = th(v) \tag{2.16}$$

Для побудови моделі рекомендаційної системи, ШНМ можна використовувати подібно до Баєсових мереж. Однак, досі не існує переконливого дослідження щодо підвищення продуктивності при використанні нейронних мереж в області рекомендацій.

У своєму дослідженні[12] автори провели комплексний експеримент щодо використання декількох алгоритмів машинного навчання для рекомендацій веб-сайту. Основною метою цього експерименту було порівняння простого Баєсового класифікатора з більш обчислювально-затратними альтернативними методами, такими як дерева рішень та нейронні мережі. Результати експерименту показали, що дерева рішень працюють значно гірше. ШНМ та Баєсовий класифікатор показали приблизно однакові результати продуктивності. Тому автори дійшли висновку, що рекомендаційні системи не мають потреби у використанні нелінійних класифікаторів, таких як ШНМ.

### 2.5.6 Дерева прийняття рішень

Дерево рішень — це модель, яка є сукупністю правил для прийняття рішень. Графічно цю модель можна уявити у вигляді деревоподібної структури, де моментам прийняття рішень відповідають вузли. Всередині вузлів відбувається розгалуження процесу, тобто поділ його на так звані “гілки” в залежності від зробленого вибору. Кінцеві (або термінальні) вузли називають “листками”. Кожен такий листок — це кінцевий результат послідовного прийняття рішень.

На початку роботи методу дані, що приходять на обробку, знаходяться у так званому “корені” дерева. Далі, залежно від того, яке рішення було прийнято у вузлах, процес класифікації зупиняється на одному з листків (термінальних вузлів),

де шуканій позначці класу присвоюється те чи інше позначення. На рисунку (2.6) зображено структуру такого дерева

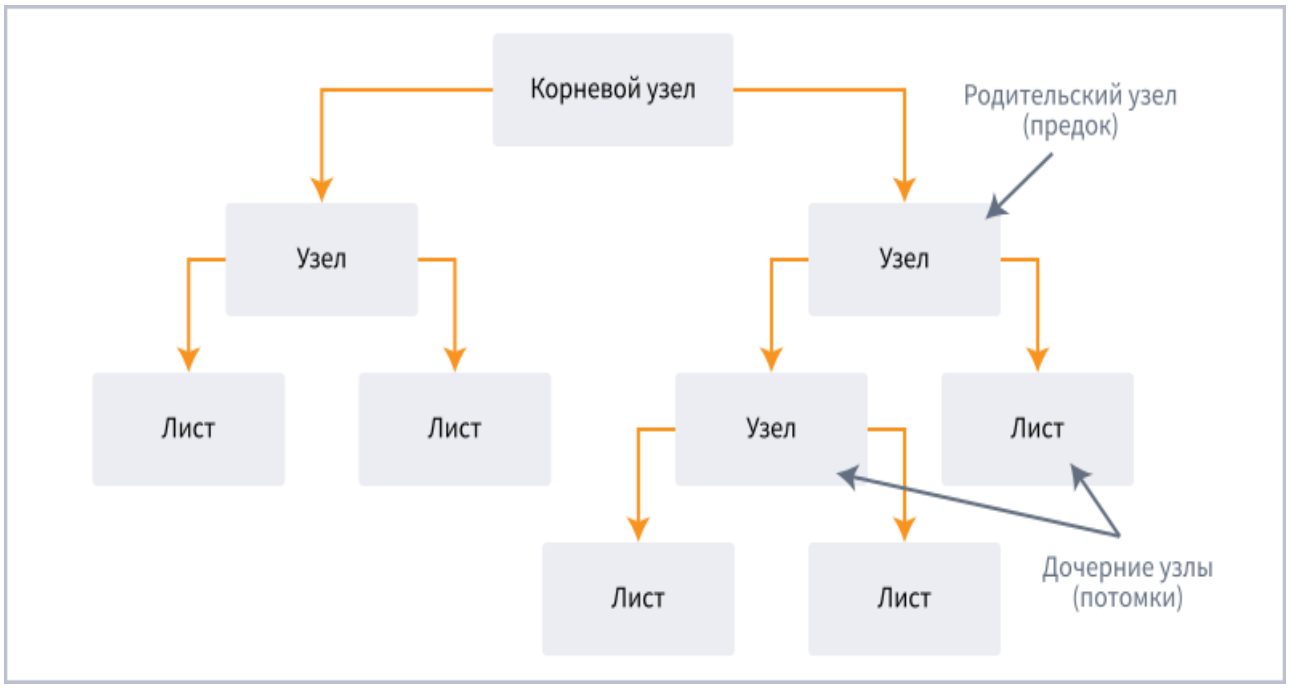


Рисунок 2.6 — Структура дерева прийняття рішень

Метод дерев прийняття рішень для задач класифікації полягає у тому, щоб здійснити розбиття вхідних даних на групи до тих пір, поки не будуть отримані однорідні (або майже однорідні) їх множини. Сукупність правил, котрі дають такий розподіл, дозволяє потім зробити прогнозування щодо приналежності до найбільш імовірного класу для нових даних[13].

Задача розподілу даних може бути вирішена за допомогою максимізації отримання інформації, що визначається наступною формулою:

$$\Delta_i = I(p) - \sum_{j=1}^{k_i} \frac{N(v_j) I(v_j)}{N} \tag{2.17}$$

де $k_i$ — значення атрибуту,

$N$ — кількість ітерацій роботи методу,

$v_j$ — стан розподілу відповідно значень атрибуту $j$,

$I(v_j)$ — функція, що вимірює домішок міток у вузлі.

Домішок міток (англ. node impurity) — це міра однорідності присвоєних міток у даному вузлі[14]. Існують різні реалізації для обчислення цього показника. Розглянемо основні з них.

Коефіцієнт Gini (англ. Gini impurity) — вимірює ймовірність неправильної класифікації нового екземпляра випадкової величини. Це є важливим, у випадку, коли цей екземпляр був класифікований випадковим чином, відповідно до розподілу міток класів з набору даних. Цей показник широко використовується для вирішення задач класифікації, та обчислюється за формулою

$$\sum_{i=1}^{C} f_i(1-f_i) \tag{2.18}$$

де $f_i$ — це частота присвоєння мітки $i$ у вузлі,

$C$ — кількість унікальних міток.

Інформаційна ентропія — це показник, який являє собою питому неозначеність на символ первинного алфавіту та характеризує алфавіт в цілому[15]. У формальному вигляді інформаційна ентропія визначається за формулою

$$\sum_{i=1}^{C} -f_i \log(f_i) \tag{2.19}$$

де $f_i$ — це розподіл ймовірностей присвоєння мітки $i$ у вузлі,

$C$ — кількість унікальних міток.

Основною перевагою побудови класифікатора за допомоги методу дерев прийняття рішень є те, що він не потребує багато обчислювальних ресурсів та здатний надзвичайно швидко проводити класифікацію невідомих екземплярів даних.

У полі рекомендаційних систем дерева рішень можуть застосовуватись для побудови моделі. Однією з ключових можливостей є використання характерних ознак даних для побудови дерева рішень, яке робить класифікацію усіх змінних, що приймають участь у моделюванні поведінки користувача системи. Автори у своїй праці[16] використали цю ідею для реалізації дерева рішень на основі семантичних даних про уподобання користувача. Слід зазначити, що такий підхід є цікавим з

теоретичної точки зору, проте точність роботи даної системи, як повідомляють автори, є гіршою, ніж при використанні інших методів класифікації.

### 2.5.7 Сингулярний розклад

Сингулярний розклад (SVD) — це потужна методика для зменшення розмірності. Він є особливою реалізацією матричної факторизації. Цей метод розв'язує проблему пошуку нижчої розмірності у просторі властивостей об'єктів, де нові властивості являють собою "концептуальні поняття", а характеристика кожного поняття у контексті колекції може бути обчислена.

Базується алгоритм SVD на теоремі про сингулярний розклад матриці[17]. В теоремі йдеться про те, що для будь-якої матриці $A_{mxn}$ існує розклад у добуток трьох матриць $U, D, V^T$:

$$A = U * D * V^T \tag{2.20}$$

Матриця $U$ інтерпретується як матриця прихованих ознак користувача, матриця $V$ — як матриця прихованих ознак предмета рекомендації, матриця $D$ являє собою діагональну матрицю, яка містить невід'ємні сингулярні числа розташовані у порядку спадання. Ідея матричної факторизації зображена на рисунку (2.7)

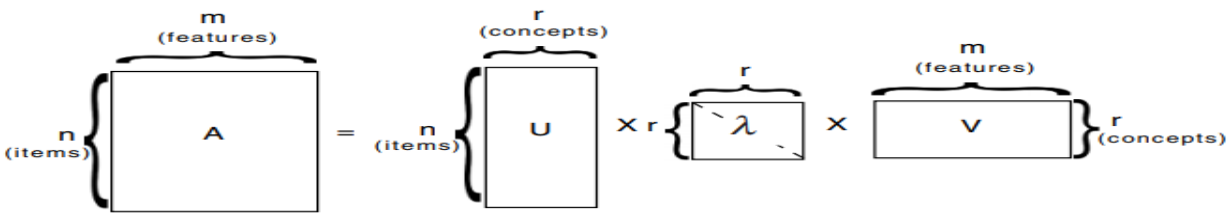


Рисунок 2.7 — Сингулярний розклад матриці

Алгоритм відносно простий, але він дозволяє не тільки передбачати оцінки. За його допомогою можна оцінюючи історію, виявляти приховані інтереси користувачів.

