НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи і методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки»

на тему: «Система персоналізованої рекомендації новин на основі текстових характеристик контенту»

Виконав:
студент IV курсу, групи КІ-13
Старокожко Антон Олегович _____

Керівник:
професор кафедри математичних методів системного аналізу,
д.т.н., доцент, Недашківська Надія Іванівна _____

Консультант з економічного розділу:
доцент кафедри економічної кібернетики,
к.е.н., доцент, Рощина Надія Василівна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри штучного інтелекту,
к.т.н., доцент, Комариста Богдана Миколаївна _____

Рецензент:
старший.викладач кафедри системного проектування,
к.т.н., Письменний Ігор Олександрович _____

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

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра штучного інтелекту

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«15» січня 2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Старокожка Антона Олеговича

1. Тема роботи «**Система персоналізованої рекомендації новин на основі текстових характеристик контенту**», керівник роботи Недашківська Надія Іванівна, доктор технічних наук, доцент, затверджені наказом по НН ІПСА від «26» травня 2025 р. № 1759-с.

2. Термін подання студентом роботи «09» червня 2025 року.

3. Вихідні дані до роботи: Набір новинних статей із текстовими описами та поведінковими даними користувачів

4. Зміст роботи: Аналіз предметної області дослідження та даних, вибір доцільних методів і моделей, розробка алгоритму обраними методами, оцінка отриманих результатів

5. Перелік ілюстративного матеріалу: електронна презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Надія Василівна, доцент, к. е. н.	03.02	09.06

7. Дата видачі завдання «03» лютого 2025 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Огляд літератури за темою	21.04.2025	Виконано
2	Підготовка першого розділу	28.04.2025	Виконано
3	Підготовка другого розділу	05.05.2025	Виконано
4	Розробка програмного продукту	12.05.2025	Виконано
5	Підготовка третього розділу	19.05.2025	Виконано
6	Оформлення дипломної роботи	02.06.2025	Виконано
7	Підготовка презентації доповіді	09.06.2025	Виконано

Студент Антон СТАРОКОЖКО

Керівник Надія НЕДАШКІВСЬКА

РЕФЕРАТ

Дипломна робота: 123 с., 20 рис., 8 табл., 30 посилань, додаток.

ГЛИБОКЕ НАВЧАННЯ, ЕМБЕНДИНГИ, МОДЕЛЬ DKN, XLEARN, PYTORCH, ПРОБЛЕМА ХОЛОДНОГО СТАРТУ, ГРАФ ЗНАНЬ, МЕХАНІЗМ УВАГИ, PYTHON

Об’єктом дослідження є система персоналізованої рекомендації новин.

Предметом дослідження є методи та моделі побудови рекомендаційних систем з використанням текстових характеристик контенту.

Метою роботи є розробка та оцінка ефективності системи персоналізованих рекомендацій новин, яка враховує текстові особливості контенту та поведінкові дані користувачів.

У дипломній роботі оглянуто сучасні підходи до побудови рекомендаційних систем, зокрема контентну, колаборативну фільтрацію та гібридні методи. Проаналізовано переваги та недоліки кожного з підходів у контексті рекомендації новин. Розглянуто традиційні алгоритми (Factorization Machines, FFM) і сучасні нейронні архітектури (DKN), що використовують графи знань та механізм уваги.

Розроблено програмну систему персоналізованої рекомендації новин на основі текстових характеристик. Здійснено попередню обробку набору даних, що включає новинні статті та поведінкову інформацію користувачів. Виконано побудову та навчання моделей FFM і DKN. Проведено оцінку ефективності моделей за допомогою відповідних метрик точності.

Отримано результати, що демонструють перевагу нейромережевої моделі DKN у контексті коротких текстів і проблеми семантичної неоднозначності.

ABSTRACT

Bachelor's thesis: 123 p., 20 figures, 8 tables, 30 references, appendix

DEEP LEARNING, EMBEDINGS, MODEL DKN, XLEARN, PYTORCH, COLD START PROBLEM, KNOWLEDGE GRAPH, ATTENTION MECHANISM, PYTHON

The object of research is a system of personalized news recommendation.

The subject of the study is methods and models for building recommender systems using textual characteristics of the content.

The purpose of the study is to develop and evaluate the effectiveness of a personalized news recommendation system that takes into account textual features of the content and user behavioral data.

The thesis reviews modern approaches for building recommender systems, including content-based, collaborative filtering, and hybrid methods. The advantages and disadvantages of each approach in the context of news recommendation are analyzed. Traditional algorithms (Factorization Machines, FFM) and modern neural architectures (DKN) using knowledge graphs and the attention mechanism are considered.

A software system for personalized news recommendation based on textual characteristics is developed. The dataset, which includes news articles and user behavioral information, was pre-processed. The FFM and DKN models were built and trained. The efficiency of the models is evaluated using the relevant accuracy metrics.

The results demonstrate the advantage of the DKN neural network model in the context of short texts and the problem of semantic ambiguity.

ЗМІСТ

ВСТУП

Щодня[1] люди мають справу з величезними обсягами інформації. Кількість доступної інформації зросла до нечуваних раніше масштабів завдяки більш доступному Інтернету та появі численних онлайн-платформ. Новини, музика, відео, блоги, пости в соціальних мережах і реклама – все це безперервно оновлюється і створюється у великих масштабах. Проблема інформаційного перевантаження стає все більш нагальною, оскільки споживачі намагаються знайти справді цінний, цікавий чи корисний контент у постійному потоці інформації.

Ця проблема особливо помітна в медіабізнесі, оскільки платформи конкурують за увагу користувачів, пропонуючи дедалі більше контенту. Традиційних підходів до пошуку та фільтрації недостатньо, щоб задовольнити попит на персоналізований досвід. Сьогодні споживачі віддають перевагу платформам, які визначають їхні вподобання і надають контент, що відповідає їхнім інтересам, а не змушують їх скролити новинну стрічку у пошуку потрібного матеріалу.

Окремою категорією в цих системах є рекомендаційні системи новин, оскільки цей тип контенту – динамічний і відображає події в режимі реального часу. Побудова ефективних алгоритмів у цій галузі вимагає врахування не лише історії переглядів користувачів, але й текстових особливостей самої новини – теми, ключових слів, тональності, структури та інших семантичних властивостей.

