МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

Комплексна магістерська робота

РОЗРОБКА БІБЛІОТЕКИ ДЛЯ ОПТИМІЗАЦІЇ СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ РІЗНИХ ТИПІВ

Виконали: студенти групи ПМІМ-22с
спеціальності
122 – комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Гайдук М. П.
(прізвище та ініціали)

(підпис) Кутянський О. Р.
(прізвище та ініціали)

Керівник (підпис) Дреботій Р. Г.
(прізвище та ініціали)

Рецензент (підпис) Борачок І.В.
(прізвище та ініціали)

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*
Кафедра *інформаційних систем*
Спеціальність *122 — комп'ютерні науки*
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри. *проф. Шинкаренко Г. А.*
« *5* » *вересня* **20** *22* **року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТАМ

Гайдуку Максиму Петровичу
(прізвище, ім'я, по батькові)

Кутянському Остапу Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Розробка бібліотеки для оптимізації створення рекомендаційних систем різних типів*

керівник роботи *Дреботій Роман Григорович, кандидат фізико-математичних наук, доцент.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від « *13* » *вересня* 20*22* року № 15

2. Строк подання студентом роботи *12.12.2022р.*

3. Вихідні дані до роботи *Інтернет-ресурси за тематикою роботи, датасети для перевірки роботи алгоритмів.*

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)
Розробка наступних алгоритмів:
1. Косинусна міра подібності (Гайдук)
2. K-найближчих сусідів (Гайдук)
3. Коефіцієнт Жаккара (Кутянський)
4. SVD (Кутянський)
5. Нейронної мережа з активаційною функцією softmax (Гайдук, Кутянський)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація дипломної роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *05.09.2022 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1.	*Аналіз алгоритмів*	*Вересень*	*Виконано*
2.	*Розробка алгоритмів*	*Жовтень*	*Виконано*
3.	*Програмна реалізація*	*Жовтень*	*Виконано*
4.	*Тестування програми*	*Листопад*	*Виконано*
5.	*Оформлення роботи*	*Грудень*	*Виконано*

Студенти _____ *Гайдук М. П.*
(підпис) (прізвище та ініціали)

_____ *Кутянський О. Р.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Дреботій Р. Г.*
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Сьогодні як ми бачимо майже усі бізнеси переходять в онлайн, тобто можна зробити замовлення багатьох товарів, послухати музику, подивитись фільми використовуючи лише браузер та інтернет, отож як для будь якого бізнесу, потрібно заохочувати до себе нову аудиторію щоб збільшити обсяг продажів та збільшити прибуток. Один з найкращих методів заохочення аудиторії є таргетова реклама, яка реалізується системою рекомендацій.

Найкращий метод для досягнення даної цілі є створення системи, яка на основі зібраних даних про вподобання інших користувачів або характеристик предметів зможе надати рекомендації. Основне припущення рекомендаційних систем полягає в наступному: ті, хто однаково оцінювали будь-які предмети в минулому, є також схильні давати схожі оцінки іншим предметам і в майбутньому. Даний метод дозволяє бізнесу вибудовувати до кожного клієнта індивідуальний підхід, демонструючи ті товари, на які користувач має більшу ймовірність звернути увагу. Це надає перевагу одразу обидвом сторонам, бізнес — може цілеспрямовано демонструвати користувачам продукти, які можуть їх зацікавити, а пошук потрібних предметів для користувачів стане комфортнішим та швидшим.

РОЗДІЛ 1. ТЕХНОЛОГІЇ

Наш стек технологій складається з основної мови програмування Python, тому що його продуктивність чудова, особливо з завданнями машинного навчання. Також легка інтеграція з додатками користувачів. Також будемо використовувати допоміжні бібліотеки такі як: Numpy і Pandas, які полегшать наші розрахунки.

Плюси мови Python:

1. Простота — код Python є стислим і читабельним навіть для нових розробників, що корисно для проектів машинного та глибокого навчання. Завдяки своєму простому синтаксису розробка додатків за допомогою Python є швидкою в порівнянні з багатьма мовами програмування.
2. Масова онлайн-підтримка — Python є мовою програмування з відкритим вихідним кодом і користується чудовою підтримкою з багатьох ресурсів і якісної документації по всьому світу. Він також має велику та активну спільноту розробників, які надають свою допомогу на будь-якому етапі розвитку.
3. Швидкий розвиток — у Python є легкий та дружній для розуміння синтаксис. Крім того, численні бібліотеки сприяють розробці програмного забезпечення.
4. Продуктивність — деякі розробники стверджують, що Python відносно повільний порівняно з іншими мовами програмування. Незважаючи на те, що швидкість не є однією з сильних сторін Python, вона забезпечує рішення, відоме як Cython. Це наднабір мови Python, розроблений для досягнення продуктивності коду так само, як і мова C.

РОЗДІЛ 2. МЕТОД КОЛАБОРАТИВНОЇ ФІЛЬТРАЦІЇ

Метод колаборативної фільтрації — це один з методів побудови прогнозу в рекомендаційних системах, який використовує уподобання або оцінки користувачів для створення прогнозу уподобань іншого користувача. Прогнози складаються індивідуально для кожного користувача, хоча інформація, що використовується, зібрана від багатьох учасників. Це відрізняє колаборативну фільтрацію від більш простого підходу, який дає усереднену оцінку для кожного об'єкта інтересу, наприклад того, що базується на кількості поданих за нього уподобань.

Даний метод ділиться на такі типи:

1. Заснований на пам’яті;
2. Заснований на моделі.

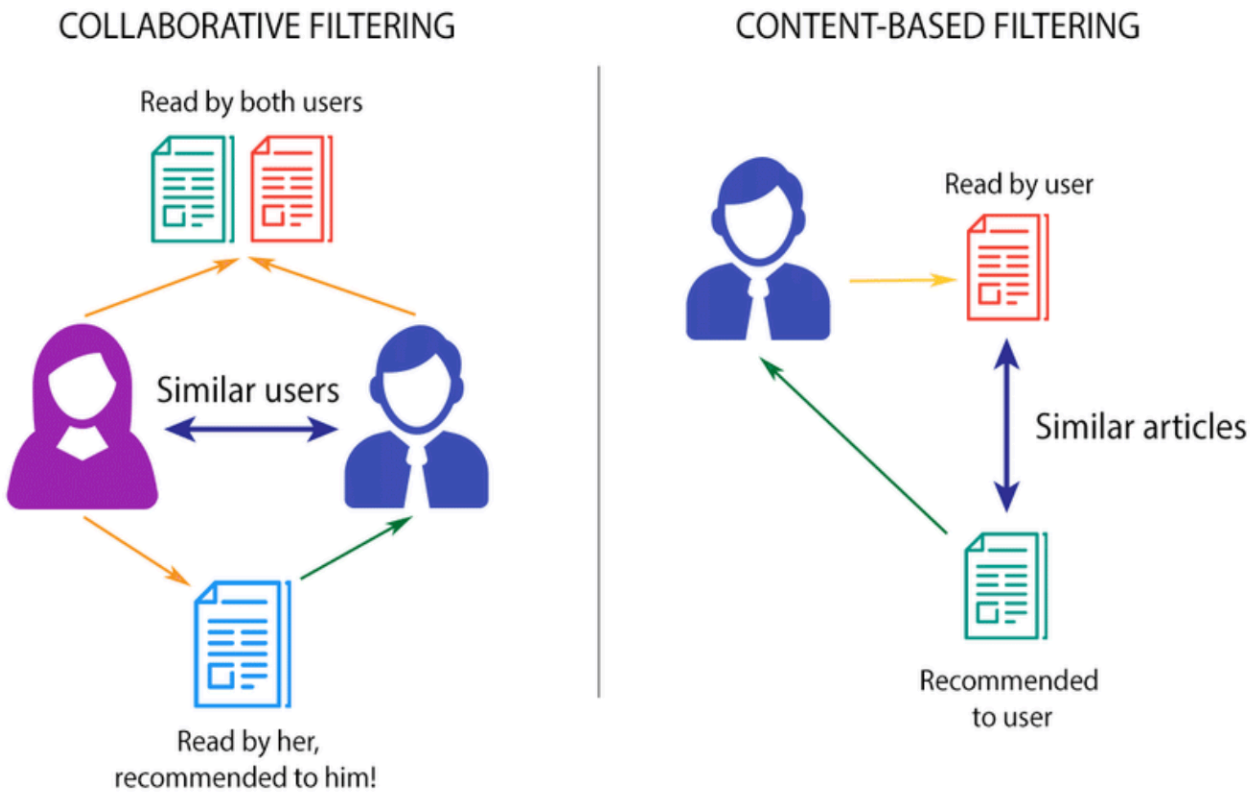

Рисунок 2.1 — Порівння колаборативної фільтрації та фільтрації на основі характериск предметів.

Даний метод може бути поділений на дві основні секції:

1. Фільтрування користувач - предмет;
2. Фільтрування предмет - предмет.

Фільтрування користувач - предмет — вибираємо користувача, потім вибираємо користувачів, які є подібними, на основі міри подібності або рейтингу і рекомендуємо предмет, які «подібні» користувачі вподобали або оцінили.

Фільтрування предмет - предмет — вибираємо усі предмети і також усіх користувачів, які їх вподобали. Далі знаходимо інші предмети, які користувачі і «подібні» користувачі вподобали і на основі цих предметів надається рекомендація.

Розглянемо міри подібності, які використовуються при розв'язанні задач класифікування, та створення прогнозів.

3.1 Коефіцієнт Жаккара

Нехай у нас є дано дві сутності: користувачі та предмети, які користувачі вподобали. З кожним користувачем **i** ми будемо асоціювати множину предметів, які користувач вподобав — u_i. Для початку потрібно визначити коефіцієнт «подібності» двох користувачів **i** та **j** за коефіцієнтом Жаккара

$$similarity(i,j) = \frac{|u_i \cap u_j|}{|u_i \cup u_j|} \qquad (3.1.1)$$

Тепер нехай в нас є деякий предмет **k,** який користувач не вподобав ще, тобто не належить до множини u_i. Тоді формулу для знаходження рекомендації можна записати так

$$rec(i,k) = g * \frac{\sum_{j \in J_k} similarity(i,j)}{n_k}, \qquad \text{де} \qquad (3.1.2)$$

n_k — це кількість користувачів, які вподобали предмет

J_k — це множина користувачів, які вподобали предмет

$$g = \frac{1}{\sum_{u_i \in U} similarity(u, u_i)} \text{ — нормуючий множник} \quad (3.1.3)$$

Приклад :

Розрахуємо міри подібності для двох множин:

u = {1, 1, 2, 3, 0, 1}

v = {0, 0, 1, 0, 2, 2}

Перетин множин u та v:

$$u \cap v = \{1, 8, 4\} \quad (3.1.4)$$

Об’єднання множин u та v:

$$u \cap v = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \quad (3.1.5)$$

Потужність перетину множин u та v дорівнює 3, а потужність об’єднання множин u та v дорівнює 9.

Згідно з формулою (3.1.1), коефіцієнт Жаккара буде:

$$similarity(i, j) = \frac{|u_i \cap u_j|}{|u_i \cup u_j|} = \frac{3}{9} = 0{,}333 \quad (3.1.6)$$

Отже дані алгоритми мають такі переваги:

1. Можливість швидкого старту, тобто не потрібно задіювати велику кількість людей, для реалізації даного методу;
2. Простота в обчисленнях;
3. Відсутність проблеми «cold start», тобто будь-які дані, які додаються в систему або базу, легко інтегруються з даним типом, не потрібно придумавати ніяких механізмів для навчання системи.

А також і недоліки:

1. Зниження продуктивності, коли дані є розрідженні;
2. При великій кількості данних, дана інтелектуальна система буде вимагати велику кількість ресурсів, щоб швидко обраховувати потрібні результати.