Історія використання SVD у якості інструмента для покращення рекомендацій відома вже давно. Автор у своїй праці [18] описує два різних методи використання сингулярного розкладу у контексті рекомендаційних систем.

Перший спосіб використання SVD — це виявлення латентних відносин між користувачами та продуктами. Для досягнення цієї мети, автор спочатку заповнює відсутні оцінки у матриці середнім значенням рейтингу по базі, і далі проводить нормалізацію, обчислюючи середнє значення для кожного користувача. Отримана матриця розкладається за допомогою SVD, і отримана декомпозиція використовується вже безпосередньо для обчислення прогнозів.

Другий спосіб полягає у тому, щоб використовувати SVD для створення низьковимірного простору для поліпшення роботи класифікатора, що працює за алгоритмом найближчих сусідів.

Однією з великих переваг використання SVD в області розв’язання проблем рекомендації є можливість обчислювати апроксимовану декомпозицію. Це дозволяє приймати рейтинги нових користувачів до існуючої моделі, без її перерахунку. Ця ідея була пізніше формалізована та розширена авторами публікації[19] в онлайн-моделі SVD.

Наразі, з розвитком чисельних методів обчислень, використовуються різні варіанти алгоритмів матричної факторизації, такі як NNMF. Їх основна ідея полягає у розбитті матриці оцінок користувачів на дві матриці: матрицю ознаків користувачів та матрицю ознаків елементів. Ці алгоритми працюють краще за SVD у випадку обробки відсутніх значень у матриці оцінок. Проте такий недолік можна усунути, попередньо замінивши нулі у матриці оцінок середнім значенням рейтингу по базі.

Основна проблема використання методів матричної факторизації полягає у тому, вони схильні до перенавчання. Проте, вже існують інші варіанти даного методу, наприклад NNMF, який дозволяє уникнути цієї проблеми. С. Рендал та Л.

Шмідт у своїй праці[20] повідомляють про свій досвід використання NNMF для онлайн-рекомендацій.

## 2.6 Колаборативна фільтрація

Колаборативна фільтрація – це механізм, який генерує рекомендації, що базуються на моделі попередньої поведінки користувача. Ця модель може бути побудована виключно на основі поведінки даного користувача або, що більш ефективно, з урахуванням поведінки інших користувачів з подібними характеристиками[21].

Колаборативна фільтрація використовує два різних типи вхідних даних: безліч користувачів і безліч об'єктів інтересу. Відносини між користувачами і об'єктами інтересу зазвичай виражаються за допомогою оцінок, наданих користувачами, і використовуються для прогнозування оцінок, які користувач міг би поставити неоціненим об'єктів інтересу. Якщо припустити, що користувач в даний час взаємодіє з коллаборативной системою рекомендацій, то система повинна ідентифікувати найближчих сусідів, тобто користувачів з аналогічною поведінкою, а потім екстраполювати з рейтингів схожих користувачів на рейтинг даного користувача.

Два основних підходи до колаборативної фільтрації – це орієнтація на користувача та орієнтація на об’єкт рекомендацій. Обидва варіанти передбачають, в якою мірою користувач буде цікавитися об'єктами, які ще не були ним оцінені. орієнтація на користувача ідентифікує його найближчих сусідів і на основі цих даних  обчислює прогноз користувача для певного об’єкта інтересу. На відміну від орієнтації на об’єкт рекомендацій, де на основі елементів для поточного об'єкта шукаються “сусіди”, які отримали аналогічні рейтинги.

Термін “проблема холодного старту” відноситься до ситуації, коли виникає необхідність надання початкових оцінок до того, як алгоритм зможе визначити

релевантні рекомендації. Така проблема характерна як для алгоритмів колаборативної фільтрації, так і для рекомендацій на основі вмісту. Користувачі при даному підході повинні оцінювати набір елементів, перш ніж алгоритм зможе визначити найближчих сусідів. При використанні рекомендаційних алгоритмів на основі контенту користувач також повинен вказати цікаві йому об'єкти до того моменту, коли алгоритм буде здатний визначати елементи, схожі на вже оцінені користувачем.

## 2.7 Адаптація моделі матричної факторизації для розв’язку задачі

Отже, щоб передбачити оцінку капітана команди $C$ до гравця $P$, необхідно розглянути деякий вектор $p_c$ для даного капітана та вектор $q_p$ для даного гравця. Скалярний добуток цих векторів і буде потрібною нам оцінкою

$$\bar{r} = \langle p_c, q_p \rangle \tag{2.21}$$

Проте, неможливо знайти сингулярний розклад, не знаючи самої матриці. Але можна скористатись цією ідеєю, та розробити модель яка буде працювати схожим чином. Для цього позначимо рейтинг гравця $\bar{r}$ через базові предиктори

$$\bar{r}_{c,p} = \mu + b_c + b_p + v_p^T u_c \tag{2.22}$$

де $\mu$ — середнє значення рейтингу по базі,

$b_c$ — базові предиктори капітанів команд,

$b_p$ — базові предиктори гравців,

$v_p$ — вектор факторів, що характеризує гравця,

$u_c$ — вектор факторів, що характеризує капітана команди.

З урахуванням формули (2.22), задача приймає наступний вигляд: необхідно знайти такі значення предикторів, які найкраще наближують величину $\overline{r_{c,p}}$ .

Найкращими будуть ті предиктори, які дають мінімальну похибку, що визначається за формулою

$$F(\mu, b_c, b_p, v_p, u_c) = \sum_{(c,p)\in D}(r_{c,p} - \overline{r_{c,p}})^2 = \sum_{(c,p)\in D}(r_{c,p} - \mu - b_c - b_p - v_p^T u_c)^2 \quad (2.23)$$

Функцію $F$ можна мінімізувати за допомогою методу градієнтного спуску. Тобто, взяти часткові похідні для кожного аргументу та рухатись у напрямку протилежному до напрямку цих часткових похідних. Для того, щоб уникнути ефекту перенавчання алгоритму, потрібно додати до формули (2.4.4) коефіцієнт регуляризації $\lambda$ [22]. Отже, задача мінімізації виглядатиме наступним чином:

$$\begin{aligned} b_*, q_*, p_* = \arg\min \sum_{(c,p)\in D}(r_{c,p} - \mu - b_c - b_p - v_p^T u_c)^2 + \\ \lambda(\sum_c b_c^2 + \sum_p b_p^2 + \|q_p\|^2 + \|p_c\|^2) \end{aligned} \quad (2.24)$$

Розв’язати цю задачу можна за допомогою методу градієнтного спуску, ітераційна формула виглядає наступним чином:

$$w_i = w_i - \gamma \frac{\partial F(w_i)}{w_i} \quad (2.25)$$

де $w_i$ — аргумент функції похибки,

$\gamma$ — коефіцієнт швидкості навчання алгоритму.

Враховуючи формули (2.24) — (2.27) правила для мінімізації параметрів функції похибки методом градієнтного спуску матимуть такий вигляд:

$$\begin{aligned} b_c &= b_c + \gamma(e_{c,p} - \lambda b_c), \\ b_p &= b_p + \gamma(e_{c,p} - \lambda b_p), \\ q_{p,j} &= q_{p,j} + \gamma(e_{c,p} p_{c,j} - \lambda q_{p,j}), \\ p_c &= p_c + \gamma(e_{c,p} p_{c,j} - \lambda p_{c,j}) \\ \mu &= \mu + \gamma e_{c,p} \mu \end{aligned} \quad (2.26)$$

для всіх $j$, де $e_{c,p} = r_{c,p} - \overline{r_{c,p}}$ — похибка на даному тестовому наборі даних.

## 2.8 Метрики для оцінювання роботи алгоритму

Метрики дають змогу оцінити різницю між порахованою оцінкою та реальною оцінкою. Найчастіше в області рекомендаційних систем використовуються такі метрики, як середня абсолютна похибка (MAE) та нормалізована середня абсолютна похибка (RMSE).

Середню абсолютну похибку можна обчислити як абсолютну різницю між передбаченням алгоритму та реальними оцінками:

$$\left|\overline{E}\right| = \frac{\sum_{i}^{N} \left|r_i - \overline{r_i}\right|}{N} \tag{2.27}$$

Незважаючи на деякі обмеження при оцінці систем, орієнтованих на рекомендації певної кількості об'єктів, простота обчислення та статистичні властивості зробили цю метрику однією з найпопулярніших для оцінки рекомендаційних систем.

Отримана за допомогою минулої метрики середньоквадратична похибка, має вагомий внесок при великих помилках в прогнозах:

$$\left|\overline{E}\right| = \sqrt{\frac{\sum_{i}^{N} \left|r_i - \overline{r_i}\right|}{N}} \tag{2.28}$$

Основна причина використання метрики RMSE полягає в тому, що помилки, виявлені за допомогою цієї метрики, можуть надати мати більш сильний вплив на рішення користувача.