---

[1] Тут і нижче використано такий інструмент штучного інтелекту як чат-бот з генеративним штучним інтелектом ChatGPT виключно для корегування та редагування тексту, створеного автором цієї дипломної роботи, на основі автоматизованої перевірки граматики, структури та стилю, що відповідає Політиці використання штучного інтелекту для академічної діяльності в КПІ ім. Ігоря Сікорського (протокол №11 Вченої ради КПІ ім. Ігоря Сікорського від 11 грудня 2023 р.).

РОЗДІЛ 1 СТАН І ТЕНДЕНЦІЇ РОЗВИТКУ СИСТЕМ РЕКОМЕНДАЦІЙ У СФЕРІ НОВИН

1.1 Актуальність проблеми

Сьогодні ми все більше звертаємо увагу на онлайн-платформи, як-от веб-сайти та мобільні додатки, коли споживаємо контент. Телебачення та друковані видання вже не такі популярні, бо цифрові технології розвиваються, а ми, споживачі, хочемо більше інтерактивності та персоналізації [1].

Нові медіа-платформи з'явилися повсюдно в результаті вдосконалення технологій, змінюючи не лише способи отримання інформації, але й те, як споживачі взаємодіють з матеріалами [2].

Користувачі більше зацікавлені в інтерактивному досвіді у нових медіа. Вони можуть брати активну участь через блоги, форуми, соціальні мережі та інші канали. Залишаючи коментарі, репости та поширюючи медіаконтент, користувачі перетворилися з пасивних споживачів інформації на активних розповсюджувачів.

Також, у контексті нових медіа користувачі демонструють унікальну різноманітність. Завдяки постійному розвитку нових технологій користувачі мають доступ до широкого і різноманітного спектру медіа-каналів і матеріалів, включаючи текст, фотографії, аудіо та відеоматеріали.

Користувачі дедалі більше зацікавлені у задоволенні своїх індивідуальних потреб. За допомогою інтернету користувачі можуть вільно обирати та змінювати контент відповідно до власних потреб, інтересів та смаків. Нові медіа можуть надавати контент і послуги, які краще відповідають індивідуальним запитам споживачів, використовуючи такі стратегії, як рекомендаційні системи, що покращують користувацький досвід загалом.

Розвиток цифрових технологій та зростаючий попит на більш інтерактивні та персоналізовані формати, безумовно, є причиною зміщення фокусу споживачів від традиційних медіа до онлайн-платформ. Крім того, використання

рекомендаційних систем є важливим для покращення користувацького досвіду, оскільки вони дозволяють обирати та змінювати рекомендації відповідно до індивідуальних потреб. Дослідження сучасних тенденцій споживання медіаконтенту має вирішальне значення для розвитку нових медіа та взаємодії користувачів з ними, особливо враховуючи стрімкий розвиток технологій та зростаючу важливість індивідуальних запитів.

1.2 Розвиток систем для підбору контенту

Перші спроби для розвитку рекомендаційних систем були досить примітивними, вони включали рекомендації на основі популярності товарів у певній категорії або прості рейтингові системи, де користувачам пропонувались найпопулярніші товари чи послуги. Такі алгоритми відображали базову стратегію відбору, але при цьому не враховували персональні вподобання кожного із користувачів [3].

Першим значним кроком стали системи колаборативної фільтрації, які почали використовувати історію взаємодії користувачів з контентом або товаром для прогнозування майбутніх рекомендацій. Ці системи ґрунтувалися на тому, що люди зі схожими оцінками контенту мають схожі вподобання. Алгоритми рекомендацій, які використовуються на багатьох онлайн-платформах і в роздрібній торгівлі, були розроблені з використанням цієї методології. Наприклад, Amazon почав використовувати колаборативну фільтрацію в 1990-х роках, щоб надавати рекомендації щодо товарів на основі минулих покупок клієнтів, які були схожі на попередні [4].

Пізніше з'явилися системи, побудовані з використанням більш досконалих методів, таких як глибоке навчання та нейронні мережі. Навіть якщо попередні вподобання користувача були нечіткими, це дозволило врахувати набагато більше факторів і зробити набагато кращі прогнози щодо того, який контент буде

для нього привабливим. Ці методи набули популярності на потокових платформах, таких як YouTube [5] і Spotify [6], Netflix [7]. Використання таких моделей, як нейронні мережі з глибоким навчанням, дає змогу значно покращити якість рекомендацій, адже вони здатні навчатися на величезних обсягах даних, включаючи як явні вподобання (наприклад, оцінки контенту), так і приховані патерни поведінки, які можуть не бути очевидними на перший погляд. Зокрема, через використання рекурентних нейронних мереж (RNN) і трансформерів для аналізу послідовностей переглядів і взаємодій, ці системи здатні враховувати контекст користувацької поведінки в часі, а також виконувати точне прогнозування, базуючись на складних зв'язках між різними категоріями контенту та інтересами користувачів.

Завдяки новим досягненням, системи рекомендацій постійно вдосконалюються. Сучасні великі мовні моделі (ВММ), зокрема ті, що працюють на основі трансформерів, не лише краще передбачають інтереси користувачів, а й глибше розуміють контекст і складні запити. Наприклад, на платформах на кшталт Google News або TikTok ці моделі допомагають персоналізувати контент, навіть якщо смаки користувача змінюються. У результаті системи стають більш гнучкими, що покращує досвід користувачів [8].

Розвиток рекомендаційних систем демонструє перехід від простих евристичних методів до складних інтелектуальних алгоритмів, які можуть досліджувати складні взаємодії між користувачем і контентом. Кожен крок процесу розробки, від простих систем, до більш складних методів, що використовують глибоке навчання і ВММ, створював нові можливості для поліпшення релевантності та персоналізації пропозицій. Ці модифікації не лише покращили користувацький досвід, але й відіграли значну роль в успіху цифрових платформ на ринку.

1.3 Рекомендаційні системи

У попередньому розділі ми визначили, що перехід від простих евристичних методів підбору контенту відбувся завдяки впровадженню рекомендаційних систем. У цьому підрозділі буде подано загальний огляд поняття рекомендаційних систем та особливостей їх застосування в сучасних умовах.