3.2 Косинусна міра подібності

Косинусна міра подібності – це міра кута між двома ненульовими векторами. Два однаково напрямлені вектори мають косинусну міру подібності, рівну «1», два ортогональні вектори мають міру подібності «0», а два вектори, що мають діаметрально протилежні напрями, мають міру подібності «-1». Слід зазначити, що в основному в задачах використовується косинусна міра подібності в діапазоні від «0» до «1». Одиничні вектори вважаються максимально «подібними», якщо вони паралельні, а у випадку ортогональності вектори вважаються максимально «розбіжними».

Косинусна міра подібності відноситься до обох секцій фільтрування користувач - предмет та предмет - предмет, тому завдяки ній можна діставити рекомендації по предметах, або користувачах.

Користувач - предмет

Нехай у нас дано дві сутності: користувачі та предмети, яким користувачі дали оцінку. Де A_i це оцінка предметів даного користувача, а B_i - оцінка предметів іншого користувача**.**

$$similarity = cos(\theta) = \frac{A * B}{||A|| * ||B||} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}} \qquad (3.2.1)$$

Предмет - предмет

У цьому випадку, обраховується матриця подібності використовуючи косинусну міру подібності. І вже на основі предметів користувача якому потрібно дати рекомендацію, беруться значення з матриці та обраховується середнє арифметичне між однаковими предметами подібності, та в результаті отримується список предметів та їх значення від 0 до 1.

Нехай дано n предметів, побудуємо матрицю рекомендацій розміром n x n, де $n_{ij}, i \leq n, j \leq n$ - значення косинусної міри подібності між предметами i та j.

Таблиця 3.2.2 — Матриця подібності.

	Предмет 1	**Предмет 2**	**…**	**Предмет n**
Предмет 1	1	0.8	…	0.56
Предмет 2	0.8	1	…	0.74
…	…	…	1	…
Предмет n	0.56	0.74	…	1

Перевага даного типу рекомендаційних систем є те, що найкраще цей алгоритм підходить коли маємо дані високої розмірності і коли величина векторів не має значення.

Одним з основних недоліків косинусної подібності є те, що не враховується величина векторів, а лише їх напрямок. На практиці це означає, що відмінності в значеннях не враховуються повністю. Якщо взяти, наприклад, систему рекомендацій, то подібність косинуса не враховує різницю в шкалі оцінок між різними користувачами.

Приклад:

Розрахуємо міри подібності для наступних векторів:

$a = (3,2,5)$

$b = (1,6,3)$

Обчислимо норми векторів, a та b

$$||A|| = \sqrt{\sum_{i=1}^{n} a_i^2} \approx 6{,}16$$

$$||B|| = \sqrt{\sum_{i=1}^{n} b_i^2} \approx 6{,}78$$

Обчислимо скалярний добуток векторів a та b. Це буде сума добутків координат даних векторів:

$$\sum_{i=1}^{n} a_i * b_i = 30$$

Використовуючи формулу косинусної міри подібності (3.2.1) та знайдені результати вище, знайдемо косинусну міру подібності:

$$similarity = cos(\theta) = \frac{30}{6{,}16 * 6{,}78} = 0.72$$

3.3 Метод k-найближчих сусідів

Метод k-найближчих сусідів - це алгоритм, який зберігає весь набір даних та дає змогу знаходити подібні предмети на основі загальних рейтингів і робити прогнози, використовуючи середній рейтинг найближчих сусідів з топ-k предметів.

Нехай нам дано k предметів, і у кожного предмету є n кількість характеристик для порівнянь. Візьмем 2 предмети a та b, і знайдемо коефіцієнт подібності між ними використавши формулу Евклідової відстані (3.3.1).

$$dist(a, b) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \ldots + (b_n - a_n)^2} \qquad (3.3.1)$$

Якщо значення Евклідової відстані рівне 0, то вважається, що цей предмет можна рекомендувати користувачу, так як характеристики предмету є однакові. Зі збільшенням відстані між предметами, коефіцієнт подібністі предметів зменшується.

Попробуєм розібрати на прикладі, де нам потрібно дати рекомендацію вхідному предмету (червона зірка). На даному рисунку 3.3.2 зображені елементи в двовимірній системі координат, і для того щоб знайти рекомендації вхідному предмету, потрібно знайти Евклідову відстань між усіма предметами, і взяти k варіантів які мають найменшу відстань.

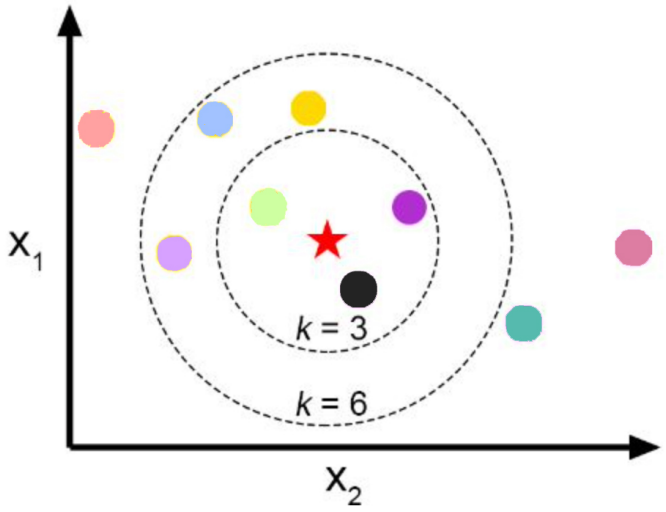

Рисунок 3.3.2 — Розміщення предметів в системі координат.

В даному випадку якщо на рис. (3.3.2) k = 3, так буде охоплювати 3 предмети рекомендації, а при k = 6 буде 6 предметів.

Переваги даного типу рекомендаційних систем:

1. Це досить простий алгоритм, який інтуітивно можна зрозуміти;
2. Однією з найбільших переваг K-NN є те, що K-NN можна використовувати як для задач класифікації, так і для задач регресії (прогноз на основі вибірки об'єктів з різними ознаками);
3. K-NN - це підхід, заснований на пам'яті. Класифікатор одразу адаптується, коли ми збираємо нові навчальні дані. Це дозволяє алгоритму швидко реагувати на зміни введених даних під час використання в реальному часі.

Недоліки:

1. K-NN може бути дуже простим у реалізації, але в міру зростання набору даних ефективність або швидкість алгоритму дуже швидко падає;
2. Алгоритм дуже чутливий до сторонніх предметів, оскільки він просто обирає сусідів на основі критеріїв відстані.

Приклад:

Нехай нам дано вибірку книг (3.3.3), де кожна книга характеризується наступними полями:

1. Кількість сторінок;
2. Середній рейтинг;
3. Рік випуску.

Таблиця 3.3.3 — Вибірка книг.

	Кількість сторінок	Середній рейтинг	Рік випуску
Книга 1	435	4.5	1999
Книга 2	123	3.8	2003
Книга 3	378	4.1	1987
Книга 4	256	4.9	2012

Маючи наступні вхідні дані (3.3.4), визначити 2 книжки, які найбільше підійдуть для читача.

Таблиця 3.3.4 — Вхідні дані книги для надання рекомендацій.

	Кількість сторінок	Середній рейтинг	Рік випуску
Книга 0	225	4	2005

Для знаходження рекомендацій, спочатку обчислимо Евклідову відстань за формулою (3.3.1) між Книгою 0 та Книгами 1 - 4.

$$1: dist(K_0, K_1) = \sqrt{(435 - 225)^2 + (4{,}5 - 4)^2 + (1998 - 2005)^2} = 210.09$$

$$2: dist(K_0, K_2) = \sqrt{(123 - 225)^2 + (3{,}8 - 4)^2 + (2003 - 2005)^2} = 102.02$$

$$3: dist(K_0, K_3) = \sqrt{(378 - 225)^2 + (4.1 - 4)^2 + (1987 - 2005)^2} = 154.06$$

$$4: dist(K_0, K_4) = \sqrt{(256 - 225)^2 + (4.9 - 4)^2 + (2012 - 2005)^2} = 31.8$$

Результат: Книга 4 та Книга 2.

Отже, якщо людина прочитала Книгу 0, то їй можна рекомендувати Книгу 2 та 4.

РОЗДІЛ 4. КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ ЗАСНОВАНА НА МОДЕЛІ

Даний тип надає рекомендації, вимірюючи параметри статистичних моделей для оцінки користувачів, які є побудовані за допомогою алгоритмів кластеризації, факторизації матриць і машинного навчання.

Розглянемо алгоритм факторизації матриць. Факторизації — це операція розкладу об’єкта на його прості компоненти. Для початку складемо таблицю, яка буде містити оцінки предметів від користувачів. Таблиця буде мати наступний вигляд

Таблиця 4.1 — Оцінки предметів користувачів.

	Користувач 1	Користувач 2	Користувач 3	Користувач 4	Користувач 5
Предмет 1	0	0	2	0	1
Предмет 2	2	0	0	5	0
Предмет 3	0	2	1	0	5
Предмет 4	5	5	0	0	3

Завдання полягає в тому, щоб апроксимувати значення в таблиці, де стоять 0, тобто надати рекомендації.

Існує кілька найпоширеніших способів як ми можемо розкласти матрицю :

1. Principal component analysis (метод головних компонент);
2. Singular-value decomposition (сингулярний розклад);
3. Non-negative matrix factorization (Розклад невід'ємних матриць).

Вибір даного способу прямо залежить від вашої моделі і вибирається емпіричним способом, хоча варто пам’ятати, що спосіб NNMF приймає дані лише із значеннями від 0 до + нескінченості, як на вхід так і на вихід, натомість метод SVD приймає на вхід і вихід значення будь-якого діапазону.

Опишемо основну ідею. Нехай наша початкова матриця V(n * m) розкладається на дві інші матриці, W(n * k) та H(k * m) , тобто у вигляді формули можна записати так

$$V(n * m) = W(n * k) * H(k * m) \text{ , де} \qquad (4.2)$$

k - кількість компонент. Компоненти — це може бути якась категорія в магазині(овочі, фрукти, випічка, продовольчі товари), жанри музики і фільмів і тд.

На даному малюнку можна побачити ідею факторизації матриці:

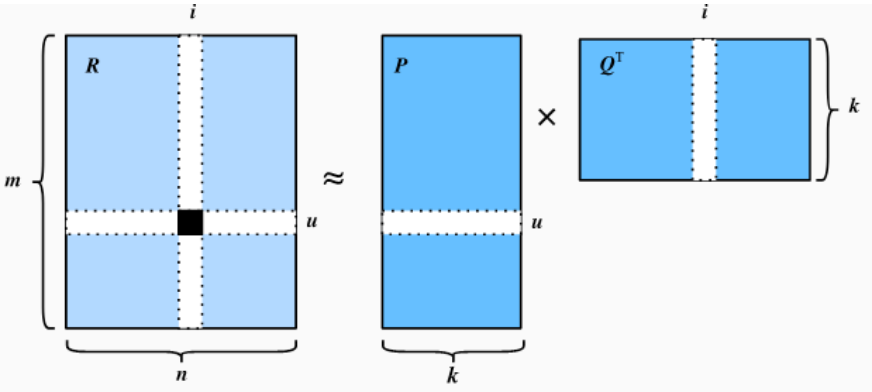

Рисунок 4.3 — Ідея факторизації матриці.

Перевагою даного типу рекомендаційних систем є те, що алгоритм набагато ефективніше обробляє розріджені матриці, так як можна написати ефективний алгоритм множення матриць.

Недоліками:

1. Необхідно знайти баланс між точністю і розміром моделі, адже можна втратити корисну інформацію у зв’язку із скороченням моделі;
2. Не може підтримувати нові дані, які не були надані під час тренувань, потрібно реалізовувати додаткові механізми, щоб уникнути цієї проблеми.

4.1 Опис алгоритму SVD

Нехай задано:

1. k керуючих параметрів;
2. m — загальна кількість користувачів;
3. n — загальна кількість предметів.