При розробці системи рекомендацій, особливо для рекомендацій, заснованих на зібраній інформації про активність користувачів, важливо пам'ятати, що потрібно оптимізувати не тільки одну метрику. Тобто для коректних рекомендацій в області кіберспорту, потрібно не тільки рекомендувати гравців з найвищий рейтингом. Слід, також звернути увагу на певні бізнес-правила, такі як врахування середнього рейтингу гравців у команді, географічне положення команди, мова спілкування.

## Висновки до розділу 2

У даному розділі було наведено історію виникнення рекомендаційних систем та інформацію про відомі компанії, які використовують рекомендаційні алгоритми. Також було наведено відомі методи аналізу інформації, що використовуються для розв’язку проблем формування рекомендацій.

# 3 ЗАСОБИ РОЗРОБКИ

Під час розробки програмного продукту, потрібно з відповідальністю підходити до вибору технологій та засобів для програмної реалізації. Найчастіше, вибір технологій залежить від особливостей предметної області, сховища даних, вже реалізованих модулів для потрібних обчислень та досвід самого розробника. Засоби для програмної реалізації обираються спираючись на вподобання розробника, підтримку обраних технологій та зручність написання, відлагодження та тестування програмного коду, швидкість роботи середовища. Так як програмний продукт складається з серверної частини, яку реалізовано на платформі .NET Core, та клієнтської, яку зроблено за допомогою бібліотеки веб-компонентів Angular, то середовищем для розробки було обрано Visual Studio Code. Сховище даних працює на сервері PostgreSQL, тому для адміністрування бази даних було обрано застосунок PgAdmin.

## 3.1 Середовище розробки Visual Studio Code

Середовище розробки Visual Studio Code – це легковісний, але багатофункціональний редактор програмного коду, який підтримує роботу з більш ніж 30 мовами програмування та форматами файлів, включаючи C#, C++, JavaScript, TypeScript, Python, Go [23]. Також він має підтримку системи контролю версій Git, інтерфейс керування хмарними службами Azure та Docker контейнерів. Додаток Visual Studio Code можна використовувати на комп’ютерах під управлінням Windows, macOS та Linux.

Редактор Visual Studio Code базується на Electron. Electron – це технологія побудови графічних інтерфейсів для персональних комп’ютерів з використанням

платформи Node.js та Chromium. Також, це середовище розробки має підтримку плагінів, які можна знайти на сайті Visual Studio Marketplace. Вони містять у собі доповнення до редактора, підтримку додаткових мов програмування, статичні аналізатори коду та налаштування до кольорових схем редактора коду.

Редактор програмного коду підтримує технологію IntelliSense. Вона забезпечує інтелектуальне доповнення на основі мовної семантики та аналізу програмного коду. Якщо ця служба знає можливі доповнення, то при введенні команд будуть з’являтися пропозиції щодо закінчення її назви. Подальше налаштування служби IntelliSense та прив’язку ключових клавіш можна провести у меню конфігурацій.

## 3.2 Система баз даних PostgreSQL

Середовище PostgreSQL – це відкрита система об'єктно-реляційних баз даних, яка використовує і розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші робочі навантаження.

База даних PostgreSQL була розроблена у 1986 році, як частина проекту POSTGRES в Каліфорнійському університеті в Берклі і має більш ніж 30 років активного розвитку на основній платформі[24]. Вона отримала репутацію завдяки перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, модульності та відданості спільноти розробників програмного забезпечення з відкритим кодом для послідовного надання ефективних та інноваційних рішень. PostgreSQL працює на всіх основних операційних системах, з 2001 року сумісний з ACID, і має потужні додатки, такі як популярний геопросторовий розширювач бази даних PostGIS.

Особливий підхід PostgreSQL має до написання функцій. Функції є блоками коду, що виконуються на сервері, а не на клієнті бази даних. Хоча вони можуть бути написані на мові SQL, проте реалізація додаткової логіки, такої як умовні переходи

та цикли виходить за рамки SQL та потребує використання певних розширень. Тому функції можна писати з використанням вбудованої мови PL/pgSQL, а також інших мов, таких як Python, Ruby, C++, Java, R.

## 3.3 Платформа .NET Core

Технологія .NET Core - це універсальна платформа розробки з відкритим кодом, яку підтримує корпорація Майкрософт і співтовариство .NET на сайті GitHub. Вона є кроссплатформеною, тобто підтримує Windows, macOS і Linux та може використовуватися для створення додатків для пристроїв, хмарних сервісів та Інтернету речей[25].

Платформа .NET Core дозволяє створювати додатки і бібліотеки на мовах C #, Visual Basic і F #. Ці мови вже інтегровані або можуть бути інтегровані в популярні текстові редактори та інтегровані середовища розробки, такі як Visual Studio, Visual Studio Code, Sublime Text і Vim.

Для розробки веб-серверних додатків можна використовувати ASP.NET Core Web API. Ця технологія представляє спосіб побудови програм ASP.NET, який спеціально орієнтований для роботи в стилі REST (Representation State Transfer). Ця архітектура передбачає застосування методів або типів запитів HTTP для взаємодії з сервером.

## 3.4 Платформа для розробки веб-застосувань Angular

Середовище Angular — це платформа для побудови веб-додатків. Вона дозволяє вести розробку новітніх, автономних та високопродуктивних застосувань Progressive Web Apps, мобільних додатків з використанням Apache Cordova, Ionic та

NativeScript[26]. Платформа генерує для написаних шаблонів програмний код, який є оптимізованим для сучасних віртуальних машин JavaScript, надаючи всі переваги рукописного коду з продуктивністю фреймворку.

Для керування Angular проектом використовується інтерфейс командного рядка Angular CLI. За його допомогою можна створити новий проект, додати зовнішню бібліотеку, налаштувати конфігурацію, запустити статичний аналізатор коду, скомпілювати програму та запустити модульні тести.

Для написання компонентів використовується TypeScript. TypeScript – це мова програмування, розроблена компанією Microsoft та виступає як інструмент для розробки веб-додатків, який розширює можливості JavaScript[27]. Він підтримує обернену сумісність з мовою JavaScript та компілюється в нього ж. Фактично, після компіляції програму на TypeScript можна запускати в будь-якому сучасному веб-браузері або використовувати сумісно з серверною платформою Node.js.

Серед особливостей мови TypeScript можна виділити явну статичну типізацію, підтримку класів, як в традиційних об’єктно-орієнтованих мовах та можливість підключення модулів.

## 3.5 Ведення журналу системних подій

Коли система потрапляє до замовника, вона починає працювати у його середовищі. Всі помилки, які виникають при роботі програми, повинні потрапляти до журналу подій, інакше розробникам без нього неможливо буде відтворити ту ситуацію, яка призвела до прояву дефекту системи. Відповідно, неможливим буде і усунення цього дефекту. Також такий журнал може бути корисним і для системного адміністратора, який може провести аналіз того, що відбувається при роботі з програмою на конкретному комп’ютері користувача.

Технічно, це завдання можна здійснити, додавши до програмного продукту функцію ведення системного журналу подій. Для того, щоб потім легко

орієнтуватися у журналі подій, треба мати чітке уявлення про те, чому та як пишуться такі журнали, чітко уявляти предметну область та задачу, яку автоматизує це програмне забезпечення. В цьому випадку слід розглянути структуру різних видів системних журналів:

— системний журнал у вигляді текстового файлу. Спосіб, при якому кожна подія записується у файлі окремим рядком. Використовується досить часто та є простим з точки зору реалізації. Відкриття такого журналу можна здійснити будь-яким текстовим редактором;

— системний журнал у вигляді XML(JSON) файлів. В такому випадку, кожна подія записується до файлу не одним рядком, а кількома. Такий журнал набагато складніший для аналізу, тому що кожна подія може мати набір додаткової інформації. Для перегляду такого журналу найчастіше використовуються спеціальні програми;

— системний журнал у вигляді таблиці у базі даних. Додатки, які використовують бази даних або самі є СУБД, досить часто використовують базу даних як сховище для журналу подій. Здебільшого це окрема таблиця бази даних, кожен рядок якої є окремою подією. Таке ведення журналу часто може негативно позначитися на загальній продуктивності бази даних, так як запис до бази даних є операцією, що потребує багато ресурсів. Платформа .NET Core має вбудовану підтримку ведення системного журналу подій, та гнучке налаштування його структури.

## 3.6 Тестування програмного забезпечення

Тестування програмного забезпечення — це процес перевірки відповідності між реальною та очікуваною поведінкою програми, що здійснюється на певному наборі тестів. Загалом, під тестуванням можна розуміти цілий комплекс заходів

щодо контролю якості системи, який включає в себе планування робіт, проектування тестів, виконання тестування та аналіз отриманих результатів.

Тестування проводять з метою:

— підвищення ймовірності того, що програмний засіб, який тестується, буде відповідати вимогам замовника;

— підвищення ймовірності того, що програмний засіб буде працювати коректно за будь-яких умов;

— надання актуальної інформації про поточний стан програмного продукту.

У сучасній практиці розробки програмного забезпечення розрізняють різні види тестування програмних продуктів:

— модульне тестування — перевіряє функціональність та шукає помилки у частинах програми, які є доступними та можуть бути протестовані окремо, наприклад класи, функції, модулі;

— інтеграційне тестування — перевіряє правильність взаємодії між компонентами системи;

— системне тестування — у ході цього заходу виявляються помилки, такі як неправильне використання ресурсів системи, непередбачені комбінації даних користувацького рівня, відсутня або невірна функціональність, незручність використання;

— операційне тестування — перевіряє чи система задовольняє вимогам користувачів та виконує свою роль у середовищі експлуатації, як це було визначено у бізнес-моделі системи.