Рекомендаційні системи – це програми, алгоритми або інструменти, які надають користувачам рекомендації або ідеї щодо речей, які можуть їх зацікавити. Залежно від галузі, це можуть бути люди, книги, фільми, готелі, ліки тощо. Рекомендаційні системи використовуються для генерування індивідуальних рекомендацій, щоб допомогти людині у прийнятті рішень, передбачаючи вподобання кожного користувача на основі зібраної про нього інформації та властивостей об’єктів для рекомендації [9].

Дані системи почали активно використовуватися із появою, веб-сайтів, із широким вибором товарів, таких як Amazon. Вони набули популярності, через автоматизацію підбору товарів для кінцевого користувача, оскільки він не мав ані достатньої експертизи, ані часу для перегляду всіх можливих варіантів. Для розуміння, того, яку рекомендацію надати людині, системі потрібно збирати дані, це може бути зроблено, як прямим чином, через збір фідбек на товар, що включає розгорнуті коментарі, оцінку за п’ятибальною шкалою або ж і неявним чином, основуючись на даних про вже переглянуті товари, або інших попередніх дій користувача. Чим більше історичних даних зібрано для кожного унікального користувача тим краще система зможе зробити прогноз [4].

Оскільки вони допомагають користувачам знаходити інформацію та товари, які може бути складно знайти самостійно, рекомендаційні системи є альтернативою пошуковим системам, а також використовуються для прогнозу рекомендацій. До найвідоміших інтернет-порталів, які використовують алгоритми рекомендаційних систем, належать Amazon, eBay, Netflix, Rozetka,

Spotify та YouTube. Також цей тип систем, широко використовується і для рекомендацій новин на таких майданчиках, як Microsoft News, BBC News, New York Times та TSN

1.4 Методи побудови рекомендаційних систем

Зараз колаборативна фільтрація, контентна фільтрація та гібридні підходи – це методи, які найбільш часто використовують для створення рекомендаційних систем. Залежно від того, які дані є і які завдання стоять перед системою, у кожного з них є свої унікальні плюси, мінуси та області застосування. Щоб краще зрозуміти, як працює кожен підхід і коли його варто використовувати, давайте подивимося на їхні основні характеристики та відмінності.

Системи рекомендацій на основі контенту вивчають раніше оцінений контент або об'єкти і створюють модель або профіль уподобань користувача на основі цих особливостей. Характеристики раніше вподобаних об'єктів, такі як жанр фільму, тема новини, стиль пісні, складові продукту або інші релевантні метадані, беруться до уваги під час аналізу. Алгоритм здатний скласти широку картину вподобань користувача, використовуючи ці атрибути для пошуку спільних рис у вподобаних речах. У цьому профілі встановлюються смаки користувача, що є основою для створення пропозицій щодо нових об’єктів. Профіль зазвичай реалізується у вигляді векторного представлення, де кожна позиція представляє певний атрибут контенту, а його значення вказує на зацікавленість користувача в цьому атрибуті. Наприклад, у профілі користувача переважатимуть науково-популярні статті, якщо він регулярно їх читає. У результаті система створює персоналізовану карту інтересів, яка часто оновлюється відповідно новій інформації. Основними недоліками, такого типу фільрації, є проблема холодного старту для користувачів та проблема виділення ознак об’єкту.

Процес рекомендації порівнює ознаки елементів контенту із профілем користувача. Обчислюючи схожість між вектором профілю та векторами нових сутностей контенту, система тепер шукає елементи, які найбільше відповідають інтересам користувача. У простих реалізаціях можна використовувати косинусну подібність, евклідову відстань та інші метрики подібності. Більш складні стратегії включають методи машинного навчання, такі як регресійні або класифікаційні моделі, засновані на минулих даних. Результатом цієї процедури є оцінка релевантності, яка вказує на ймовірний ступінь зацікавленості користувачів у запропонованому товарі.

Колаборативна фільтрація використовує оцінки, які користувач та інші користувачі присвоїли об'єктам. Вона ґрунтується на тому, що люди, швидше за все, зберігатимуть подібні вподобання в майбутньому, якщо вони раніше давали певним об'єктам співставні оцінки. Таким чином, можна зробити висновок, що оцінка нового товару Користувачем А, ймовірно, збігатиметься з оцінкою, яку поставив би Користувач Б, якби він раніше давав аналогічні оцінки тому самому набору об’єктів. Це дає змогу системі передбачити, які об’єкти із найбільшою ймовірністю обрав би користувач, навіть якщо вони ще не були розглянуті [10].

Два основних методи колаборативної фільтрації – на основі об’єктів (item based) і на основі користувачів (user based). При використанні методу на основі користувачів алгоритм знаходить користувачів зі схожими моделями оцінювання і пропонує товари, яким ці користувачі поставили високі оцінки. При використанні методу на основі об’єктів система пропонує користувачеві об’єкти на основі об’єктів, які перед цим вже обирав користувач, та їхньої схожості до нових об’єктів, які ще не були розглянуті.

Колаборативна фільтрація використовує лише дані про взаємодію користувача з об'єктом у минулому (наприклад, оцінки, покупки або кліки) і не вимагає попередніх знань про атрибути самих об'єктів. Однак він може зіткнутися з такими труднощами, як розрідженість (велика кількість відсутніх значень у матриці «користувач-об'єкт») і проблема «холодного старту» (відсутність інформації про нових користувачів або нові об'єкти). У гібридних

рекомендаційних системах колаборативна фільтрація часто використовується в поєднанні з іншими стратегіями, в тому числі з фільтрацією на основі контенту, щоб подолати ці недоліки.

Гібридні підходи стають дедалі популярнішими як спосіб підвищити точність і релевантність висновків, одночасно долаючи недоліки окремих підходів до розробки рекомендаційних систем. Ці стратегії поєднують переваги різних підходів, таких як, колаборативна фільтрація та контентно-орієнтована фільтрація. Інтеграція багатьох джерел інформації, включаючи історію взаємодії користувача, описовий зміст запропонованих об'єктів і контекстні аспекти (наприклад, час, місце, тип пристрою), стає можливою завдяки гібридизації.