Тоді ми можемо розділити матрицю $R_{(m*n)}$ на матрицю $P_{(m*k)}$ та $Q_{(n*k)}$, так що при множенні P на Q в нас буде відтворюватися матриця R:

$$R_{m*n} = P_{(m*k)} * \Sigma_{(k*k)} * Q^T_{(n*k)} \quad , \text{ де} \qquad (4.1.1)$$

1. R — матриця, в котрій індекси по рядках це унікальні ідентифікатори користувачів, індекси по стовпцях — унікальні ідентифікатори педметів, а R[i,j] — оцінка, що користувач поставив предмету.
2. P — матриця, яка показує зв'язок між користувачами та керуючими параметрами.
3. Q — матриця, що показує зв'язок між предметами та керуючими параметрами.
4. ∑ — діагональна матриця з вагами для керуючих параметрів.

Тепер для того, щоб отримати прогнозований рейтинг для предмету використовуємо дану формулу:

$$R_{ui} = \sum_{k=1}^{k} P_{uk} * \Sigma_k * Q_{ki} \tag{4.1.2}$$

Тепер нам потрібно заповнити матрицю P та Q таким чином, щоб апроксимація до матриці R, відбувалась з мінімальною помилкою. Для цього будемо використовувати алгоритм градієнтного спуску.

$$e_{ui}^2 = (r_{ui} - r_{ui}^{\bullet})^2 = (r_{ui} - \sum_{k=1}^{k} p_{uk} * \sigma_k * q_{ki})^2 \quad \text{, де} \tag{4.1.3}$$

1. e_{ui}^2 — помилка;
2. r_{ui} — правильна оцінка, який користувач поставив предмету;
3. $r_{ui}^{\bullet}$ — оцінка, котру спрогнозувала система.

Основна ціль — підібрати такі значення в матрицях p та q, щоб помилка була мінімальною. Тепер нам потрібно задати правило, як оновлювати дані для p_{uk} та q_{ki}. Спочатку запишемо градієнти

$$\frac{\delta}{\delta p_{uk}}(e_{ui}^2) = -2(r_{ui} - r_{ui}^{\bullet}) * q_{ki} = -2 * e_{ui} * q_{ki} \tag{4.1.4}$$

$$\frac{\delta}{\delta q_{ki}}(e_{ui}^2) = -2(r_{ui} - r_{ui}^{\bullet}) * p_{uk} = -2 * e_{ui} * p_{uk} \tag{4.1.5}$$

Тепер маючи градієнти можемо записати і правило оновлювання даних з матриць p та q.

$$p'_{uk} = p_{uk} - \alpha * \frac{\delta}{\delta p_{uk}} * e_{ui}^2 = p_{uk} + 2 * e_{ui} * q_{ki} \quad (4.1.6)$$

$$q'_{ki} = q_{ki} - \alpha * \frac{\delta}{\delta q_{ki}} * e_{ui}^2 = q_{ki} + 2 * e_{ui} * p_{uk} \text{ , де} \quad (4.1.7)$$

α — коефіцієнт швидкості навчання.

4.2 Опис нейронної мережі з активаційною функцією softmax

4.2.1 Будова нейронної мережі

Нейронні мережі — один із напрямків досліження в галузі штучного інтелекту, яка має здібність до самонавчання та може поступово підвищувати свою продуктивність.

Найменша обчислювальна одиниця, з чого складається мережа — нейрон. Нейрон — це математична функція, яка приймає на вхід дані, перетворює їх, та повертає нові вихідні дані.

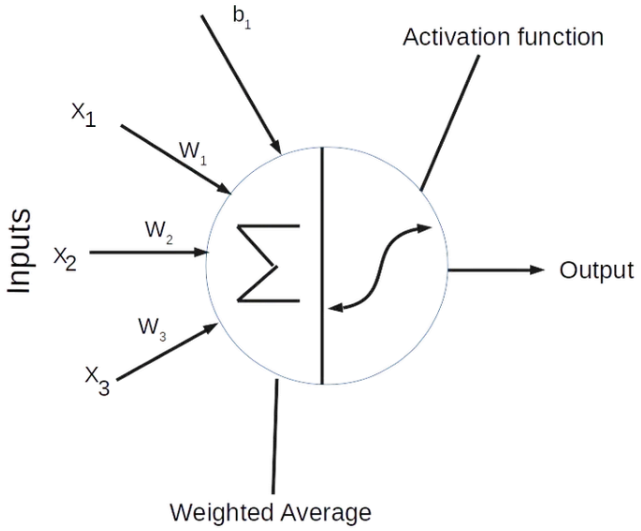

Рисунок 4.2.1.1 — Будова нейрону.

На рис. 4.2.1.1 представлено будову математичної моделі нейрона. Отже він складається із двох частин: суматор та активаційної функції. Нейрон приймає дані x_1, x_2, x_3, і перемножує із вагами w_1, w_2, w_3. Після цього, потрібно передати результат в активаційну функцію, щоб отримати вихідні дані. Активаційна функція — це неперервно диференційована функція, яка показує залежність вихідних даних нейроної мережі від вхідних. Переважно обирають нелінійні функції, для того, щоб результат також не мав лінійної залежності і для кращого процесу навчання мережі.

Нейронна мережа об’єднює певну кількість нейронів в групи, яких також може буде декілька. Дані групи називають шарами нейронної мережі. Найпростіша нейрона мережа називається перцептроном і є зображенна на рисунку 4.2.1.2.

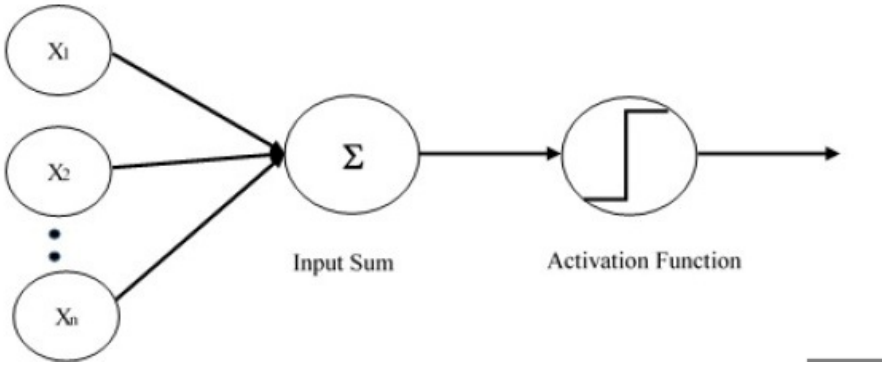

Рисунок 4.2.1.2 — Будова нейрону.

Також мережа може мати і складнішу структуру, наприклад як показано на рис. 4.2.1.3.

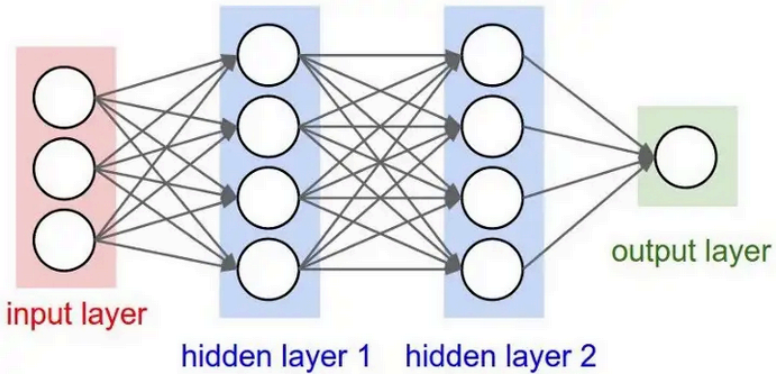

Рисунок 4.2.1.3 — Будова нейрону.

На рис. 4.2.1.3, можемо бачити що мережа, складається з вхідного шару, прихованих шарів та вихідного шару із результатами. Розглянемо кожний шар детальніше:

1. Вхідний шар — це є вхідні дані, кількість вузлів в даному шарі дорівнює кількості характеристик даних;
2. Прихований (обчислюванний) шар — використовується для обчислення ваг. Нейронні мережі можуть мати декілька прихованих шарів, де кожний шар на вхід приймає вихідні дані з попереднього шару;
3. Вихідний шар — показує результат «передбачення» нейронної мережі. Якщо нашим завдання є бінарна класифікація, то вихідний шар може складатися з одного елементу, якщо завданням є багатокласова класифікація — кількість елементів дорівнює кількості класів.

Також згідно з рис. 4.2.1.3. всі шари нейронної мережі з'єднанні між собою зв'язками, відповідно кожний зв'язок повинен мати значення ваги та зміщення. Дані значення — це параметри, які використовуються для навчання мережі. Ваги представляють собою характеристики вхідних даних, наприклад: вік, зріст, вага людини, хімічний склад продукту, тощо, зміщення — це додаткова умова, яка застосовується при підрахунку вихідного значення, наприклад: вихід має змінюватися якщо x більше 1 тощо.

4.2.2 Функція оцінки

Функція оцінки (4.2.2.1) допомагає визначити як точно мережа видає результати, а також підкоригувати ваги і зміщення. Дана функція має показувати на скільки прогноз близький чи далекий від істиних значень і відповідно від результату «винагородити» або «покарати» модель. Вибір функції втрат залежить від завдання, для проблем класифікації можна використовувати функцію cross-entropy.

$$L = -\sum_{i}^{C} y_i log(\hat{y}_i) \text{ , де} \qquad (4.2.2.1)$$

1. C — кількість класів;

2. y_i — правильне значення;
3. $\hat{y}_i$ — результат нейронної мережі.

4.2.3 Функція активації softmax

Основна ідея функції активації softmax (4.2.3.1) полягає в тому, щоб розподілити ймовірність різних класів так, щоб їхня сума дорівнювала 1.

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}}, \text{ де} \qquad (4.2.3.1)$$

N - загальна кількість класів;

z_i - результат попереднього шару нейронної мережі;

a_i - ймовірність відношення до певного класу.

Варто зазначити, що передостанній шар нейронної мережі повинен містити на виході таку саму кількість нейронів як кількість класів.

Приклад:

Нехай нам дано 3 класи { клас_1, клас_2, клас_3 }, відповідно їхні індекси будуть { 0, 1, 2 }. Припустимо нам дано вектор предметів, які є вихідними даними нейронної мережі z = [0.25, 1.23, -0.8].

Тепер застосуємо функцію softmax по вектору z.

$$a_0 = \frac{e^{0.25}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.249$$

$$a_1 = \frac{e^{1.23}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.664$$

$$a_2 = \frac{e^{-0.8}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.087$$

З результату можемо сказати наступне:

1. У векторі z максимальне значення становить 1,23, яке після застосування активації softmax відповідає 0,664 що є найбільшим числом у векторі виходу softmax. Подібним чином 0,25 і -0,8 становитимуть 0,249 і 0,087;
2. Записи у вихідному векторі softmax мають значення від 0 до 1;
3. У задачі багатокласової класифікації, де класи є взаємовиключними, зверніть увагу на те, що сума результату softmax рівна 1: $0.249 + 0.664 + 0.087 = 1$.

Результат: у векторі [0.249, 0.664, 0.087], значення 0.664 є найбільшим, яке означає що з ймовірністю 66.4% предмет відноситься до класу **клас_2**, і відповідно 24.9% та 8.7% до класів **клас_1** і **клас_3**.

4.2.4 Похідна функції активації та оцінки

В даному розділі знайдемо похідну функції активації softmax.