### 3.6.1 Модульне тестування

Основна ідея модульного тестування полягає у тому, щоб писати тести, в яких перевіряється найменша “одиниця” коду. Модульні тести зазвичай написані на тій самій мові програмування, на якій написаний код основної програми. Вони

створюються безпосередньо для перевірки цього коду. Тобто, підсумувавши, можна сказати, що модульні тести — це код, який перевіряє коректність роботи іншого коду.

Під час написання модульних тестів, настає момент, коли вони перетворюються в деякі абсолютні вимоги до певних модулів коду на низькому рівні. Тобто, модульні тести можна вважати абсолютною специфікацією. Ця специфікація визначає, що за певних умов, з конкретним набором вхідних даних, має місце результат, який розробник повинен отримати від цього модулю. У випадку сучасних об'єктно-орієнтованих мов програмування, цим модулем являється клас.

Наразі існує дві основних причини проведення модульного тестування програмних додатків: покращення дизайну коду та створення автоматизованого набору регресійних тестів.

Перша причина з'являється у ході написання самих тестів, коли потрібно ізолювати найменшу одиницю коду. Такі заходи можуть призвести до того, що розробник матиме можливість виявити проблеми в структурі самого програмного коду. Можлива така ситуація, що ізолювання класу, який перевіряється, без включення його залежностей, буде досить важкою задачею. Це є проблема занадто зв'язаного коду.

З іншого боку, може виявитись, що базова функціональність, яку розробник намагається протестувати, розповсюджується на декілька різних модулів, що надводить на думку про помилки у проектуванні модулів. Також, потрібно навести той факт, що при написанні модульних тестів можна знайти реальну помилку у реалізації певної функціональності, оскільки вони змушують розробника думати про нестандартні варіанти використання та перевіряти різні набори вхідних даних.

Друга причина полягає у тому, щоб створити набір регресійних тестів, які будуть працювати як специфікація поведінки програмного забезпечення на низькому рівні. Ця причина постає тоді, коли розробник вносить зміни до програмного коду. Для того, щоб перевірити, чи внесені зміни не порушили роботу існуючої функціональності, і використовують регресійні тести.

### 3.6.2 Підхід TDD

Розробка через тестування (TDD) — це загальновідома практика написання програмних додатків. Підхід TDD полягає в тому, щоб перш ніж написати код для певної функції, спочатку потрібно написати тест, який служитиме специфікацією, тобто визначатиме, що буде робити даний код. Таким чином, модульне тестування керує процесом розробки програмного коду.

Цей процес є ітераційним: спочатку пишеться тест, потім реалізовується мінімальна функціональність для його проходження, і врешті, модифікація з метою реорганізації та очищення коду функції. Нижче наведено блок-схему даного підходу рисунок (3.1)

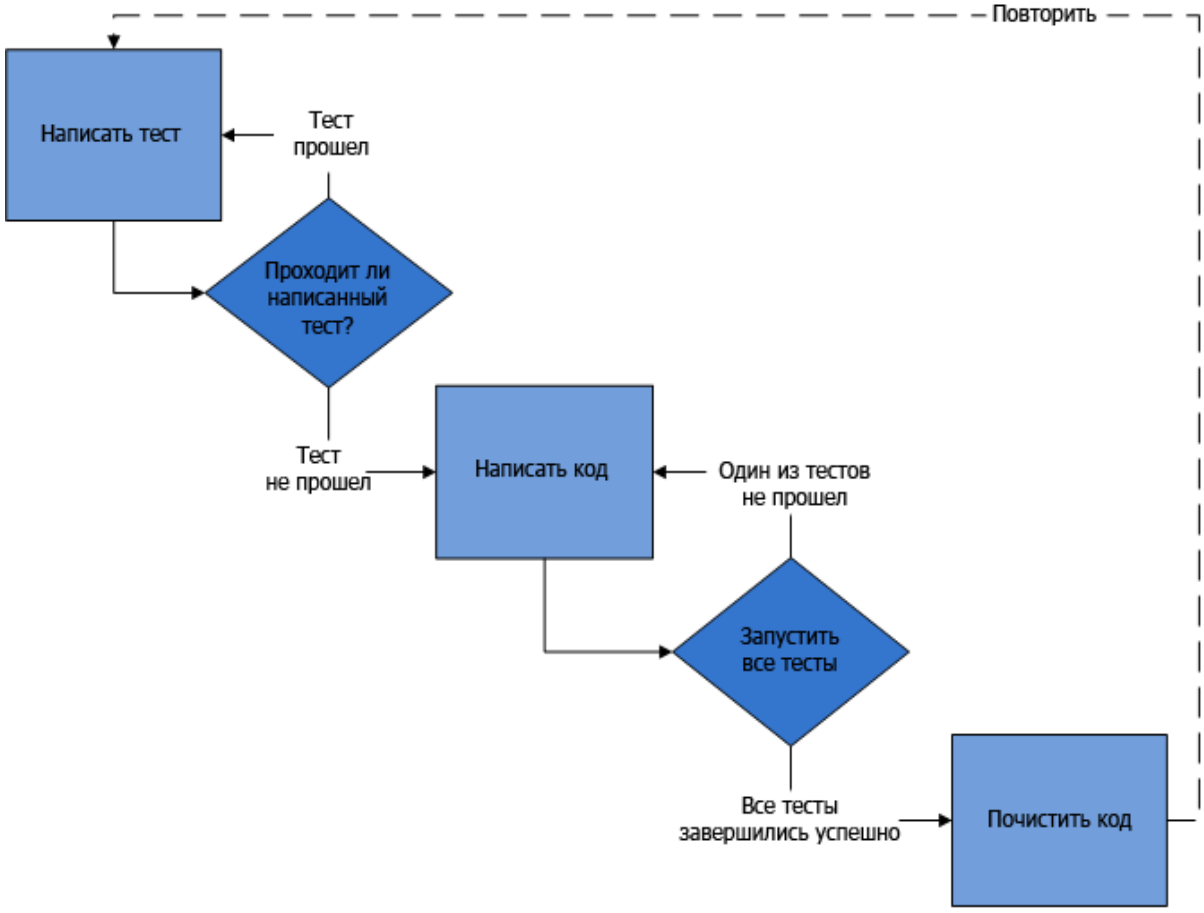


Рисунок 3.1 — Блок-схема підходу TDD

### 3.6.3 Тестування на платформі .NET Core за допомоги бібліотеки xUnit

У ході даної дипломної роботи використовується метод модульного тестування. Для проведення даного заходу, було використано бібліотеку xUnit.net[28].

Бібліотека xUnit.net — це безкоштовний інструмент для тестування модулів, написаних на платформі .NET Framework з відкритим вихідним кодом. Ця засіб підтримує такі мови програмування, як C#, F#, VB.NET, тощо. Також, xUnit.net інтегрується з такими плагінами для систем написання коду як Reshaper, CodeRush, TestDriven.NET та Xamarin.

При створенні проекту з тестами, слід правильно його йменувати. Однією з найкращих практик є спосіб найменування з використанням назви проекту, що буде тестуватися, тобто <Назва_Проекту>.Tests. Дотримуючись такого підходу, можна запускати проекти з тестами через вікно командного рядка, використовуючи шаблон *.Tests.dll.

Відповідно до правил найменування проектів з тестами, також існують і правила найменування для тестових методів. Загальноприйнятим способом є наступний: [Метод_що_тестується]_[Сценарій]_[Очікувана_поведінка]. Наприклад, потрібно протестувати клас CountriesController, який містить метод Get, що повертає список всіх країн команд. Відповідно до правил, тестовий клас потрібно назвати CountriesControllerTest з методом Get_WhenInvoke_ShouldReturnsAllCountries.

## Висновки до розділу 3

У даному розділі було розглянуто технології та підходи, які було використано при розробці рекомендаційної системи. Було наведено короткий опис кожної з технологій та розглянуто процес перевірки програмного коду.

# 4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В даному розділі міститься інформація про структуру програми, опис розроблених модулів та реалізація обраного алгоритму для розв’язку задачі.