Однак існує низка обчислювальних і технічних труднощів, пов'язаних з використанням гібридних методів. Зокрема, ці системи повинні аналізувати величезні обсяги різноманітних даних, інтегрувати різнорідні моделі (наприклад, змішування глибоких нейронних мереж і матричної факторизації) та мають значну кількість гіперпараметрів. Крім того, оцінка продуктивності гібридних систем вимагає більш складних метрик.

Гібридні рекомендаційні системи довели свою високу ефективність у різних галузях. Одним з таких прикладів гібридної архітектури є механізм рекомендацій Netflix [7], який поєднує аналіз контенту та колаборативну фільтрацію для надання індивідуальних пропозицій щодо відео. Схожим чином Spotify [10] використовує гібридний алгоритм, який створює музичні пропозиції, поєднуючи історію прослуховування користувача, метадані треків і схожі дії користувачів. Іншою ілюстрацією є система рекомендацій від Amazon [11], яка надає релевантні рекомендації щодо товарів, поєднуючи контент-аналіз, видобування правил асоціацій та спільну фільтрацію. Гібридні моделі, які враховують поточні тенденції та контекстні елементи, також часто використовуються новинними агрегаторами, такими як Google News і Microsoft News [12]. Нижче представлено порівняльну таблицю цих методів для підведення висновків стосовно різних методів для побудови систем (табл. 1.1).

Таблиця 1.1 – Порівняльний аналіз методі для рекомендаційних систем

Метод	*Переваги*	*Недоліки*
Контентна фільтрація	Створює персоналізований профіль користувача, Добре працює з новими/рідкісними об’єктами, Не потребує даних інших користувачів	Проблема холодного старту для нових користувачів, Складність у виділенні релевантних ознак, Ризик "інформаційної бульбашки"
Колаборативна фільтрація	Базується на вподобаннях реальних користувачів, Виявляє неочевидні зв'язки між об’єктами, Не вимагає метаданих про об’єкти	Проблема холодного старту (нові користувачі/об’єкти), Розрідженість даних, Зниження ефективності при малій активності користувачів
Гібридна система	Поєднує переваги контентної та колаборативної фільтрації, Покращує точність рекомендацій	Висока обчислювальна складність, Складна інтеграція різних моделей, Велика кількість гіперпараметрів, Ускладнене оцінювання якості рекомендацій

З порівняльного аналізу основних методів розробки рекомендаційних систем можна зробити висновок, що кожен з них має унікальні переваги та

недоліки, які впливають на те, наскільки доцільно використовувати їх у конкретному застосуванні. Хоча контент-орієнтовані системи пропонують високий ступінь персоналізації, вони не здатні ефективно обслуговувати нових користувачів і покладаються на повноту і якість описових характеристик об'єктів. Хоча колаборативна фільтрація може виявити приховану схожість між об'єктами або особами, вона має проблеми розрідженості та «холодного старту», коли не вистачає даних. Поєднуючи переваги фундаментальних методів, гібридні системи демонструють підвищену гнучкість і точність рекомендацій, але вони також характеризуються більшою складністю обчислень та інтеграції моделей.

Висновки до розділу 1

У першому розділі здійснено дослідження предметної області, зокрема проаналізовано актуальність застосування рекомендаційних систем у сучасному цифровому середовищі. Встановлено, що розвиток нових медіа, зміна споживацьких звичок та прагнення до персоналізації контенту зумовили стрімкий перехід уваги користувачів до онлайн-платформ, де рекомендаційні алгоритми відіграють ключову роль у взаємодії з контентом.

Зокрема, розглянуто три основні стратегії розробки рекомендаційних систем – контента фільтрація, колаборативна фільтрація та гібридні підходи. Виявлено, що кожна стратегія має свої переваги та недоліки. Наприклад, контента фільтрація дає змогу враховувати вподобання користувача на основі атрибутів об'єктів, колаборативна фільтрація знаходить схожі моделі поведінки користувачів, а гібридні стратегії поєднують переваги обох стратегій для підвищення точності та адаптивності.

У розділі також обговорюються питання розрідженості даних, «холодного старту» та оцінки рекомендацій. Щоб проілюструвати велику практичну користь

рекомендаційних систем, розглянуто приклади їхнього впровадження у компаніях.

Таким чином, системи рекомендацій є важливим компонентом сучасних інформаційних технологій, які пропонують користувачам персоналізовану та релевантну взаємодію. Це закладає основу для наступних розділів дипломної роботи, в яких ми заглибимося в методи створення і вдосконалення рекомендаційних систем.

РОЗДІЛ 2 МЕТОДИ І МОДЕЛІ ДЛЯ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

2.1 Традиційні стратегії формування персоналізованих рекомендацій

Як вже описано у попередніх розділах, традиційними методами для побудови рекомендаційних систем є методи на основі контентної та колаборативної фільтрацій. У цьому підрозділі буде більш детально розглянуто кожен із них.

Система контентної фільтрації відбирає елементи на основі співвідношення між вмістом елементів та уподобаннями користувача, на відміну від системи колаборативної фільтрації, яка відбирає елементи на основі кореляції між користувачами зі схожими вподобаннями. Опишемо проблему більш формально.

Нехай у нас є U – це множина користувачів, $U = \{u_1, u_2, \dots, u_m\}$, а $I = \{i_1, i_2, \dots, i_n\}$ – це множина об'єктів (товарів, новин, тощо). Кожен об'єкт $i_j \in I$ описується вектором ознак $x_j \in \mathbb{R}^n$, що відображає його вміст (жанр, ключові слова, тощо). Для кожного користувача формується $u_i \in U$ формується його профіль $p_i \in \mathbb{R}^m$, що агрегує вектори ознак об'єктів із якими користувач взаємодіяв. Завдання полягає в тому. щоб для кожного користувача u_i знайти такі сутності i_j для яких очікуване значення сумісності $sim(p_i, x_j)$ є максимальним. У ролі функцій сумісності можуть бути використані різноманітні функції такі як косинусна подібність, кореляція пірсона та сумісність Жаккара (Jaccard). Далі буде розглянуто більш детально кожну із цих функції сумісності.