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}} \text{ (напевно просто дати посилання номер)} \qquad (4.2.4.1)$$

Для знаходження похідної використаємо дану формулу

$$f(x) = \frac{g'(x)}{h'(x)} = \frac{g'(x)h(x) \; - \; h'(x)g(x)}{h(x)^2}, \text{ де} \qquad (4.2.4.2)$$

- $g(x) = e^{z_i}$
- $h(x) = \sum_{k}^{c} e^{z_k}$

Тепер,

$$\frac{da_i}{dz_j} = \frac{d}{dz_j}\left(\frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}}\right) = \frac{d}{dz_j}\left(\frac{g(x)}{h(x)}\right) \qquad (4.2.4.3)$$

Обчислимо $g'(x)$:

$$\frac{d}{dz_j}(g(x)) = \frac{d}{dz_j}(e^{z_i}) = \frac{d}{dz_j}(e^{z_i})\ \frac{dz_j}{dz_j}(z_j) = e^{z_i}\frac{dz_j}{dz_j}(z_j) = \begin{cases} e^{z_i}\ if\ i = j \\ 0\ \ if\ i \neq j \end{cases} \quad (4.2.4.4)$$

Обчислимо $h'(x)$:

$$\frac{d}{dz_j}(h(x)) = \frac{d}{dz_j}\left(\sum_k^c e^{z_k}\right) = \frac{d}{dz_j}\left(\sum_{k=1,k\neq j}^c e^{z_k} + e^{z_j}\right) =$$

$$= \frac{d}{dz_j}\left(\sum_{k=1,k\neq j}^c e^{z_k}\right) + \frac{d}{dz_j}(e^{z_j}) = 0 + e^{z_j} = e^{z_j} \quad (4.2.4.5)$$

Отже ми отримали два варіанта:

1. При i = j, тоді

$$\frac{da_i}{dz_j} = \frac{e^{z_i}\sum_{k=1}^c e^{z_k} - e^{z_j}e^{z_i}}{(\sum_{k=1}^c e^{z_k})^2} = \frac{e^{z_i}(\sum_{k=1}^c e^{z_k} - e^{z_j})}{(\sum_{k=1}^c e^{z_k})^2} =$$

$$= \frac{e^{z_i}}{\sum_{k=1}^c e^{z_k}} * \frac{\sum_{k=1}^c e^{z_k} - e^{z_j}}{\sum_{k=1}^c e^{z_k}} = a_i(1 - a_j) = a_i(1 - a_i) \text{ так як i = j;} \quad (4.2.4.6)$$

2. При i ≠ j, тоді

$$\frac{da_i}{dz_j} = \frac{0\sum_{k=1}^c e^{z_k} - e^{z_j}e^{z_i}}{(\sum_{k=1}^c e^{z_k})^2} = \frac{-e^{z_j}e^{z_i}}{(\sum_{k=1}^c e^{z_k})^2} = -a_i a_j \quad (4.2.4.7)$$

Знайдемо похідну функції оцінки cross entropy, яка буде використовуватися для навчання мережі, на етапі зворотнього поширення помилки:

$$\delta E(\theta)) = \frac{d}{dz_i}\left[-\sum_{k=1}^c y_k log(a_k)\right] = -\sum_{k=1}^c y_k \frac{d(log(a_k))}{dz_i} = -\sum_{k=1}^c y_k \frac{d(log(a_k))}{da_k} * \frac{da_k}{dz_i} =$$

$$= -\sum_{k=1}^c \frac{y_k}{a_k} * \frac{da_k}{dz_i} = -\left[\frac{y_i}{a_i} * \frac{da_i}{dz_i} + \sum_{k=1,k\neq i}^c \frac{y_k}{a_k} * \frac{da_k}{dz_i}\right] = -y_i + y_i a_i + \sum_{k=1,k\neq i}^c y_k a_i =$$

$$= a_i(y_i + \sum_{k=1,k\neq i}^{c} y_k) - y_i = a_i + \sum_{k=1}^{c} y_k - y_i = a_i - y_i \quad (4.2.4.8)$$

4.2.5 Навчання нейронної мережі

Найважливіший критерій нейронної мережі — це можливість її навчання. Як вже було зазначено, нейронна мережа — це сукупність нейронів, які можуть бути об'єднанні в шари, через які проходять дані, відповідно дані можуть проходити повз багато шарів і на виході ми отримуємо результат, проте ми не впевненні чи правильний він. Відповідно, для процесу навчання використовуються коефіцієнти, параметри та корелюючі функції, які застосовуються до ваг та зміщеннь.

Отже навчання нейронної мережі — це пошук коефіцієнтів ваг та зміщень, які після проходження через суматор та активаційну функцію дозволять отримати вихідний результати, який є максимально наближений до істиного.

Навчання мережі можна поділити на навчання з учителем, та навчання без вчителя:

1. Навчання з вчителем — це концепція, коли нейронній мережі надають вибірку вхідних даних, як результат отримуємо вихідні дані та порівнюємо ці дані з істиними, після чого можна обрахувати точність та помилку навчання мережі.
2. Навчання без вчителя — це концепція, коли нейронній мережі надають вибірку вхідних данних, проте не надають правильних відповідей. Нейромережа починає кластеризувати дані, тобто визначає класи відносно вхідних даних, та видає результати різних типів, що відповідають за вхідни дані.

Навчання нейронної мережі відбувається у два етапа:

1. Пряме поширення помилки.
2. Зворотнє поширення помилки.

Детальніше про дані етапи буде описано у відповідних підпунктах.

4.2.6 Етап прямого поширення помилки

Під час етапу прямого поширення помилки робиться прогноз відповіді.

Рисунок 4.2.6.1 — Етап прямого поширення помилки.

Згідно рис. 4.2.6.1 показано основну ідею даного етапу, а саме вихідний результат поперенього шару, є вхідними даними наступного шару. В кінці отримано результат $\hat{Y}$, за допомогою якого можемо порахувати помилку E. Основне завдання полягає в тому, щоб підібрати ваги параметри таким чином, щоб мінімізувати помилку.

Значення кожного нейрона можна обрахувати наступною формулою:

$$y_i = \sum_i x_i w_{ij} + b_j \text{ , де} \qquad (4.2.6.2)$$

1. x_i — вхідні дані;
2. $w_{i,j}$ — ваги;
3. b_j — зміщення.

Для оптимізації компютерних обчисленнь дану формулу можна записати таким способом:

Y = XW + B , де

1. X = $[x_1 \dots x_i]$;
2. W = $\begin{bmatrix} w_{11} & \dots & w_{1j} \\ \vdots & \ddots & \vdots \\ w_{i1} & \dots & w_{ij} \end{bmatrix}$;
3. B = $[b_1 \dots b_j]$.

4.2.7 Етап зворотнього поширення помилки

Даний етап відповідає за мінімізацію помилки, яку ми отримали по закінченню попереднього етапу, під час даного етапу відбуваються зміни у значеннях ваг та зміщення. Для обрахунків нам потрібно знайти відповідні похідні:

1. $\frac{\partial E}{\partial W}$;
2. $\frac{\partial E}{\partial B}$;
3. $\frac{\partial E}{\partial X}$.

Нижче буде обраховано кожну із них.

Обрахуємо формулу для коригування ваг. Нехай в нас дана матриця розміром i * j, де i — індекс нейрона із вхідними даними, j — індекс нейрона із вихідними даними.

$$\begin{bmatrix} \frac{\partial E}{\partial w_{11}} & \cdots & \frac{\partial E}{\partial w_{1j}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{w_{i1}} & \cdots & \frac{\partial E}{\partial w_{ij}} \end{bmatrix} \tag{4.2.7.1}$$

Використовуючи ланцюгове правило, при диференціюванні складної функції, ми можемо записати:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{ij}} + \ldots + \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial E}{\partial y_i}x_i, \tag{4.2.7.2}$$

отже, можемо записати формулу $\frac{\partial E}{\partial W}$:

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial w_{11}}x_1 & \cdots & \frac{\partial E}{\partial w_{1j}}x_1 \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{w_{i1}}x_i & \cdots & \frac{\partial E}{\partial w_{ij}}x_i \end{bmatrix} =$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_i \end{bmatrix} [\frac{\partial E}{\partial y_1} \ \dots \ \frac{\partial E}{\partial y_j}] = X^T \frac{\partial E}{\partial Y} \tag{4.2.7.3}$$

Обахуємо формулу для коргування зміщень. Нехай дано масив зміщень:

$$[\frac{\partial E}{\partial b_1}, \frac{\partial E}{\partial b_2} \ \dots \ \frac{\partial E}{\partial b_j}] \tag{4.2.7.4}$$

Використовуючи ланцюгове правило, при диференціюванні складної функції, ми можемо записати:

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial b_j} + \dots + \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial b_j} = \frac{\partial E}{\partial y_j}, \tag{4.2.7.5}$$

отже, можна записати формулу $\frac{\partial E}{\partial B}$:

$$\frac{\partial E}{\partial B} = [\frac{\partial E}{\partial b_1}, \frac{\partial E}{\partial b_2} \ \dots \ \frac{\partial E}{\partial b_j}] = \frac{\partial E}{\partial Y} \tag{4.2.7.6}$$

Тепер обрахуємо $\frac{\partial E}{\partial X}$:

$$\frac{\partial E}{\partial X} = [\frac{\partial E}{\partial x_1}, \frac{\partial E}{\partial x_2} \ \dots \ \frac{\partial E}{\partial x_i}] \tag{4.2.7.7}$$

Використовуючи ланцюгове правило, при диференціюванні складної функції, ми можемо записати:

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_i} + \dots + \frac{\partial E}{\partial y_i}\frac{\partial y_j}{\partial x_i} = \frac{\partial E}{\partial y_1}w_{i1} + \dots + \frac{\partial E}{\partial y_i}w_{ij}, \tag{4.2.7.8}$$

отже, можна записати формулу $\frac{\partial E}{\partial X}$

$$\frac{\partial E}{\partial X} = \left[(\frac{\partial E}{\partial y_1}w_{11} + \dots + \frac{\partial E}{\partial y_j}w_{1j}) \dots (\frac{\partial E}{\partial y_1}w_{i1} + \dots + \frac{\partial E}{\partial y_j}w_{ij}) \right] =$$

$$[\frac{\partial E}{\partial y_1} \ \dots \ \frac{\partial E}{\partial y_j}] \begin{bmatrix} w_{11} & \dots & w_{1j} \\ \vdots & \ddots & \vdots \\ w_{i1} & \dots & w_{ij} \end{bmatrix} = \frac{\partial E}{\partial Y} W^T \tag{4.2.7.9}$$

В результаті, можемо записати формулу коригування ваг та зміщень:

$$\Delta w_t = -\epsilon \frac{\delta E(\theta)}{\delta w} \quad (4.2.7.10)$$

$$\Delta b_t = -\epsilon \frac{\delta E(\theta)}{\delta b}, \text{ де} \quad (4.2.7.11)$$

1. ϵ — параметр швидкості навчання;
2. $\delta E(\theta)$ — похідна функції оцінки;
3. t — номер ітерації.

РОЗДІЛ 5. РЕЗУЛЬТАТИ

5.1 Косинусна міра подібності

Були взяті наступні дані для перевірки роботи алгоритму, це є датасет який містить понад 300 продуктів, кожен із значеннями:

1. Кількості калорій;
2. Жирів;
3. Білків;
4. Насичених жирів;
5. Вуглеводів;
6. Клітковини.

На рис. 5.1.1 вигляд перших 8 елементів датасету:

	id	Food	Measure	Grams	Calories	Protein	Fat	Sat.Fat	Fiber	Carbs	Category
1	1	Cows' milk	1 qt.	976	660	32	40	36	0	48	Dairy products
2	2	Milk skim	1 qt.	984	360	36	0.01	0.01	0	52	Dairy products
3	3	Buttermilk	1 cup	246	127	9	5	4	0	13	Dairy products
4	4	Evaporated, undiluted	1 cup	252	345	16	20	18	0	24	Dairy products
5	5	Fortified milk	6 cups	1.419	1.373	89	42	23	1.4	119	Dairy products
6	6	Powdered milk	1 cup	103	515	27	28	24	0	39	Dairy products
7	7	skim, instant	1 1/3 cups	85	290	30	0.01	0.01	0	42	Dairy products
8	8	skim, non-instant	2/3 cup	85	290	30	0.01	0.01	1	42	Dairy products

Рисунок 5.1.1 — Вигляд перших 8 елементів датасету.

Було взято 100 тестових елементів, і збудована матриця подібності (рис. 5.1.2), на рисунку будуть лише 10x10 елементів, так як сама матриця є велика 100x100 елементів.