## 4.1 Структура програмного продукту

Програмний продукт є клієнт-серверним додатком. Серверна частина реалізована на платформі .NET Core, клієнт – за допомогою технології побудови компонентів на Angular. Для забезпечення комунікації між сервером на клієнтом використовується архітектурний стиль взаємодії компонентів REST. Всі запити виконуються за допомогою HTTP-методів. Формат даних в якому приходять відповіді на запити є JSON. В якості сховища даних використовується реляційна база даних PostgreSQL. Схематично проект має такий вигляд (рисунок 4.1):

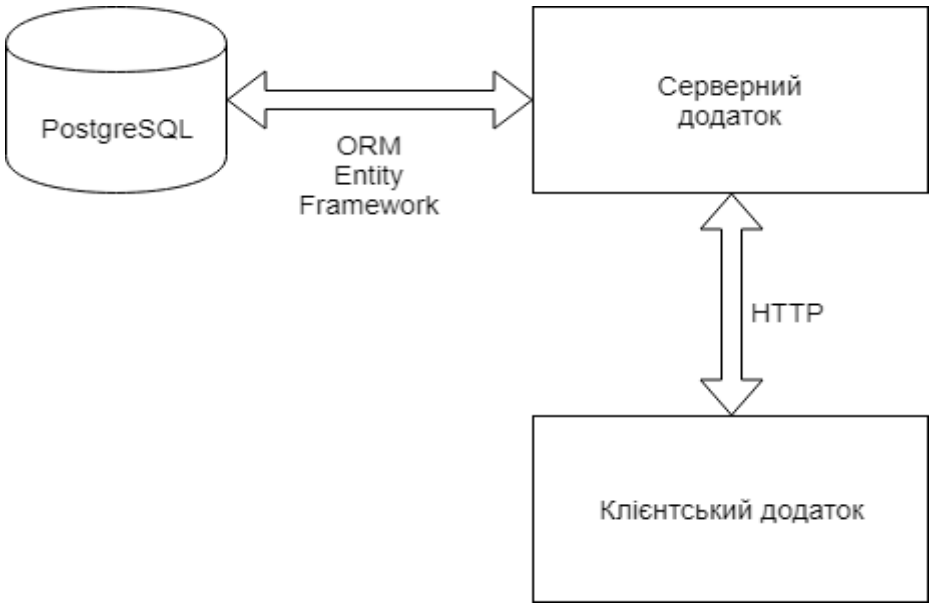


Рисунок 4.1 — Схема проекту

## 4.2 Реалізація алгоритму прорахунку рекомендацій

Алгоритм реалізовано у вигляді DLL-бібліотеки, що підключається до проекту. Вхідними даними до алгоритму є кількість ітерацій для навчання алгоритму, число факторів для гравця і капітана команди, швидкість навчання алгоритму, крок навчання алгоритму та коефіцієнт регуляризації. Вхідні параметри встановлюються аналітиком, згідно до його міркувань та особливостей набору даних.

Програмна реалізація містить у собі 2 класи. Model – це клас, який є представленням моделі вихідних даних для алгоритму. Фактично він є класом для даних, які зберігаються у формат JSON для подальшого використання параметрів алгоритму. Та клас Recommender який виконує налаштування параметрів моделі та її збереження для подальшого використання у серверній частині програми. Діаграма класів для модулю має вигляд (Рисунок 4.2):

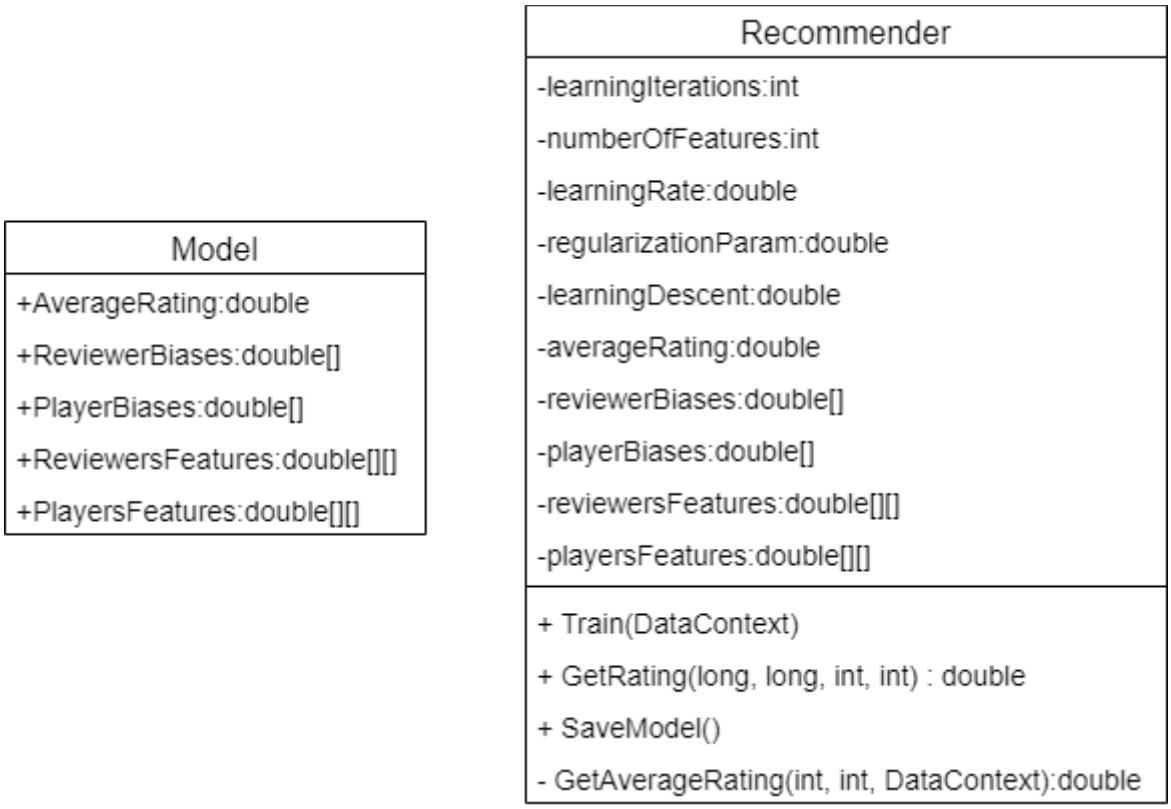


Рисунок 4.2 — Діаграма класів для модулю рекомендацій

## 4.3 Реалізація доступу до даних

Доступ до бази даних від серверної частини здійснюється за допомогою технології Entity Framework. Застосунок Entity Framework є продовженням технології Microsoft ActiveX Data та надає можливість працювати з базами даних через об’єктно-орієнтований код C#. Використання цієї технології має ряд суттєвих переваг:

— розробнику не потрібно думати про код доступу до даних;

— не потрібно знати про деталі роботи бази даних;

— не потрібно писати SQL-запити власноруч;

— можна використовувати технологію LINQ.

Розробник працює безпосередньо з класами C#, які є об’єктним представленням таблиць в базі даних. Для того, щоб мати змогу працювати з цими класами, треба створити модель бази даних. В рамках цієї дипломної роботи використовується підхід Database First до існуючої бази даних.

При написанні модулів доступу до даних, часто виникає проблема дублювання програмного коду запитів до сховища даних. Розв’язати цю проблему допомагає шаблон проектування Репозиторій.

Репозиторії — це класи або компоненти, які інкапсулюють логіку, необхідну для доступу до джерел даних. Вони централізують загальні функціональні можливості доступу до даних, забезпечуючи кращу обслуговуваність і роз'єднуючи інфраструктуру або технологію, що використовується для доступу до баз даних з рівня домену. Це дозволяє зосередитися на логіці збереження даних, а не на тонкощах доступу до даних.

Загалом, репозиторій дозволяє заповнювати дані в оперативній пам’яті, які надходять з бази даних у якості об’єктів доменних класів. Коли об’єкти знаходяться в оперативній пам’яті, їх можна змінити та зберегти назад до бази за допомоги механізму транзакцій.

При використанні цього шаблону можна легко тестувати програму за допомогою модульних тестів, так як відбувається абстрагування від конкретного сховища даних. Тому, що з'єднання з базою даних можуте вийти з ладу, і що більш важливо, виконання сотень тестів на базі даних може зайняти багато часу. Також, записи у базі даних можуть змінюватись у процесі тестування та впливати на результати самих тестів.

Що стосується проблеми абстрагування сховища даних при модульному тестуванні, то логіка працює на предметних областях доменів в пам'яті. Передбачається, що клас репозиторію доставив їх, тому можна сконцентруватися на тестуванні правильності роботи логіки програми, а не особливостей самого сховища.

## Висновки до розділу 4

У даному розділі було наведено структуру побудованого програмного додатку. Було описано реалізацію алгоритму рекомендацій та розглянуто тонкощі написання програмного коду доступу до даних на платформі .NET Core.

# 5 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

Даний розділ містить інформацію про інсталяцію та запуск програмного продукту, системні вимоги та довідкову інформацію для користувача.

## 5.1 Системні вимоги

Для запуску та коректної роботи програмного забезпечення комп’ютер повинен мати встановлене середовище .NET Core Runtime версії 2.x. Так як середовище є крос-платформеним, то підтримуються комп’ютери на базі операційних систем Windows, macOS та Linux. Також потрібно мати один із веб-браузерів: Chrome — остання версія, Firefox — остання версія, Internet Explorer — версія не нижче 11, Safari — остання версія.