Косинусна подібність – цей метод використовує косинус кута між двома векторами з початковою точкою в нулях осей координат (формула 2.1).

$$sim_{cos}(p, x) = \frac{p \cdot x}{||p|| ||x||} = \frac{\sum_{i=1}^{n} p_i * x_i}{\sqrt{\sum_{i=1}^{n} {p_i}^2} \sqrt{\sum_{i=1}^{n} {x_i}^2}}. \quad (2.1)$$

Косинусна подібність має низку переваг. Зокрема, вона є незалежною від масштабу векторів, що дає змогу коректно порівнювати тексти різної довжини без спотворення результатів. Крім того, функція виявляє семантичну схожість шляхом оцінки напрямку векторів, що є особливо важливим при роботі з векторними представленнями тексту, такими як Glove, TF-IDF, Word2Vec. Водночас косинусна подібність має і певні обмеження: вона не враховує абсолютну вагу окремих ознак, що може призводити до завищення подібності між векторами із розрідженим, але співнапрямленими набором ознак. Також ця метрика менш ефективна при обробці категоріальних ознак.

Кореляція Пірсона – це функція, яка вимірює схожість або кореляцію між двома об'єктами даних, порівнюючи їхні атрибути та обчислюючи оцінку в діапазоні від -1 до +1. Високий бал вказує на високу схожість, тоді як бал, близький до нуля, вказує на відсутність кореляції (формула 2.2).

$$sim_{pearson}(p,x) = \frac{\sum_{i=1}^{n}(p_i - \bar{p})(x_i - \bar{x})}{\sum_{i=1}^{n}\sqrt{(p_i - \bar{p})^2}\sum_{i=1}^{n}\sqrt{(x_i - \bar{x})^2}}. \quad (2.2)$$

Основна перевага цієї функції здатність враховувати різні масштаби оцінювання, тобто вона згладжує індивідуальні упередження користувачів у виборі об’єктів. Недоліком є те, що кореляція Пірсона чутлива до шуму при малій кількості спільних переглядів, і вона неефективна у випадку нелінійної залежності або сильно розріджених даних, що поширено у рекомендаційних системах.

Сумісність Жаккара (Jaccard) – це метрика, що вимірює схожість між двома множинами як відношення кількості спільних елементів до кількості унікальних елементів в об’єднанні (формула 2.3).

$$sim_{jacard}(p,x) = \frac{|p \cap x||}{|p \cup x|} = \frac{\sum_{i=1}^{n} p_i * x_i}{\sum_{i=1}^{n} \max(p_i, x_i)}. \quad (2.3)$$

Ця функція особливо ефективна для роботи з бінарними або категоріальними даними, дає змогу ефективно порівнювати множини незалежно від їхнього розміру. Недоліком є те, що вона не враховує силу взаємодії, тому всі елементи вважаються рівнозначними, також може бути малоефективною при порівнянні дуже малих або дуже великих множин.

Наступним типом фільтрації є колаборативна фільтрація, вона вже у свою чергу не використовує ознаки контенту напряму для прогнозування рекомендацій, а основується на факті того. шо користувачам із подібними історіями взаємодій подобається схожий контент. Якщо розглядати проблему більш формально, то можна описати наступне.

Знову ж таки нам дано множину користувачів $U = \{u_1, u_2, \ldots, u_m\}$ та множину об'єктів $I = \{i_1, i_2, \ldots, i_n\}$. І також розріджена матриця взаємодій користувачів із сутностями $R \in \mathbb{R}^{N \times M}$, де R – це розріджена матриця, що складається із r: $r_{u,i} \in \mathbb{R}$ – це відома взаємодія (1 – якщо взаємодіяв із сутністю, 0 – в протилежному випадку) або оцінка користувача *u* для елемента *i*. Тепер вже відштвохуючись від цього, ціллю цієї задачі є знаходження такої функції прогнозування $f(u,i)$, яка давала б оцінити невідомі взаємодії користувача *u*, тобто для невідомих елементів *i*, знайти: $\hat{r}_{u,i} = f(u,i)$. Для знаходження цієї функції прогнозування є два загальних підходи.

Перший із них, це підхід який заснований на пам'яті (memory-based approach). Тут у свою чергу є два інших підходи, заснований користувачах (user-based), та на попередній історії уподобань користувача (item-based).

Перш за все розглянемо підхід орієнтований на користувача. Це підхід прогнозує рекомендації на основі уподобань користувача, які схожі на вподобання інших користувачів. Наприклад, якщо користувач оцінює контент

так само, як і користувач, для якого необхідно зробити прогноз, то можна припустити, що вони мають схожі інтереси.

1. Перш за все потрібно обрати користувачів, які максимально схожі на користувача, для якого потрібен прогноз, це можна зробити на основі функцій сумісності, наприклад тих, які були описані для систем контеної фільтрації, та історій уподобань користувачів. Після цього формуємо $N_k(u) \subset U/\{u\}$ – множина k користувачів, які найбільш схожі до поточного користувача на основі $sim(u, v)$ – обрана функція сумісності.
2. Тепер потрібно обрахувати прогнозовану оцінку для поточного користувача та сутності i, яку користувач ще не бачив (формула 2.4).

$$\hat{r}_{u,i} = \overline{r_u} + \frac{\sum_{v \in N_k(u)} sim(u, v)(r_{v,i} - \overline{r_v})}{\sum_{v \in N_k(u)} |sim(u, v)|}, \quad (2.4)$$

де $\overline{r_u}$, $\overline{r_v}$ – середні оцінки користувачів,

$r_{v,i}$ – оцінка користувача v для сутності i.

3. Після обрахування прогнозованих оцінок $\hat{r}_{u,i}$ для $i \in I/\{u\}$ обираються N найкращих сутностей на основі прогнозованих оцінок.

Зазначений підхід є відносно простим у реалізації, однак має два суттєвих недоліки, які унеможливлюють його ефективне застосування в межах нашої задачі. По-перше, він не є масштабованим, обчислення схожості між усіма парами користувачів є вкрай ресурсомістким при їх великій кількості. По-друге, висока розрідженість даних ускладнює процес виявлення подібних користувачів, що значно знижує якість результатів [13].