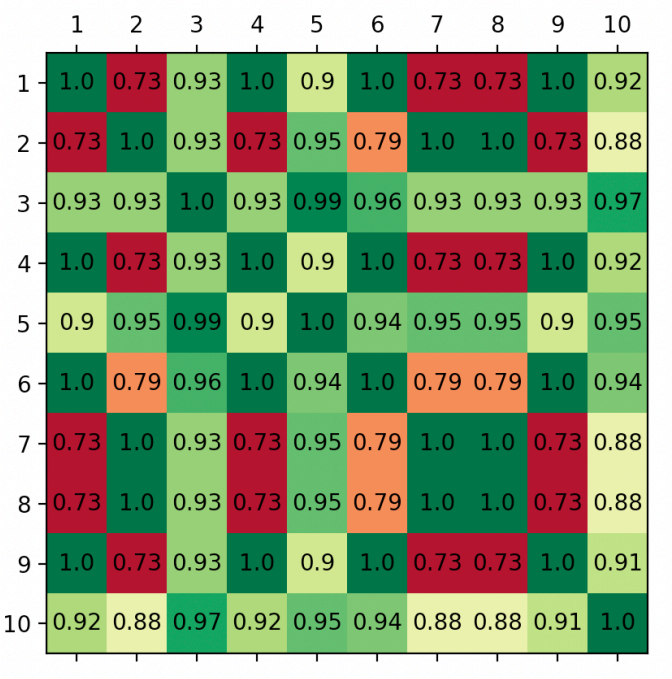

Рисунок 5.1.2 — Матриця подібності
перших 10 елементів

Було взято вибірку предметів [15, 22, 49], для отримання рекомендацій. Був отриманий наступний результат (рис. 5.1.3), це є кортеж з двох значень, де перше значення це унікальний ідентифікатор, а друга ймовірність того що цей предмет сподобається користувачу:

```
Cosine Similarity initialized (Item based)
(44, 0.897)
(17, 0.89)
(67, 0.89)
(28, 0.883)
(25, 0.88)
(29, 0.88)
(21, 0.877)
(24, 0.877)
(61, 0.877)
(65, 0.877)

Process finished with exit code 0
```

Рисунок 5.1.3 — Результати рекомендації.

5.2 Алгоритм k-найближчих сусідів

Були взяті наступні дані для перевірки роботи алгоритму, це є датасет який містить понад 178 різних вин, кожен із зазначенням хімічного аналізу вина.

Ось вигляд перших 8 елементів датасету:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.ph...	Proanth	Color.int	Hue	OD	Proline
1	1	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
2	2	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050
3	3	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185
4	4	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480
5	5	13.24	2.59	2.87	21.0	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735
6	6	14.2	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
7	7	14.39	1.87	2.45	14.6	96	2.5	2.52	0.3	1.98	5.25	1.02	3.58	1290
8	8	14.06	2.15	2.61	17.6	121	2.6	2.51	0.31	1.25	5.05	1.06	3.58	1295

Рисунок 5.2.1 — Вигляд перших 8 елементів датасету.

Для рекомендацій було вибрано вибірку для тренування з 100 елементів, і 5 елементів з тестової вибірки, і ось які елементи нам рекомендує система на основі даних 5 елементів:

```
[ 82  61  44  66  88  85  60  84  76  86  62  92  64  72 100  98  89  99]

Process finished with exit code 0
```

Рисунок 5.2.2 — Результат рекомендації.

Результатом є масив унікальних ідентифікаторів вин, які підійдуть користувачу на основі 5 даних елементів.

5.3 Коефіцієнт Жаккара

Були взяти наступні дані, які можна є на рис. 5.3.1 для перевірки роботи алгоритму, де номер стовпця — ідентифікатор користувача, а рядки — вподобані користувачем предмети. Нехай в нас є 20 користувачів та 100 предметів.

Знайдемо перші 10 предметів, які мають найбільший нормований коефіцієнт подібності Жаккара, для користувача №5.

	C1	C2	C3	C4	C5
1	1	2	3	4	5
2	79	58	98	43	75
3	39	44	4	51	30
4	84	91	66	67	74
5	81	16	49	80	57

Рисунок 5.3.1 — Вигляд перших 5 елементтів датасету.

Результатом роботи програми є список кортежів, який є на рис. 5.3.2, де перший елемент нормований коефіцієнт подібності, а другий — унікальний ідентифікатор продукту. Нормований коефіцієнт подібності може набувати значення від 0, що означає, що користувачу не треба рекомендувати цей предмет, до 1, що означає, що даний предмет найкраще підійде для користувача. Отже, можемо зробити висновок, що предмет №6 найкраще підходить для користувача 5.

```
(0.192, 6)
(0.1667, 55)
(0.1525, 36)
(0.1472, 99)
(0.1472, 29)
(0.1336, 77)
(0.1322, 47)
(0.1272, 31)
(0.1272, 20)
(0.1218, 78)
```

Рисунок 5.3.2 — Результат роботи програми.

5.4 Алгоритм SVD

Усі дані про користувачів та їхні оцінки є у файлі data.csv, який зображений на рис. 5.4.1, де перша колонка — унікальний ідентифікатор користувача, друга — унікальний ідентифікатор предмета, третя — оцінка користувача даному предмету. Якщо «rating» рівний нулю, то це означає, що користувач ще не оцінив даний предмет. Використовуючи факторизацію матриць, ми знайдемо керуючі параметри, які допоможуть зробити прогноз, як користувач міг би оцінити даний предмет.

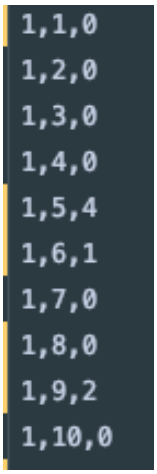

Рисунок 5.4.1 —
Структура данних файла
data.csv.

5.4.1 Налаштування гіперпараметрів

Для того, щоб алгоритм давав точніші прогнози, як і для тестової вибірки даних, так і для нових даних, потрібно вибрати правильні параметри для таких значень:

1. K — кількість керуючих чинників;
2. α — коефіцієнт швидкості навчання;
3. Epochs — кількість ітерацій виконання алгоритму градієнтного спуску.

Спочатку виберемо K. На рис. 5.4.1.1 ми можемо побачити, що при K = 15, наша рекомендаційна система показує кращий результат. Хоча при K = 5 або K =

10 помилка є меншою, але тоді наша система робить точний результат по даних на яких вона тренувалася, проте буде помилятися на нових.

Тепер виберемо α. Тут також важливо, щоб система робила точний прогноз по тестових і нових даних. На рис. 5.4.1.2 ми можемо побачити, що при $\alpha < 0.01$ та $\alpha > 0.001$, наша рекомендаційна система покаже найкращий результат. При $\alpha > 0.01$, система навчається занадто швидко, що може спричинити неправильні прогнози, на нових даних. При $\alpha < 0.0001$ система буде навчатись дуже повільно, тому нам це також не підходить.

Останій параметр, який потрібно вибрати — Epochs. Згідно з даними на рисунку 5.4.4 можна зробити висновок, що при кількості ітерацій в epochs >= 250 та epochs <= 275, система показала найкращий

Отже згідно з проведеними вище дослідженнями, для найкращої роботи системи рекомендацій були обрані наступні гіперпараметри:

1. K = 15;
2. α = [0.01, 0.001];
3. Epochs = [250, 275].

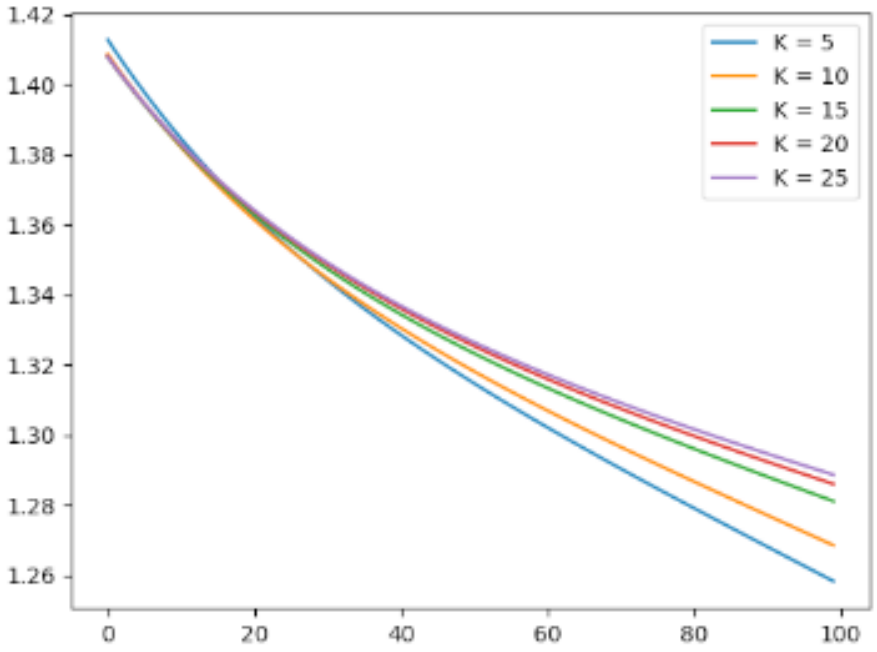

Рисунок 5.4.1.1. — Графік зменшення помилки, при зміні параметра K.

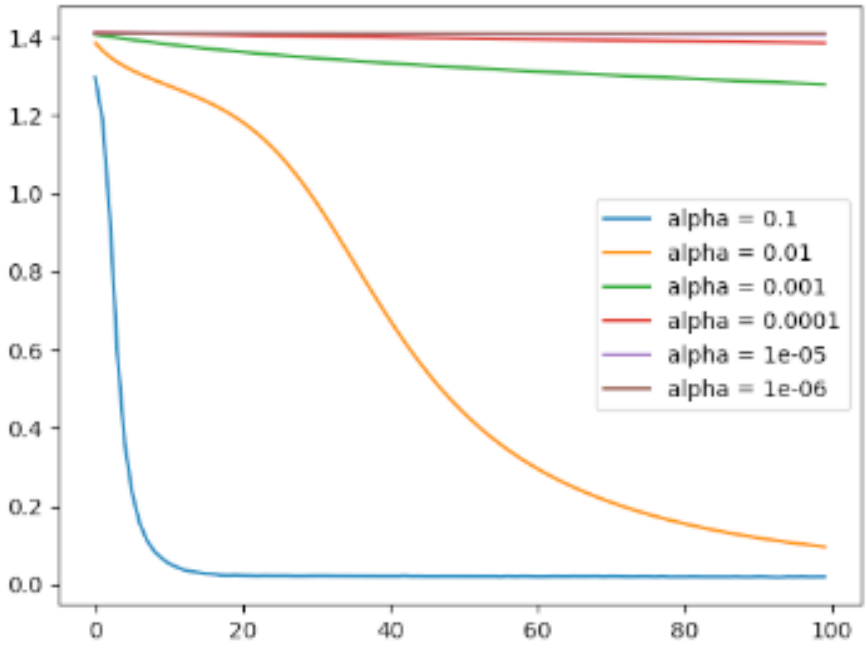

Рисунок 5.4.1.2 — Графік зменшення помилки, при зміні параметра α.

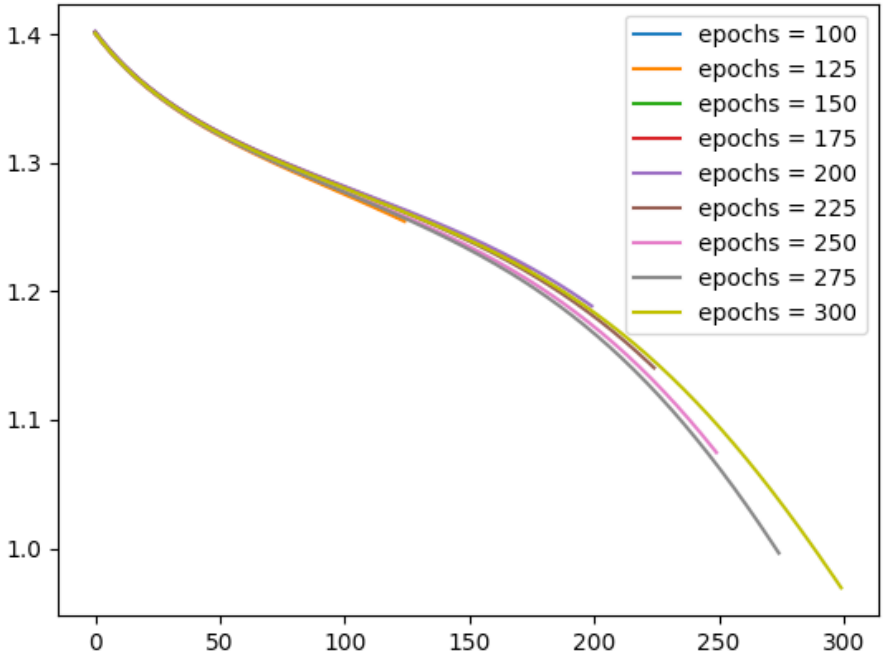

Рисунок 5.4.1.3 — Графік зменшення помилки, при зміні параметра epochs.