## 5.2 Запуск програми

Для того, щоб запустити програму потрібно перейти в директорію с виконуваним додатком та набрати в консолі команду “dotnet run” (Рисунок 5.1):

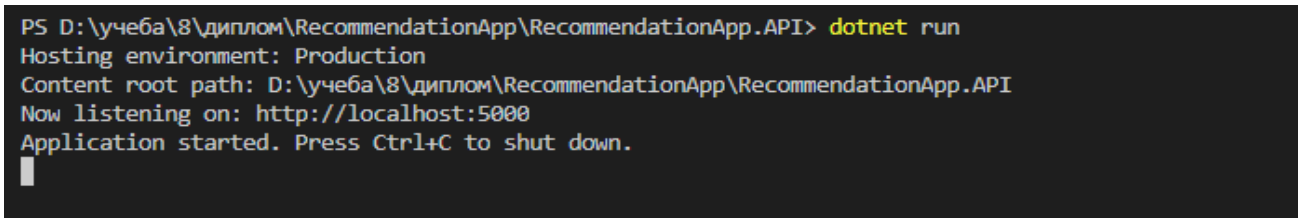


Рисунок 5.1 — Запуск програми у вікні терміналу

Потім відкриється вікно веб-браузера з інтерфейсом програми(Рисунок 5.2):

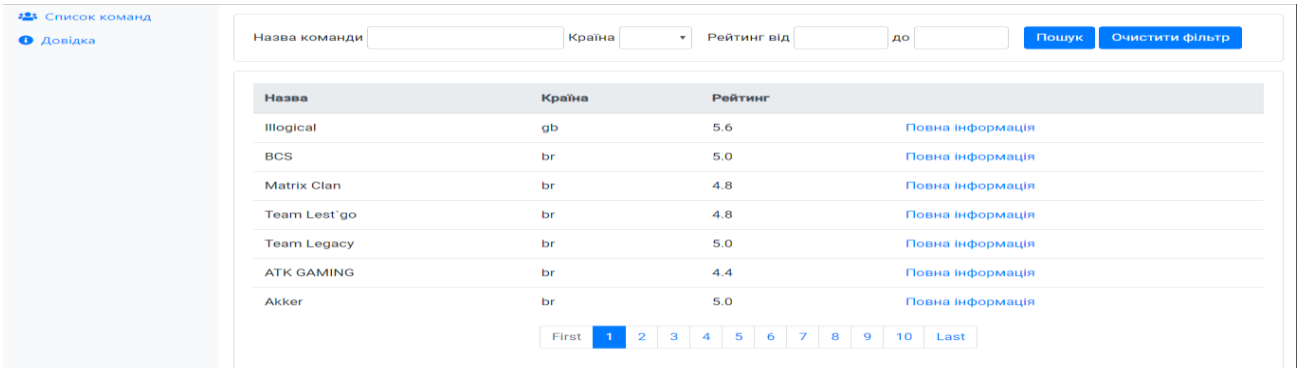


Рисунок 5.2 — Веб-інтерфейс програми

Веб-інтерфейс містить форму пошуку, список команд та головну інформацію про них.

## 5.3 Робота користувача з програмним забезпеченням

Параметризований пошук команд здійснюється за допомогою форми пошуку. Пошук здійснюється за такими критеріями: назва команди, країна та рейтинг. Наприклад, можна здійснити пошук всіх команд з Данії, для цього потрібно обрати країну зі списку та натиснути на “Пошук” (Рисунок 5.3):

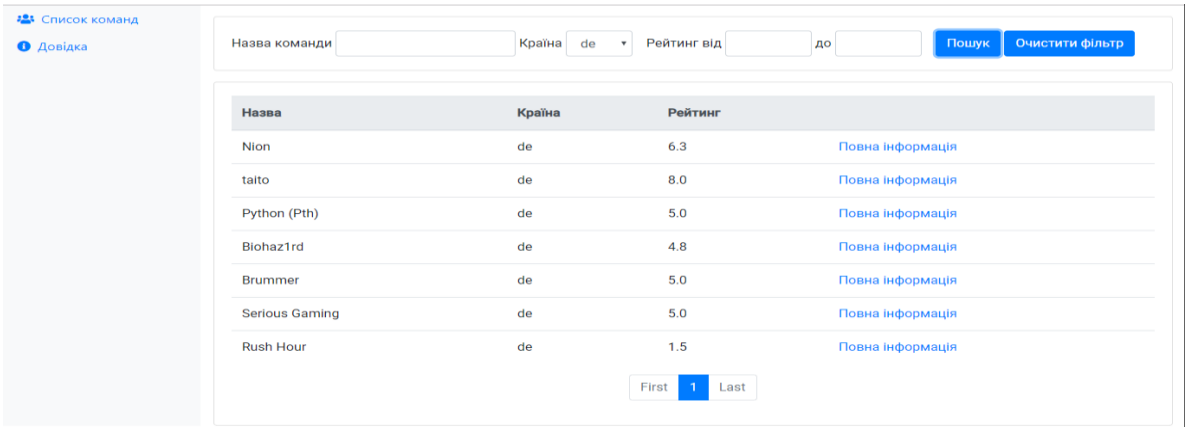


Рисунок 5.3 — Приклад роботи з формою пошуку команд

Для того, щоб переглянути інформацію про команду, потрібно обрати команду в таблиці та натиснути на посилання “Повна інформація”. Відкриється сторінка, яка містить інформацію про команду, її склад, статистику та список рекомендованих гравців (Рисунок 5.4):

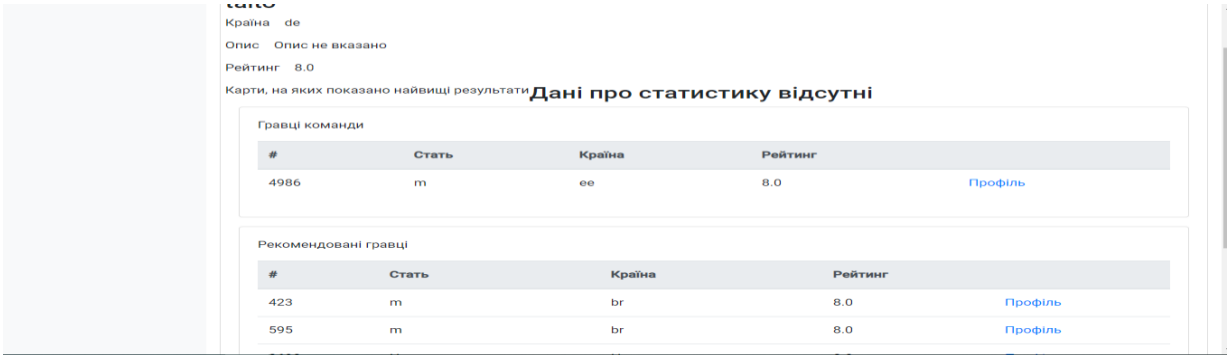


Рисунок 5.4 — Сторінка команди

Для перегляду інформації про гравця, потрібно обрати гравця у таблиці та натиснути на посилання “Профіль”. Тоді відкриється сторінка гравця (Рисунок 5.5):

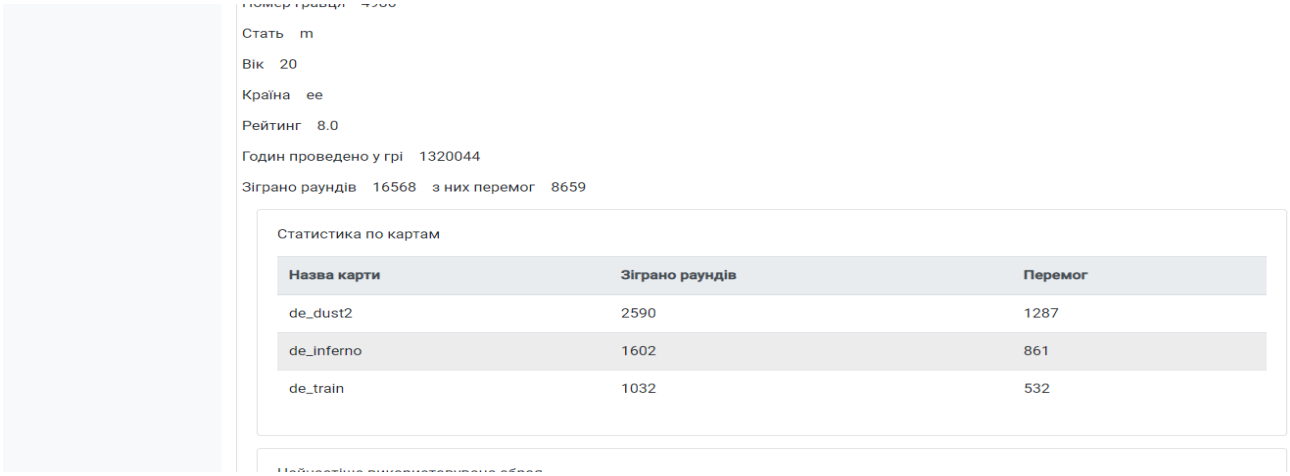


Рисунок 5.5 — Сторінка гравця

Вона містить персональну інформацію про гравця, статистику матчів та зброю. Якщо в базі даних ця інформація не заповнена, то напроти блоку зі статистикою буде написано повідомлення про те, що дані відсутні.

# ВИСНОВКИ

У ході даної дипломної роботи було розроблено систему для рекомендацій нових гравців.

Програмний продукт реалізовано у вигляді клієнт-серверного додатку. При розробці використовувались технології .NET Core та модуль для побудови веб-інтерфейсів Angular, у якості сховища даних було використано СКБД PostgreSQL.

Було наведено теоретичні відомості щодо існуючих алгоритмів рекомендацій та систем до яких вони інтегровані.

Детально розглянуто побудову додатків з клієнт-серверною архітектурою, її особливості, переваги та недоліки. У якості платформи для розробки серверної частини було використано ASP.NET Core.