Алгоритм для контенто-орієнтованого підходу, є достатньо схожим до попередньо описаного. Перш за все потрібно обрахувати для кожного елемента $i \in I$, потрібно обрахувати функцію сумісності із іншими елементами $j \in I/\{i\}$. Для обрахування функції сумісності потрібно використати множину, таких користувачів, що взаємоділяи із цими двома об’єктами, $U_{ij} = \{u \in$

$U|\, r_{u,i}, r_{u,j}$ існують}. Потім обрати k найбільш схожих елементів до i, які користувач вже оцінив – $I_k(u)$. І останній крок розрахувати прогнозовану оцінку (формула 2.5).

$$\hat{r}_{u,i} = \frac{\sum_{j \in I_k(u)} sim(i,j)\, r_{u,j}}{\sum_{j \in I_k(u)} |sim(i,j)|}. \tag{2.5}$$

Таким чином, обидва підходи ґрунтуються виключно на матриці взаємодій, що дає змогу уникнути процесу навчання моделей. Проте ці методи залишаються вразливими до низки важливих обмежень: вони чутливі до розрідженості даних, неефективні в умовах холодного старту і не масштабуються належним чином із зростанням обсягу даних.

Наступним методом прогнозування у межах колаборативної фільтрації є модельно-орієнтований підхід (model-based approach). Основними традиційними напрямками у цьому підході є використання машин факторизації (FM) та кластеризаційних методів, наприклад метод k найближчих сусідів (KNN).

Factorization Machines (FM) спрямовані на вивчення низьковимірних представлень (ембедингів) користувачів та об’єктів у спільному латентному факторному просторі. З технічної точки зору, спостережувану розріджену матрицю взаємодій між користувачами та об’єктами апроксимують як добуток двох матриць низького рангу, що містять відповідні ембединґи. Після навчання латентних факторів стає можливим прогнозування ймовірності взаємодії користувача з новими, раніше невідомими об’єктами шляхом обчислення схожості між відповідними представленнями у латентному просторі. Графічне представлення алгоритму можна побачити на рисунку 2.1

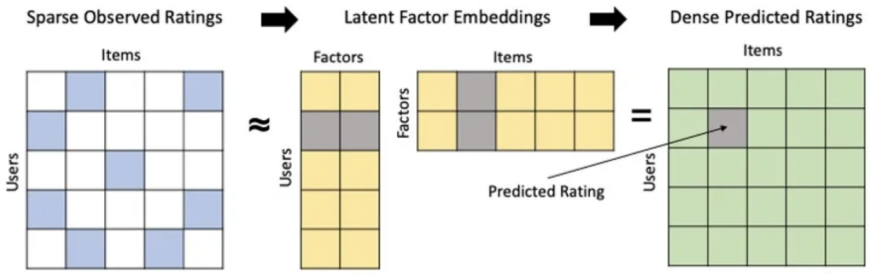

Рисунок 2.1 – Графічне представлення алгоритму FM [14]

Рівняння моделі FM (формула 2.6) описує взаємодії між ознаками, включаючи до *n*-сторонні залежності. Найбільш поширеною є модель другого порядку, яка містить лінійні ваги для кожної окремої ознаки, а також параметри для парних взаємодій між ознаками. Таким чином, модель здатна ефективно моделювати складні залежності навіть у розріджених даних, завдяки факторизації взаємодій між парами ознак у вигляді скалярного добутку їх латентних представлень [15].

$$f(x) = w_o + \sum_{p=1}^{P} w_p x_p + \sum_{p=1}^{P-1} \sum_{q=p+1}^{P} \langle v_p, v_q \rangle \, x_p x_q. \qquad (2.6)$$

У разі наявності явного фідбеку у якості функції втрат для моделі доцільно використовувати стандартні метрики, зокрема середньоквадратичну помилку (RMSE) або логістичну (log-loss). Але у випадку наявності непрямого фідбеку, ці функції не будуть ефективними. По-перше через класовий дисбаланс: більшість пар «користувач–об’єкт» мають нульове значення. По-друге в умовах непрямого фідбеку справжньою метою є побудова ефективного ранжування об’єктів відповідно до їхньої ймовірної релевантності для користувача. У цьому контексті мінімізація MSE або log-loss не гарантує високої ймовірності для нещодавно переглянутих об’єктів.

Для враховування вище перерахованих проблем, як правило використовуються ранжувальні втрати (learning-to-rank):

1. Байєсівське персоналізоване ранжування (BPR) (формула 2.7) – формуються триплети (*u, i, j*), де *i* – сутність, з якою користувач u взаємодіяв, а *j* – непереглянута сутність. Мета цієї функції втрат в тому, щоб максимізувати апостеріорну ймовірність уподобання *i* над *j* [15]

$$max_\theta \sum_{(u,i,j)\in S} ln(\sigma(\, f(u,i|\theta) - f(u,j|\theta)\,)). \tag{2.7}$$

2. Зважений наближений парний ранг (WARP) – знову ж таки формуються триплети, але в цьому випадку вже краще обирається негативний приклад. Для кожного позитивного i послідовно обираються випадкові j, доки не знайде таку сутність j, для якої (формула 2.8) [15]

$$f(u,j) > f(u,i) - margin. \tag{2.8}$$

Чим більше спроб знадобилося до виявлення «помилки ранжування», тим менший ваговий коефіцієнт для оновлення, адже модель уже непогано ранжує більшість пар.

У якості моделей колаборативної фільтрації кластеризаційні методи застосовуються для групування користувачів на основі подібності у їхніх вподобаннях. Найбільш поширеними алгоритмами в цьому контексті є методи k-середніх, гауссової суміші (GMM), ієрархічна кластеризація та спектральна кластеризація (spectral clustering). На відміну від класичної постановки задачі для кластеризації, де метою є мінімізація внутрішньо кластерної відстані. У випадку колаборативної фільтрації цільова функція змінюється так, щоб мінімізувати похибку передбачених оцінок у межах кожного кластера.

$$\sum_{i=1}^{K} \sum_{(u,j)\in D_i} (r_{uj} - \hat{r}_{uj}^{(i)})^2. \quad (2.9)$$

У підрозділі 2.1 проаналізовано традиційні підходи до формування персоналізованих рекомендацій, зокрема контентну та колаборативну фільтрацію. Контентна фільтрація дає змогу генерувати рекомендації на основі атрибутів об’єктів, проте демонструє недостатню ефективність у випадках обмеженої інформації про користувача. Натомість колаборативна фільтрація ґрунтується на подібності між користувачами та об’єктами, але є чутливою до проблеми розрідженості матриці взаємодій і погано масштабується на великих обсягах даних. У межах колаборативної фільтрації розглянуто як евристичні (memory-based), так і модельно-орієнтовані (model-based) підходи, зокрема модель Factorization Machines, яка забезпечує ефективне моделювання взаємодій між користувачами й об’єктами. Крім того, проаналізовано функції сумісності, що використовуються для побудови прогнозів, з урахуванням їхніх переваг і недоліків залежно від характеру вхідних даних.