5.4.2 Демонстрація роботи

Отже, використовуючи знайдені гіперпараметри із пункту 5.4.1, побудуємо матрицю рекомендацій, для нашого набору данних.

	C1	C2	C3	C4	C5
1	0.0	0.0	0.0	0.0	4.0
2	0.0	0.0	0.0	0.0	3.0
3	0.0	4.0	0.0	4.0	0.0
4	0.0	5.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0

Рисунок 5.4.2.1 — Перші 5 рядків та 5 стовпців початкової матриці.

Далі запустимо функцію, яка застосує алгоритм SVD на початковій матриці.

```
Iteration: 20 ; error = 1.3070
Iteration: 40 ; error = 1.1868
Iteration: 60 ; error = 0.8763
Iteration: 80 ; error = 0.5548
Iteration: 100 ; error = 0.3716
Iteration: 120 ; error = 0.2667
Iteration: 140 ; error = 0.2004
Iteration: 160 ; error = 0.1564
Iteration: 180 ; error = 0.1260
Iteration: 200 ; error = 0.1044
Iteration: 220 ; error = 0.0886
Iteration: 240 ; error = 0.0770
Iteration: 260 ; error = 0.0683
```

Рисунок 5.4.2.2 — Інформація про кількість ітерацій та помилку.

Згідно з рисуноком 5.4.2.2 можна побачити, що помилка в останній ітерації рівна 0.0683, що є дуже хорошим результатом. Тепер подивимось як апроксимувалиця значення у новій матриці.

	C1	C2	C3	C4	C5
1	2.1222761713238074	3.7925365499445203	3.4614714010355545	3.3877519833364174	3.968729884739667
2	2.9565705503163873	4.08898281599081	3.0983928552615883	2.9507959366864536	2.9604664137694354
3	5.277683677009224	4.063208167456981	2.789133716281873	3.9394447864418476	3.274797639206527
4	2.828486302933918	4.9519703630283844	3.5264119504362443	2.197123428318558	4.031156238416607
5	4.895390221323886	5.212715223326575	3.3551267186677	3.8021210639087535	4.175199159542351

Рисунок 5.4.2.3 — Апроксимована матриця.

Як можна бачити із результату, що є на рис. 5.4.2.3, нулі замінилися на «прогнозовані» оцінки, а оцінки користувачів відрізняються не більше ніж значення помилки — 0.0683. Отже можна зробити висновок, що факторизація матриць відбулась успішно.

5.5 Нейронної мережа з активаційною функцією softmax

Були взяті наступні дані для перевірки роботи алгоритму, це є вибірка даних, яка містить понад 200 пісень з сервісу Spotify, кожна із значеннями:

1. danceability - описує на скільки пісня підходить для танцювання;
2. energy - енергія пісні;
3. key - клас висоти, набір усіх тонів віддалених на ціле число октав;
4. loudness - гучність;
5. mode - типу звукоряду, з якої приходить мелодійний зміст;
6. speechiness - присутність слів у пісні;
7. acousticness - акустичність;
8. instrumentalness - інстурментальність;
9. liveness - присутність бек вокалу;
10. valence - позитивність пісні;
11. tempo - темп, кількість ударів на хвилину;
12. duration_ms - тривалість пісні в мілісекундах;
13. time_signature - приблизно загальний розмір пісні.

На рисунку 5.5.1 вигляд перших 10 елементів датасету:

	id	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
1	0	0.803	0.624	7	-6.764	0	0.0477	0.451	0.000734	0.1	0.628	95.968	304524	4
2	1	0.762	0.703	10	-7.951	0	0.306	0.206	0.0	0.0912	0.519	151.329	247178	4
3	2	0.261	0.0149	1	-27.528	1	0.0419	0.992	0.897	0.102	0.0382	75.296	286987	4
4	3	0.722	0.736	3	-6.994	0	0.0585	0.431	1.18e-06	0.123	0.582	89.86	208920	4
5	4	0.787	0.572	1	-7.516	1	0.222	0.145	0.0	0.0753	0.647	155.117	179413	4
6	5	0.778	0.632	8	-6.415	1	0.125	0.0404	0.0	0.0912	0.827	140.951	224029	4
7	6	0.666	0.589	0	-8.405	0	0.324	0.555	0.0	0.114	0.776	74.974	146053	4
8	7	0.922	0.712	7	-6.024	1	0.171	0.0779	3.96e-05	0.175	0.904	104.964	161800	4
9	8	0.794	0.659	7	-7.063	0	0.0498	0.143	0.00224	0.0944	0.308	112.019	247460	4
10	9	0.853	0.668	3	-6.995	1	0.447	0.263	0.0	0.104	0.745	157.995	165363	4

Рисунок 5.5.1 — Вигляд перших 10 елементів датасету.

5.5.1 Налаштування параметрів

В нейронній мережі потрібно налаштувати наступні параметри, такі як:

1. α — коефіцієнт швидкості навчання;
2. Iterations — кількість ітерацій навчання нейронної мережі;
3. Layers — кількість приховних шарів та їхня кількість нейронів.

Спочатко підберемо параметр α. При його підборі, важливо щоб нейронна мережа не вчилась швидко і не вчилась повільно. Результати підбору зображені на рис 5.5.1.1.

З рис. 5.5.1.1 видно, що при значення $\alpha = 0.1$, наша нейронна мережа вчиться надто швидко, що зможе спричинити до перенавчання мережі, це означає що дана мережа буде видавати помилкові результати для нових даних.

При значенні $\alpha < 0.01$, нейронна мережа вчиться надто повільно, тому нам це так само не підходить.

Значення $\alpha = 0.01$ є оптимальним для вирішення даної задачі.

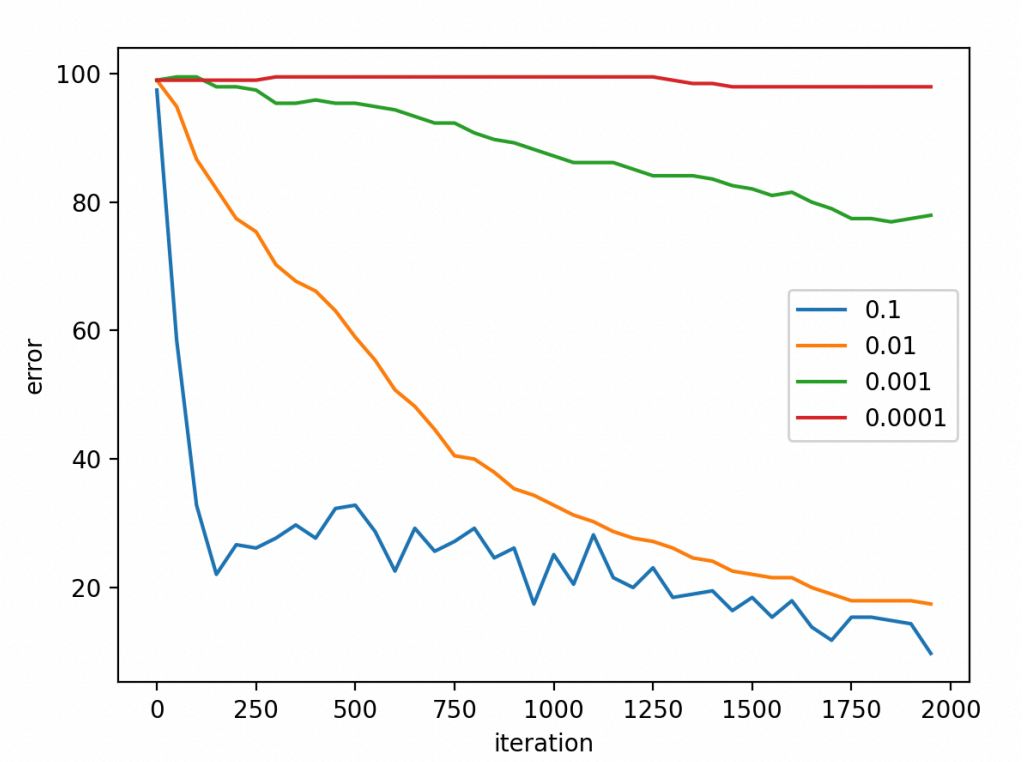

Рисунок 5.5.1.1 — Підбір параметру α.

Далі підберемо параметр Iterations, резальтати зображані на рис. 5.5.1.2.

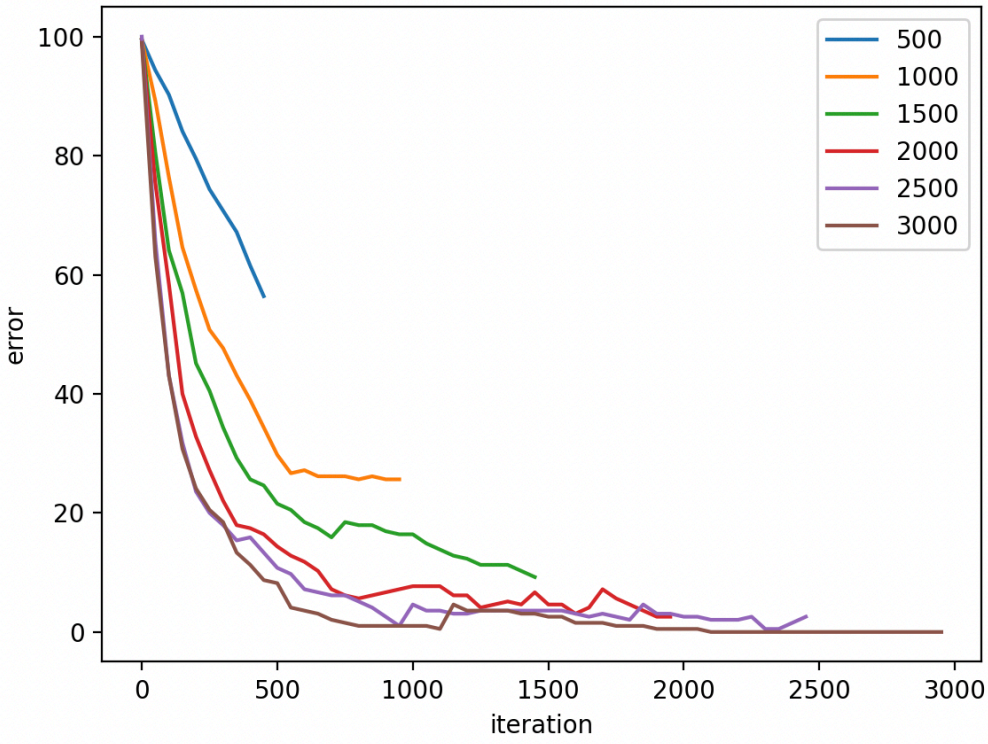

Рисунок 5.5.1.2 — Підбір параметру Iterations.

З рис 5.5.1.2 видно, що в деяких варіантах не достатня кількість інтерацій було проведена для навчання мережі, а в деяких мережа починала перенавчатись відповідно це спричинило б помилкові результати. Хорошими значеннями ітерацій для даної задачі є в проміжку [800, 1100].