Також було розглянуто платформу для побудови клієнтської частини Angular, її особливості та переваги.

Було розглянуто тему перевірки правильності роботи програмного коду, його види та використані додатки для тестування.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Николенко С.А. Рекомендательные системы. СПб: Изд-во Центр Речевых Технологий, 2012. 53 с.

2. Goldberg D., Nichols D., Oki B. M., Terry D. Using collaborative filtering to weave an information Tapestry // Special issue on information filtering, 1992. Vol. 35, Issue 12 P. 61-70.

3. Королева Д.Е., Филиппов М.В. Анализ алгоритмов обучения коллаборативных рекомендательных систем. Инженерный журнал: наука и инновации, 2013, вып. 6 [Електронний ресурс]. — Режим доступу до ресурсу: http://engjournal.ru/articles/816/816.pdf

4. Resnick P., Iakovou N., Sushak M., Bergstrom P., Riedl J. GroupLens: An Open Architecture for Collaborative Filtering of Netnews // Proceeding 1994 Computer Supported Cooperative Work Conference, 1994. P. 175-186.

5. Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor. Recommender Systems Handbook. — Springer, 2011. — 845 p.

6. O'Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: An evaluation of the performance of collaborative filtering. In Proceedings of the 14th Irish International Conference on Artificial Intelligence and Cognitive Science (AICS'03) pp. 164–168 (2003).

7. DreamTeam — The Ultimate Teambuilding and Skill-Growing Platform [Електронний ресурс]. — Режим доступу до ресурсу: https://dreamteam.gg

8. Гмурман В.Е. Руководство к решению задач по теории вероятностей и математической статистике. М.: Высш. школа, 1979, 400 с.

9. Кононюк А.Ю. Нейронні мережі і генетичні алгоритми. — К.: "Корнійчук", 2008. — 446 с.

10. McCulloch W.S., Pitts W. A logical calculus of the ideas immanent in nervous activity // Bulletin of Mathematical Biophysics, 1943.–№ 5.–P.115-133.

11. М.А. Новотарський, Б.Б. Нестеренко. Штучні нейронні мережі: обчислення // Праці Інституту математики НАН України. – Т50. – Київ: Ін-т математики НАН України, 2004. – 408 с.

12. Michael J. Pazzani, Daniel Billsus. Content-based Recommendation Systems. Rutgers University, New Brunswick, NJ, 2007. P. 341.

13. Brett Lantz. Machine Learning with R. Packt Publishing, Birmingham - Mumbai, 2013.

14. Decision Trees — MLib — Spark 1.3.0 Documentation [Електронний ресурс] — Режим доступу: https://spark.apache.org/docs/1.3.0/mllib-decision-tree.html

15. А.Е. Кононюк Информациология. Общая теория информации. К.: “Освіта України”, 2011. Книга 2. 476 с.

16. Lam, S.K., Riedl, J.: Shilling recommender systems for fun and profit. In Proceedings of the 13th International World Wide Web Conference pp. 393–402 (2004).

17. O’Mahony, M.P., Hurley, N.J., Silvestre, G.C.M.: Promoting recommendations: An attack on collaborative filtering. In: A. Hameurlain, R. Cicchetti, R. Traunmuller (eds.) DEXA, Lecture Notes in Computer Science, vol. 2453, pp. 494–503. Springer (2002).

18. B. Sarwar, G. Karypis, J. Konstan, J. Reidl. Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems. Fifth International Conference on Computer and Information Science, 2002.

19. Massa, P., Avesani, P.: Trust-aware recommender systems. In: RecSys ’07: Proceedings of the 2007 ACM conference on Recommender systems, pp. 17–24. ACM, New York, NY, USA (2007).

20. S. Rendell, L. Schmidt-Thieme Pairwise Interaction Tensor Factorization for Personalized Tag Recommendation. WSDM Conference, 2010.

21. Ekstrand M. D., Riedl J. T., Konstan J. A. Collaborative Filtering Recommender Systems // Foundations and Trends in Human — Computer Interaction, 2011. Vol. 4, No. P. 81-173.

22. Y. Koren, R. Bell, C. Volinsky Matrix Factorization Techniques For Recommeder Systems. IEEE Computer Society, 2009.

23. Редактор Visual Studio Code [Електронний ресурс] — Режим доступу: https://code.visualstudio.com/docs

24. Документація серверу баз даних PostgreSQL [Електронний ресурс] — Режим доступу: https://www.postgresql.org/docs

25. Керівництво користувача по .NET Core [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/core/

26. Документація платформи для побудови веб-додатків Angular [Електронний ресурс] — Режим доступу: https://angular.io/docs

27. Документація до мови TypeScript [Електронний ресурс] — Режим доступу: https://www.typescriptlang.org/docs/home.html

28. Документація до бібліотеки тестування xUnit [Електронний ресурс] — Режим доступу: https://xunit.net/#documentation

29. Фаулер М. Архитектура корпоративных программных приложений : пер. С англ. / М. Фаулер – М. : Вильямс, 2006. – 544 с.

# ДОДАТОК А

Алгоритмізація та реалізація алгоритму аналізу інформації щодо пропозицій покращення статусу гравця/команди гравців

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5292_19Б

Аркушів 2

Київ 2019

| Позначення | Найменування | Примітки |
|---|---|---|
| Документація | | |
| УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5292_19Б | Записка.docx | Текстова частина дипломної роботи |
| Компоненти | | |
| УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5292_19Б | RecommendationApp.API.dll | Основний компонент програмного додатку системи рекомендацій |

# ДОДАТОК Б

Алгоритмізація та реалізація алгоритму аналізу інформації щодо пропозицій покращення статусу гравця/команди гравців

Текст програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5292_19Б

Аркушів 7

Київ 2019

Текст програми алгоритму рекомендацій

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;

namespace ConsoleApp2
{
    public class Recommender
    {
        private int learningIterations;
        private int numberOfFeatures;
        private double learningRate;
        private double regularizationParam;
        private double learningDescent;

        private double averageRating;
        private double[] reviewerBiases;
        private double[] playersBiases;
        private double[][] reviewersFeatures;
        private double[][] playersFeatures;

        public Recommender()
        {
            learningIterations = 30;
            numberOfFeatures = 20;
            learningRate = 0.005;
            regularizationParam = 0.02;
            learningDescent = 0.99;
        }

        public void Train(dreamteamContext context, List<long>
reviewerIndexToId, List<long> playerIndexToId, int numReviewers, int numPlayers)
        {
            reviewerBiases = new double[numReviewers];
            playersBiases = new double[numPlayers];

            reviewersFeatures = new double[numReviewers][];
            playersFeatures = new double[numPlayers][];

            Random rand = new Random();

            for (int reviewerIndex = 0; reviewerIndex < numReviewers;
reviewerIndex++)
            {
                reviewersFeatures[reviewerIndex] = new double[numberOfFeatures];
                for (int featureIndex = 0; featureIndex < numberOfFeatures;
featureIndex++)
                {
                    double val = rand.NextDouble();
                    reviewersFeatures[reviewerIndex][featureIndex] = val;

Console.WriteLine($"reviewersFeatures[{reviewerIndex}][{featureIndex}] =
{val}");
                }
            }
```

```
            for (int playerIndex = 0; playerIndex < numPlayers; playerIndex++)
            {
                playersFeatures[playerIndex] = new double[numberOfFeatures];
                for (int featureIndex = 0; featureIndex < numberOfFeatures;
featureIndex++)
                {
                    double val = rand.NextDouble();
                    playersFeatures[playerIndex][featureIndex] = val;

Console.WriteLine($"playersFeatures[{playerIndex}][{featureIndex}] = {val}");
                }
            }

            averageRating = GetAverageRating(reviewerIndexToId.Count,
playerIndexToId.Count, context, reviewerIndexToId, playerIndexToId);
            double dotProduct;
            double predictedRating;
            double error;

            double rmse = 0;
            double mae = 0;

            int count = 0;

            for (int i = 0; i < learningIterations; i++)
            {
                for (int reviewerIndex = 0; reviewerIndex <
reviewerIndexToId.Count; reviewerIndex++)
                {
                    long reviewerId = reviewerIndexToId[reviewerIndex];
                    for (int playerIndex = 0; playerIndex <
playerIndexToId.Count; playerIndex++)
                    {
                        long playerId = playerIndexToId[playerIndex];
                        var rating = context.Reviews.Where(r => r.ProfileId ==
playerId && r.ReviewerProfileId == reviewerId).Select(r =>
r.Rate.GetValueOrDefault())
                            .SingleOrDefault();
                        if (rating != 0)
                        {
                            dotProduct =
GetDotProduct(reviewersFeatures[reviewerIndex], playersFeatures[playerIndex]);

                            predictedRating = averageRating +
reviewerBiases[reviewerIndex] + playersBiases[playerIndex] + dotProduct;

                            if (double.IsNaN(predictedRating))
                            {
                                throw new Exception("NaN rating");
                            }

                            error = rating - predictedRating;
                            rmse += Math.Pow(error, 2);
                            mae += Math.Abs(error);
                            count++;

                            averageRating += learningRate * (error -
regularizationParam * averageRating);
                            reviewerBiases[reviewerIndex] += learningRate *
(error - regularizationParam * reviewerBiases[reviewerIndex]);
```