2.2 Персоналізовані рекомендації на основі глибоких нейронних мереж

Глибоке навчання зараз викликає великий ентузіазм. За останні кілька десятиліть глибоке навчання (DL) досягло значних успіхів у різних сферах застосування, включаючи розпізнавання мови і комп'ютерний зір. Оскільки глибоке навчання може вирішувати багато комплексних завдань і давати кращі результати за традиційні методи, науковці і підприємці наввипередки намагаються застосувати його до широкого спектру проблем [5]

Згідно із опитуванням ACM Computing Surveys, респонденти виділяють такі переваги використання DL [16].

1. Нелінійні перетворення. На відміну від лінійних моделей, глибокі нейронні мережі здатні моделювати нелінійність даних з нелінійними активаційними функціями, такими як Relu, sigmoid, Tanh тощо. Ця властивість дає змогу вловлювати складні та заплутані патерни взаємодії елементів користувача.
2. Репрезентативне навчання. Глибокі нейронні мережі ефективні у вивченні основних дескриптивних факторів та корисних репрезентацій із вхідних даних. Загалом, велика кількість описової інформації про об'єкти і користувачів доступна в додатках. Використання цієї інформації забезпечує спосіб покращити наше розуміння об'єктів і користувачів, що в результаті призведе до створення рекомендацій.
3. Моделювання послідовностей. Глибокі нейронні мережі показали багатообіцяючі результати в ряді задач послідовного моделювання, таких як машинний переклад, розпізнавання мови, чат-боти та багато інших. RNN та CNN відіграють важливу роль у цих завданнях.
4. Гнучкість. Методи глибокого навчання мають високу гнучкість, особливо з появою багатьох популярних фреймворків глибокого навчання, таких як Tensorflow, Keras, Caffe, MXnet, DeepLearning4j, PyTorch, Theano тощо. Більшість з цих інструментів розробляються за модульним принципом і мають активну спільноту.

Для подальшого порівняльного аналізу та вибору найоптимальнішої моделі глибокого навчання для розробки фінального програмного продукту обрано три сучасні архітектури – DKN [17], NRMS [18] та NAML [19].

Вибір саме цих моделей зумовлений їхніми гібридними архітектурами, наявністю відкритих реалізацій та здатністю ефективно розв’язувати проблеми рекомендацій новин. До проблем рекомендацій новин можна віднести: короткотривалість життя новинного контенту, динамічність інтересів користувачів, обмеженість поведінкових даних, семантична неоднозначність новинного контенту

DKN є однією з перших архітектур, що інтегрує зовнішні знання у формі графа знань для підвищення ефективності персоналізованої рекомендації новин. Основна ідея моделі полягає у подоланні семантичної неоднозначності новинного контенту, зокрема коротких заголовків, шляхом залучення додаткового контексту через пов’язані сутності зі знаннєвих баз.

Енкодер новин і енкодер користувача – це дві основні частини архітектури DKN. Для того, щоб розширити контекст сприйняття новин, контент новин представляється не тільки за допомогою тексту заголовка, але й набором пов'язаних з ним сутностей, визначених за допомогою техніки зв'язування сутностей (entity linking).

Для аналізу тексту заголовка з послідовності токенів виділяються локальні контекстні ознаки за допомогою згорткової нейронної мережі (CNN). При цьому об'єкти отримують власну матрицю ембедингів, а механізм уваги використовується для агрегування їхнього впливу на представлення новин. Особливою перевагою цієї моделі є використання matching layers, які обчислюють взаємозв’язок між текстовими ознаками та контекстом із графу знань. Це дає змогу динамічно адаптувати вагу знаннєвої інформації залежно від важливості кожної сутності в контексті тексту.

Після формування векторного представлення новини та користувача ці два вектори конкатенуються та подаються у фінальну нейромережу системи, яка вже на основі конкатенованих векторі прогнозує ймовірність натиску на новину.

Модель DKN має такі переваги у контексті персоналізованої рекомендації новин. Завдяки інтеграції зовнішніх знань із графа сутностей вона здатна ефективно обробляти короткі й семантично неоднозначні заголовки. Поєднання згорткових нейронних мереж для отримання локальних ознак із механізмом уваги забезпечує релевантне представлення як новин, так і профілю користувача. Водночас модель має певні обмеження: вона чутлива до якості попереднього зв’язування сутностей та характеризується високою обчислювальною складністю через необхідність обробки знаннєвих графів.

NRMS [18] – модель, яка була запропонованан для подолання пов'язаних із короткою довжиною текстів заголовків новин та динамічною зміною інтересів користувачів. NRMS фокусується на повноцінній векторизації, як новинного контенту, так і профілю користувача за допомогою механізмів самоуваги.

Основною відмінність цієї моделі від інших, це механізм самоуваги, який для створення представлення новини, використовує моделювання зв'язків між словами. Наприклад у новинному загаловку "Rockets Ends 2018 with a Win", зв'язок між словами "Rockets" та "Win" корисно для розуміння контексту цієї новини. У вищенаведеному прикладі, слово "Rockets" взагалі має зв'язок із двома словами "Ends" та "Win". Тому у механізмі уваги і використано відразу алгоритм багатоголової самоуваги для вивчення контекстуальної репрезентації слів шляхом вивчення взаємодії між словами. Моделювання репрезентація i-го слова k-ою головою уваги, представлення у формулах (2.10) та (2.11) [18].

$$\alpha_{i,j}{}^{k} = \frac{exp(e_i{}^{T} Q_k{}^{w} e_j)}{\sum_{m-1}^{M} exp(e_i{}^{T} Q_k{}^{w} e_m)}, \tag{2.10}$$

$$h_{i,k}{}^{w} = V_k{}^{w} \left(\sum_{j=1}^{M} \alpha_{i,j}{}^{k} e_j \right). \tag{2.11}$$