Останній параметр, який потрібно вибрати це Layers. Підберемо приховані шари та кількість їх нейронів. Результати можна побачити на рис. 5.5.1.3.

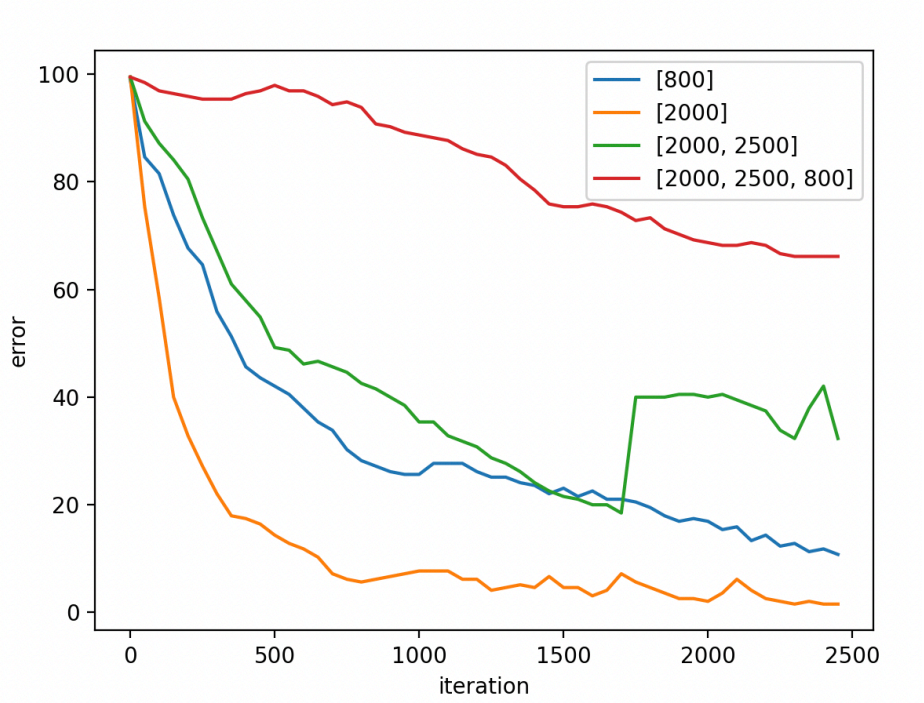

Рисунок 5.5.1.3 — Підбір параметру Layers.

З рис. 5.5.1.3 можна побачити деяку залежність між параметрами Layers та Iterations. Порівнюючи один шар з 800 нейронами та 2000 нейронами бачимо, що чим більше нейронів, тим менше помилка. А при двох - трьох шарах, помилка зменшується повільно, а в деяких випадках і зростає. Далі будемо продовжувати працювати з одним шаром.

5.5.2 Демонстрація роботи

Було навчано нейронну мережу з наступнимим параметрами:

1. α — 0.05;
2. Iterations — 900;

3. Layers — [4000].

Отримали наступний графік (рис. 5.5.2.1) відношення відсотків помилок до кількості ітерацій навчання мережі.

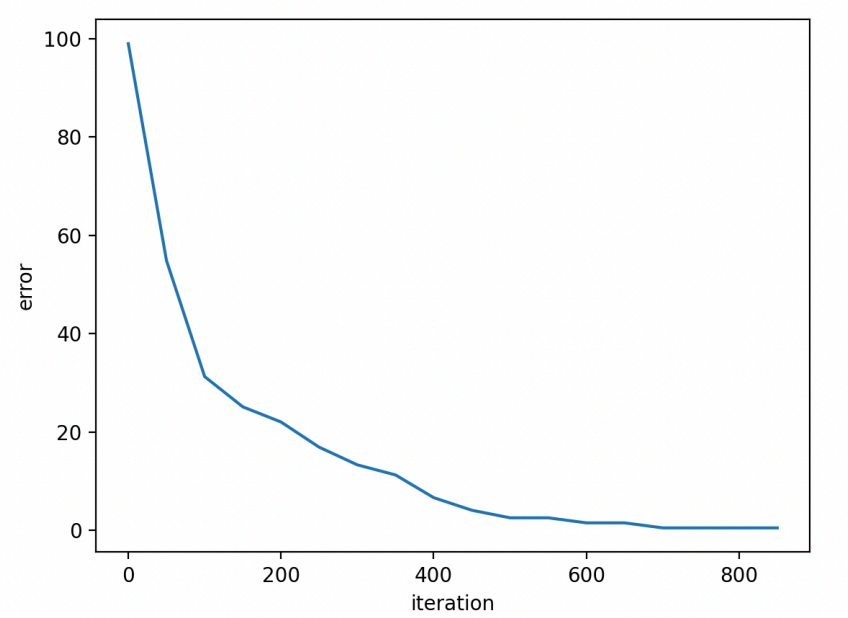

Рисунок 5.5.2.1 — Відношення помилки нейроннох мережі до кількості ітерацій.

З графіку видно, що з більшою кількістю ітерацій, нейронна мережа менше помиляється, це означає що точність буде більшою. На даному графіку в результаті отримано помилку в 0.51%, відповідно точність нейронної мережі становитиме 99.49%.

Було взято вибірку з унікальних ідентифікаторів пісень [2, 5, 46, 80] для надання рекомендацій. Був отриманий наступний результат (рис. 5.5.2.2), це є набір значень, який містить в собі унікальні ідентифікатори пісень, які можуть сподобатись користувачу на основі його даних.

```
Prediction: {1, 33, 3, 36, 65, 6, 7, 8, 71, 44, 77, 78, 82, 21}
```

Рисунок 5.5.2.2 — Результат рекомендацій.

ВИСНОВКИ

Протягом виконання цієї робити, була використана бібліотека numpy, за допомогою якої ми програмно реалізували алгоритми надання рекомендацій.

Було реалізовано наступні алгоритми надання рекомендацій:

1. Міра подібності за коефіцієнтом Жаккара;
2. Косинусна міра подібності;
3. Надання рекомендацій за допомогою метода k-найближчих сусідів;
4. Надання рекомендацій за допомогою алгоритма SVD;
5. Надання рекомендацій за допомогою нейронної мережі з активаційною функцією softmax.

На основі даних алгоритмів створюються рекомендаційні системи для різних додатків. Також був проведений аналіз їхніх переваг та недоліків. Дана бібліотека вирішує задачі з наданням рекомендацій для користувачів, використовуючи інформацію про їхні вподобання або характеристики предметів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Опис рекомендаційних систем. [Електронний ресурс] - Режим доступу: https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26
2. Опис колаборативної фільтрації. [Електронний ресурс] - Режим доступу: https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0
3. Опис будови рекомендаційних систем з колаборативною фільтрацією. [Електронний ресурс] - Режим доступу: https://realpython.com/build-recommendation-engine-collaborative-filtering/
4. Створення рекомендаційних систем. [Електронний ресурс] - Режим доступу: https://dou.ua/lenta/articles/creating-megogo-system-1/
5. Опис алгоритму k-найближчих сусідів. [Електронний ресурс] - Режим доступу: https://aman-makwana101932.medium.com/understanding-recommendation-system-and-knn-with-project-book-recommendation-system-c648e47ff4f6
6. Опис алгоритму k-найближчих сусідів. [Електронний ресурс] - Режим доступу: https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea
7. Опис алгоритму косинусної міри подібності. Електронний ресурс] - Режим доступу: https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599
8. Опис коефіцієнта Жаккара. [Електронний ресурс] - Режим доступу: https://www.learndatasci.com/glossary/jaccard-similarity/
9. Факторизація матриць. [Електронний ресурс] - Режим доступу: https://developers.google.com/machine-learning/recommendation/collaborative/matrix
10. Опис алгоритму SVD. [Електронний ресурс] - Режим доступу: https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm
11. Опис нейронної мережі. [Електронний ресурс] - Режим доступу: https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65

ДОДАТОК

В даному додатку, можна побачити код написаних алгоритмів.

Косинусна міра подібності:

```
import numpy as np

from numpy import dot

from numpy.linalg import norm

from matplotlib import pyplot as plt



class CosineRecommendation:

    def __init__(self, items):

        self.items = items

        self.matrix = np.zeros((len(items), len(items)))

        self.items_length = len(items)

        self.ids = [i[0] for i in items]

        print(self.ids)

        print('Cosine Similarity initialized (Item based)')


    def build_recommendation_matrix(self):

        for i in range(self.items_length):

            for j in range(self.items_length):

                self.matrix[(i, j)] = round(self.cosinus_index(self.get_attributes(self.items[i]), self.get_attributes(self.items[j])), 2)
```

```
def build_plot(self):
    fig, ax = plt.subplots()
    ax.matshow((self.matrix), cmap='RdYlGn')
    plt.rcParams['axes.xmargin'] = 0
    ax.xaxis.set_ticks(range(len(self.ids)), np.array(self.ids).astype(int))
    ax.yaxis.set_ticks(range(len(self.ids)), np.array(self.ids).astype(int))
    for i in range(self.items_length):
        for j in range(self.items_length):
            c = self.matrix[(j, i)]
            ax.text(i, j, (str(c)), va='center', ha='center')

    plt.show()

def get_recommendation(self, to_recommend_items):
    result = {}
    recommend_ids = [self.ids.index(i) for i in to_recommend_items]
    for item in to_recommend_items:
        item_index = self.ids.index(item)
        for i in range(self.items_length):
            if i not in recommend_ids:
                while result.get(self.ids[i]):
                    result[self.ids[i]].append(self.matrix[(item_index, i)])
```

```
                result[self.ids[i]] = [self.matrix[(item_index, i)]]

        result_tuple = [(
         int(item_id), round(sum(result[item_id]) / len(result[item_id]), 3)) for item_id in
result.keys()]
        result_tuple.sort(key=(lambda x: x[1]), reverse=True)
        return result_tuple

    @staticmethod
    def get_attributes(item):
        return list(item[1:])

    @staticmethod
    def cosinus_index(item1, item2):
        result = dot(item1, item2) / (norm(item1) * norm(item2))
        return result

np.set_printoptions(suppress=True)

def handle_cosine():
    cosine_data_filename = 'nutrients_csvfile.csv'
    df = pd.read_csv(cosine_data_filename)
    df = df.drop(df.columns[[0, 1, 2, 3, 9]], axis=1)
```

```
    df = df.replace("t", 0)
    df = df.astype(float)
    df.insert(0, "id", [i for i in range(0, len(df.index))], True)

    cs = CosineRecommendation(df.to_numpy()[0:9])
    cs.build_recommendation_matrix()
    cs.build_plot()


if __name__ == '__main__':
    handle_cosine()
```

Алгоритм k-найближчих сусідів:

```
import numpy as np


class KN:
    def __init__(self):
        print('KN init')

    def unique(self, array):
        uniq, index = np.unique(array, return_index=True)
        return uniq[index.argsort()]
```

```
    def mode(self, input_arr):
        return np.apply_along_axis((lambda x: np.bincount(x).argmax()), axis=0,
arr=input_arr)

    def euclidic_distance(self, a, b):
        distance = np.sqrt(np.sum((b - a) ** 2))
        return distance

    def train(self, x_train, item):
        point_dist = []
        for j in range(len(x_train)):
            distances = self.euclidic_distance(np.array(x_train[j]), item)
            point_dist.append(distances)

        point_dist = np.array(point_dist)
        return point_dist

    def predict(self, x_train, y, x_input, k):
        op_labels = []
        total_labels = []
        for item in x_input:
            point_dist = self.train(x_train, item)
```

```
            dist = np.argsort(point_dist)[:k]
            labels = np.array(y)[dist.astype(int)]
            total_labels = np.concatenate((total_labels, labels))
            lab = self.mode(labels)
            op_labels.append(lab)

        return np.array(self.unique(total_labels)).astype(int)


np.set_printoptions(suppress=True)


def handle_kn():
    kn_data_filename = 'wine.csv'
    df = pd.read_csv(kn_data_filename)

    Y = np.array(df['Wine'])
    X = df.drop(df.columns[[0]], axis=1).to_numpy()

    train_idx = [i for i in range(0, 100)]
    X_train = X[train_idx]
    y_train_labels = Y[train_idx]
```

```
    # Creating the testing Data
    test_idx = [i for i in range(100, 105)]
    X_test = X[test_idx]


    kn = KN()


    print(kn.predict(X_train, y_train_labels, X_test, 4))



if __name__ == '__main__':
    handle_kn()
```