```
                            playersBiases[playerIndex] += learningRate * (error
- regularizationParam * playersBiases[playerIndex]);

                            for (int featureIndex = 0; featureIndex <
numberOfFeatures; featureIndex++)
                            {
                                reviewersFeatures[reviewerIndex][featureIndex]
+= learningRate * (error * playersFeatures[playerIndex][featureIndex]
                                    - regularizationParam *
reviewersFeatures[reviewerIndex][featureIndex]);
                                playersFeatures[playerIndex][featureIndex] +=
learningRate * (error * reviewersFeatures[reviewerIndex][featureIndex]
                                    - regularizationParam *
playersFeatures[playerIndex][featureIndex]);
                            }
                        }
                    }
                }

                rmse = Math.Sqrt(rmse / count);
                mae /= count;
                learningRate *= learningDescent;
                Console.WriteLine($"{i}\t{rmse}\t{mae}");
                SaveModel();
            }
        }

        private double GetDotProduct(double[] v1, double[] v2)
        {
            return v1.Zip(v2, (a, b) => a * b).Sum();
        }

        private double GetAverageRating(int numReviewers, int numPlayers,
dreamteamContext context, List<long> reviewerIndexToId, List<long>
playerIndexToId)
        {
            double sum = 0;
            int count = 0;
            for (int i = 0; i < numReviewers; i++)
            {
                for (int j = 0; j < numPlayers; j++)
                {
                    long reviewerId = reviewerIndexToId[i];
                    long playerId = playerIndexToId[j];
                    var rating = context.Reviews.Where(r => r.ProfileId ==
playerId && r.ReviewerProfileId == reviewerId).Select(r =>
r.Rate.GetValueOrDefault())
                        .SingleOrDefault();
                    if (rating != 0)
                    {
                        sum += rating;
                        count++;
                    }
                }
            }

            Console.WriteLine($"Average rating = {sum / count}");
            return sum / count;
        }
```

```
        public double GetRating(long reviewerId, long playerId, List<long>
reviewerIndexToId, List<long> playerIndexToId)
        {
            int reviewerIndex = reviewerIndexToId.IndexOf(reviewerId);
            int playerIndex = playerIndexToId.IndexOf(playerId);
            double ratingToPredict = averageRating +
reviewerBiases[reviewerIndex] + playersBiases[playerIndex] +
GetDotProduct(playersFeatures[playerIndex],
                reviewersFeatures[reviewerIndex]);

            return ratingToPredict;
        }

        public void SaveModel()
        {
            Model model = new Model
            {
                AverageRating = averageRating,
                PlayerBiases = playersBiases,
                PlayersFeatures = playersFeatures,
                ReviewerBiases = reviewerBiases,
                ReviewersFeatures = reviewersFeatures
            };
            DataContractJsonSerializer serializer = new
DataContractJsonSerializer(typeof(Model));
            using(FileStream fs = new FileStream("model.json", FileMode.Create))
            {
                serializer.WriteObject(fs, model);
            }
        }

        public void LoadModel(string filePath)
        {
            DataContractJsonSerializer serializer = new
DataContractJsonSerializer(typeof(Model));
            Model model;
            using(FileStream fs = new FileStream(filePath, FileMode.Open))
            {
                model = (Model)serializer.ReadObject(fs);
            }

            if (model != null)
            {
                averageRating = model.AverageRating;
                playersBiases = model.PlayerBiases;
                playersFeatures = model.PlayersFeatures;
                reviewerBiases = model.ReviewerBiases;
                reviewersFeatures = model.ReviewersFeatures;
            }
        }
    }

    [DataContract]
    public class Model
    {
        [DataMember]
        public double AverageRating { get; set; }
        [DataMember]
        public double[] ReviewerBiases { get; set; }
        [DataMember]
```

```
        public double[] PlayerBiases { get; set; }
        [DataMember]
        public double[][] ReviewersFeatures { get; set; }
        [DataMember]
        public double[][] PlayersFeatures { get; set; }
    }
}
```

Текст програми класу, який використовує модуль формування рекомендацій

```
using System;
using System.Collections.Generic;
using System.Linq;
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using RecommendationApp.API.Data;
using RecommendationApp.API.Dto;
using RecommendationApp.API.Helpers;

namespace RecommendationApp.API.Controllers
{
    [ApiController]
    [Route("api/teams/{teamId}/[controller]")]
    public class RecommendedPlayersController : ControllerBase
    {
        private IPlayerRepository _playerRepository;
        private ITeamRepository _teamRepository;
        private IMapper _mapper;
        private ILogger _logger;

        public RecommendedPlayersController(IPlayerRepository playerRepository,
ITeamRepository teamRepository,
         IMapper mapper, ILogger<RecommendedPlayersController> logger)
        {
            _playerRepository = playerRepository;
            _teamRepository = teamRepository;
            _mapper = mapper;
            _logger = logger;
        }

        [HttpGet]
        public IActionResult Get(int teamId)
        {
            _logger.LogInformation("Getting recommended players for team with id
= {id}", teamId);
            var team = _teamRepository.GetTeam(teamId);

            if(team == null)
            {
                _logger.LogWarning("Team with id = {id} not found", teamId);
                return BadRequest("Team not found");
            }

            var players = _playerRepository.GetRecommendedPlayers(teamId);
            var playersToReturn =
_mapper.Map<IEnumerable<PlayerForListDto>>(players);

            foreach(var playerDto in playersToReturn)
```

```
            {
                var profile = _playerRepository.GetPlayer(playerDto.Id);
                if(profile != null)
                {
                    playerDto.Gender = profile.Gender;
                    playerDto.Country = profile.Country;
                }
            }

            float sumRatingOverTeam = 0;
            var playersOfTeam = _playerRepository.GetPlayers(teamId);
            foreach(var player in playersOfTeam)
            {
                var rating = _playerRepository.GetRating(player.Id);
                sumRatingOverTeam += rating;
            }
            var averageRatingOverTeam = sumRatingOverTeam / playersOfTeam.Count;

            playersToReturn = playersToReturn.Where(p => p.AverageRating >
averageRatingOverTeam - 1);
            playersToReturn = playersToReturn.Where(p => p.AverageRating <
averageRatingOverTeam + 1);

            return Ok(playersToReturn);
        }
    }
}
```

# ДОДАТОК В

Алгоритмізація та реалізація алгоритму аналізу інформації щодо пропозицій покращення статусу гравця/команди гравців

Опис програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5292_19Б

Аркушів 7

Київ 2019

# АНОТАЦІЯ

Даний додаток містить опис програмного додатку, який реалізує алгоритм рекомендацій щодо покращення статусу гравців. Створена програмна система має такі функції:

— надавати перегляд інформації про команди;

— надавати перегляд інформації про гравців;

— надавати можливість перегляду списку рекомендованих гравців до кожної команди.

Комунікація користувача з системою відбувається через веб-інтерфейс.

При розробці серверної частини додатку було використано мову програмування C# та платформу ASP.NET Core. Для розробки клієнтської частини було використано платформу створення веб-компонентів Angular. У якості середовища програмування було використано редактор коду Visual Studio Code.

# ЗМІСТ

# ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис програмної системи для реалізації алгоритму рекомендацій нових гравців. Додаток Б містить програмний код реалізованого алгоритму рекомендацій.

Розроблена система працює на комп’ютерах під управлінням Windows, Linux та MacOS, потребуючи попередньо встановленого додатку .NET Core Runtime.

Під час розробки системи було використано мову програмування C# та платформу Angular для побудови веб-компонентів.

# ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблена система виконує завдання щодо видачі рекомендацій командам відносно нових граців. Система призначена для використання спеціалістами у галузі аналізу кіберспортивної статистики.

Функціональні обмеження на використання даної системи, виходять тих даних, що містяться в базі і використовуються для прорахунку рекомендацій.

# ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Для реалізації алгоритму рекомендацій було потрібним створення DLL-бібліотеки, яка містить класи для роботи з алгоритмом та збереження його моделі для подальшого використання на серверній частині програми.

Серверна частина програми розроблена з використанням технології ASP.NET Core Web API, в якій реалізовано доступ до бази даних, звідки береться інформація потрібна для прорахунку рекомендацій.

Клієнтську частину було реалізовано за допомоги бібліотеки веб-компонентів Angular, та налагоджено комунікацію з серверною частиною з використанням протоколу HTTP.

# ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для розробки програмного додатку для демонстрації роботи реалізованого алгоритму рекомендацій було використано багатофункціональний редактор коду Visual Studio Code, який є зручним середовищем для написання програм як для серверної частини, так і для клієнтської.

Розроблений додаток може працювати в операційних системах сімейства Windows, Linux та MacOS завдяки крос-платформеності середовища .NET Core.

# ВИКЛИК І ЗАВАНТАЖЕННЯ

Для запуску розробленого програмного додатку необхідно мати встановлений попередньо комплекс .NET Core SDK та веб-браузер з підтримкою javascript. Сама програма інсталяції не потребує, достатньо перейти у папку з проектом та набрати у командному рядку “dotnet run”. Після запуску відкриється вікно веб-браузера і користувач зможе виконувати необхідні дії з програмою.

# ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розробленого додатку є інформація про статистику команд та гравців, яка береться з бази даних.

Вихідними даними є список рекомендованих гравців для кожної з команд.