Коефіцієнт Жаккара:

```
class Jaccard:
    def __init__(self):
        print('Jaccard init')


    def jaccard_index(self, set1, set2):
        set1, set2 = set(set1), set(set2)
        result = float(len(set1.intersection(set2))) / float(len(set1.union(set2)))
        return result


    def get_similar(self, id_liked, data: dict):
```

```
    similar = []
    for other_id, other_liked in data.items():
        if id_liked != other_liked:
            jaccard_coef = self.jaccard_index(id_liked, other_liked)
            if jaccard_coef:
                similar.append((jaccard_coef, other_id))

    similar.sort(reverse=True, key=lambda t: t[0])
    return similar

def normalize(self, id: int, data: dict, sim_ids):
    count = 0
    for sim_id in sim_ids:
        if id in data[sim_id]:
            count += 1
    return count

def recommend(self, data: dict, id_to_recommend: int):
    id_liked = data[id_to_recommend]
    similar = self.get_similar(id_liked, data)

    rank = {}
    for j_idx, id in similar:
```

```
            for item in data[id]:
                if item not in id_liked:
                    rank.setdefault(item, 0)
                    rank[item] += j_idx

        recommendations = []
        for id, j_dx in rank.items():
            occurs = self.normalize(id, data, [i[1] for i in similar])
            recommendations.append((j_dx / occurs, id))


if __name__ == '__main__':
    data = {
        15: [16, 17, 19, 20],
        16: [21, 22, 23, 25],
        17: [21, 23, 17],
        19: [25, 19, 23],
        20: [16, 23, 20]
    }

    jaccard = Jaccard()

    print(jaccard.recommend(d, 20))
```

Алгоритм SVD:

```
import numpy as np
import pandas as pd



class SVDAlg:
    def __init__(self, matrix, k_groups, alpha, beta, iterat_count):
        self.R = matrix
        self.number_users, self.number_items = matrix.shape
        self.K = k_groups
        self.alpha = alpha
        self.beta = beta
        self.iterations = iterat_count
        self.P = 0
        self.Q = 0
        self.b_u = []
        self.b_i = []
        self.b = 0
        self.samples = []


    def get_samples(self):
        return [(i, j, self.R[i, j]) for i in range(self.number_users) for j in
range(self.number_items) if self.R[i, j] > 0]
```

```
def train(self):
    self.P = np.random.normal(scale=1. / self.K, size=(self.number_users, self.K))
    self.Q = np.random.normal(scale=1. / self.K, size=(self.number_items, self.K))

    self.b_u = np.zeros(self.number_users)
    self.b_i = np.zeros(self.number_items)
    self.b = np.mean(self.R[np.where(self.R != 0)])

    self.samples = self.get_samples()

    training_results_arr = []
    for i in range(self.iterations):
        np.random.shuffle(self.samples)
        self.fit()
        mse = self.calc_error()
        training_results_arr.append((i, mse))
        if (i + 1) % 20 == 0:
            print(f'Number of iteration: {i + 1}; error value = {mse}')

    return training_results_arr

def calc_error(self):
```

```
    xs, ys = self.R.nonzero()
    prognosticated = self.get_full_matrix(self)
    error_value = 0
    for x, y in zip(xs, ys):
        error_value += np.pow(self.R[x, y] - prognosticated[x, y], 2)
    return np.sqrt(error_value / len(xs))

def fit(self):
    for i, j, r in self.samples:
        ratings = self.get_rating(i, j)
        e = (r - ratings)

        self.b_u[i] = self.b_u[i] + (self.alpha * (e - self.beta * self.b_u[i]))
        self.b_i[j] = self.b_i[j] + (self.alpha * (e - self.beta * self.b_i[j]))

        self.P[i, :] = self.P[i, :] + (self.alpha * (e * self.Q[j, :] - self.beta * self.P[i, :]))
        self.Q[j, :] = self.Q[j, :] + (self.alpha * (e * self.P[i, :] - self.beta * self.Q[j, :]))

def get_rating(self, i, j):
    return self.b + self.b_u[i] + self.b_i[j] + self.P[i, :].dot(self.Q[j, :].T)

@staticmethod
def get_full_matrix(mf):
```

```
        return mf.b + mf.b_u[:, np.newaxis] + mf.b_i[np.newaxis:, ] + mf.P.dot(mf.Q.T)
```

Генератор даних SVD:

```
import numpy as np
import pandas as pd



def populate_data():
    rand_nums = [2, 3, 5, 10]

    place_ids = [i for i in range(1, 201)]
    user_ids = [i for i in range(1, 51)]
    rating = [1, 2, 3, 4, 5]

    result_user_id = []
    result_place_id = []
    result_rate = []
    for i in user_ids:
        for j in place_ids:
            rate = 0
            if (j + i) % rand_nums[np.random.randint(0, 3)] == 0:
                rate = rating[np.random.randint(0, 5)]
            result_user_id.append(i)
```

```
            result_place_id.append(j)
            result_rate.append(rate)

    d = {'user_id': result_user_id, 'place_id': result_place_id, 'rate': result_rate}
    df = pd.DataFrame(data=d)
    df.to_csv('data.csv', index=False, header=False)

populate_data()
```

Тестування SVD:

```
import pandas as pd
import numpy as np

from SVD.SVD import SVDAlg


def run_svd():
    r_cols = ['user_id', 'place_id', 'rating']
    ratings_train = pd.read_csv('SVD/data.csv', names=r_cols,
                                dtype={'user_id': np.int16, 'place_id': np.int16, 'rating': np.float16})

    R = np.array(ratings_train.pivot(index='user_id', columns='place_id',
values='rating').fillna(0))
```

```
    mf = SVDAlg(R, k_groups=20, alpha=0.001, beta=0.01, iterat_count=60)
    mf.train()
    x = mf.get_full_matrix(mf)
    dg = pd.DataFrame(x)
    dg.to_csv('recommended_matrix.csv', index=False, header=False)


if __name__ == '__main__':
    run_svd()
```

Нейронної мережа з активаційною функцією softmax:

```
import numpy as np
import matplotlib.pylab as plt
from keras.utils import np_utils
import pandas as pd

np.set_printoptions(suppress=True)


def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

```
def softmax(z):
    exp_z = np.exp(z - np.max(z))
    return exp_z / exp_z.sum(axis=0, keepdims=True)



def sigmoid_deriv(z):
    s = 1 / (1 + np.exp(-z))
    return s * (1 - s)



class ANN:
    def __init__(self, layers_size):
        self.layers_size = layers_size
        self.params = {}
        self.L = len(self.layers_size)
        self.n = 0
        self.costs = []
        self.plot_value_x = []
        self.plot_value_y = []
        self.lr = 0
        self.iter = 0

    def init_params(self):
```

```
        np.random.seed(1)


        for l in range(1, len(self.layers_size)):
            self.params["W" + str(l)] = np.random.randn(self.layers_size[l],
self.layers_size[l - 1]) / np.sqrt(
                self.layers_size[l - 1])
            self.params["b" + str(l)] = np.zeros((self.layers_size[l], 1))


    def forward_feet(self, X):
        results = {}


        A = X.T
        for l in range(self.L - 1):
            Z = self.params["W" + str(l + 1)].dot(A) + self.params["b" + str(l + 1)]
            A = sigmoid(Z)
            results["A" + str(l + 1)] = A
            results["W" + str(l + 1)] = self.params["W" + str(l + 1)]
            results["Z" + str(l + 1)] = Z


        Z = self.params["W" + str(self.L)].dot(A) + self.params["b" + str(self.L)]
        A = softmax(Z)
        results["A" + str(self.L)] = A
        results["W" + str(self.L)] = self.params["W" + str(self.L)]
```

```
    results["Z" + str(self.L)] = Z

    return A, results

def backward_feet(self, X, Y, results):
    derivs = {}

    results["A0"] = X.T

    A = results["A" + str(self.L)]
    d_z = A - Y.T

    d_w = d_z.dot(results["A" + str(self.L - 1)].T) / self.n
    d_b = np.sum(d_z, axis=1, keepdims=True) / self.n
    d_a_prev = results["W" + str(self.L)].T.dot(d_z)

    derivs["dW" + str(self.L)] = d_w
    derivs["db" + str(self.L)] = d_b

    for l in range(self.L - 1, 0, -1):
        d_z = d_a_prev * sigmoid_deriv(results["Z" + str(l)])
        d_w = 1. / self.n * d_z.dot(results["A" + str(l - 1)].T)
        d_b = 1. / self.n * np.sum(d_z, axis=1, keepdims=True)
```

```
        if l > 1:
            d_a_prev = results["W" + str(l)].T.dot(d_z)

        derivs["dW" + str(l)] = d_w
        derivs["db" + str(l)] = d_b

    return derivs

def fit_net(self, X, Y, learning_rate=0.01, n_iterations=2500):
    np.random.seed(1)
    self.lr = learning_rate
    self.params = {}

    self.n = 0
    self.costs = []
    self.plot_value_x = []
    self.plot_value_y = []

    self.iter = n_iterations

    self.n = X.shape[0]

    self.layers_size.insert(0, X.shape[1])
```

```
self.init_params()
for loop in range(n_iterations):
    A, results = self.forward_feet(X)
    cost_value = -np.mean(Y * np.log(A.T))
    derivs = self.backward_feet(X, Y, results)

    for l in range(1, self.L + 1):
        self.params["W" + str(l)] = self.params["W" + str(l)] - learning_rate * derivs[
            "dW" + str(l)]
        self.params["b" + str(l)] = self.params["b" + str(l)] - learning_rate * derivs[
            "db" + str(l)]

    if loop % 100 == 0:
        print(f"Iteration #{loop}: Accuracy:", round(self.get_cost_value(X, Y), 2))

    if loop % 50 == 0:
        self.plot_value_x.append(loop)
        self.plot_value_y.append(100 - self.get_cost_value(X, Y))

    if loop % 10 == 0:
        self.costs.append(cost_value)
```

```
def get_cost_value(self, X, Y):
    A, _ = self.forward_feet(X)
    y_hat = np.argmax(A, axis=0)
    Y = np.argmax(Y, axis=1)
    accuracy = (y_hat == Y).mean()

    return accuracy * 100

def get_predictions(self, X, x_ids):
    A, results = self.forward_feet(X)
    A = np.array(A).T
    result = set()

    for i in range(len(A)):
        arr = np.array(A[i])
        res = arr.argsort()[-5:][::-1]
        result.update(res)

    return np.array(result - set(x_ids))

def plot_cost(self):
    plt.figure()
    plt.plot(np.arange(len(self.costs)), self.costs)
```

```
        plt.xlabel("epochs")
        plt.ylabel("cost")
        plt.show()

    def plot_error(self):
        plt.plot(self.plot_value_x, self.plot_value_y, label=self.layers_size[1:-1])
        plt.xlabel("iteration")
        plt.ylabel("error")
        plt.show()



def get_data():
    cosine_data_filename = 'parsed_data.csv'
    df = pd.read_csv(cosine_data_filename)

    train_x = np.array(df.loc[:, df.columns != 'duration_ms'])
    train_y = np_utils.to_categorical(np.array(df.iloc[:, 0]))

    return train_x, train_y, len(train_y[0])



if __name__ == '__main__':
    train_x, train_y, last_layer_shape = get_data()
```

```
layers_dims = [4000, last_layer_shape]


ann = ANN(layers_dims)
ann.fit_net(train_x, train_y, learning_rate=0.05, n_iterations=900)
ann.plot_error()


test_x = np.array([train_x[2], train_x[5], train_x[46], train_x[80]])
test_x_ids = [0, 2, 5, 46, 80]
print('\nPrediction:', ann.predict_items(test_x, test_x_ids))
```