Репрезентацію i-го слова можна побачити у формулі (2.11) – це конкатенація репрезентацій, вироблених h окремими головами механізму самоуваги, тобто $h_i{}^{w} = [h_{i,1}{}^{w}, h_{i,2}{}^{w}, \ldots, h_{i,h}{}^{w}]$; $V_k{}^{w}$ та $Q_k{}^{w}$– проекції параметрів у $k-$ у голову самоуваги та e_j – це ембединг j-го слова заголовку новини. У формулі (2.10) описано обрахування відносної важливості взаємодії між i-им та j-им словами

Наступним шаром механізму самоуваги є виділення найважливіших слів у кожному із новинних заголовків, розрахунок можна побачити у формулах (2.12), (2.13).

$$\alpha_i{}^w = q_w{}^T tanh(V_w \times h_i{}^w + v_w), \tag{2.12}$$

$$\alpha_i{}^w = \frac{exp(\alpha_i{}^w)}{\sum_{j=1}^{M} exp(a_j{}^w)}. \tag{2.13}$$

Фінальна вага кожного слова у новинному заголовку обраховується вже за допомогою механізму описаного у формулі (2.13), а обрахування ваги кожного слова розглянуто у формулі (2.12). У формулі (2.12) V_w, v_w – проекція параметрів, а q_w – вектор запиту. І вже фінальна репрезентація новини – це зважена сума контекстних представлень слів, що зазначено у формулі (2.14):

$$r = \sum_{i=1}^{M} \alpha_i{}^w h_i{}^w. \tag{2.14}$$

На рисунку 2.2, перші дві новини пов'язані між собою. Крім того, новина може бути пов’язаною з іншими новинами, які переглядає той самий користувач. Тому для створення репрезентації користувача, можна використати такий же алгоритм, як і для новин, де перший етап – це мережа багатоголової самоуваги, другий етап – це мережа адитивної уваги.

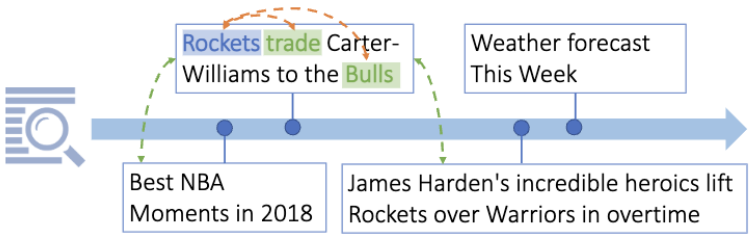

Рисунок 2.2 – Історія взаємодій користувача [18]

Після того, як отримано репрезентації користувача та новини, формується фінальна ймовірність натиску на новину за допомогою векторного добутку (формула 2.15), де u^T – репрезентація користувача, r^c – репрезентація новини.

$$\hat{y} = u^T r^c. \tag{2.15}$$

Для кращого розуміння архітектури моделі, всі попередньо описані етапи зображені на рисунку 2.3.

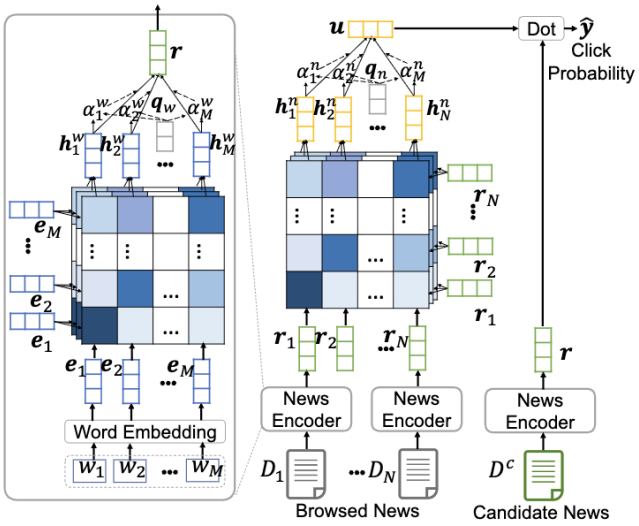

Рисунок 2.3 – Архітектура моделі NRMS [18]

Як результат, модель NRMS успішно балансує між обчислювальною складністю та представленням інтересів користувачів. Її основою є багатоголовий механізм самоуваги, який пропонує детальне моделювання семантичних зв'язків між новинами, з якими взаємодіє користувач, а також всередині заголовків новин. Це дає змогу динамічно підлаштовувати представлення профілю користувача і контенту. Використання достатньо спрощеного процесу прогнозування – скалярного добутку векторів користувача

та новин, а не багаторівневої нейромережі – є суттєвою відмінністю від моделі DKN.

NAML була запропонована з метою подолання обмежень контентного представлення новин, зокрема у випадках, коли використовується лише заголовок. Її основна концепція базується на багатовимірному (multi-view) підході, який інтегрує кілька джерел семантичної інформації – заголовок, короткий опис, категорію та підкатегорію новини – для побудови більш повного та точного векторного представлення. Такий підхід дає змогу ефективніше моделювати як зміст новинного контенту, так і інтереси користувача, особливо у випадках коротких або семантично неоднозначних заголовків.

Загальна архітектура моделі ідентична до двох представлених перед цим, енкодер новин, енкодер юзера та фінальний етап по формуванню прогнозу. Перш за все це формування репрезентації новини, для загловку та основної частини тексту використовується схожий механізм формування фінальної репрезентації.

Перш за все для кожного слова формується відповідний йому ембединг за допомогою вже натренованих ембедингів, наприклад Glove, TF-IDf та інші. Наступний шар – це згорткова нейронна мережа, яка використовується для вивчення контекстуальних репрезентації слів за допомогою виділення їхніх локальних контекстів. Позначимо контекстуальну репрезентацію i-го слова за допомогою, ${c_i}^t$, розрахунок якого позначено у формулі (2.16):

$$ {c_i}^t = ReLU\left(F_t \times {e^t}_{(i-K):(i+K)} + b_t\right). \tag{2.16} $$

У формулі (2.16) використано такі позначення. ${e^t}_{(i-K):(i+K)}$ – це конкатенація ембедингів слів із $(i-K)$ до $(i+K)$ позиції; $F^t \in R^{N_f \times (2k+1)D}$ та $b_t \in R^{N_f}$ – ядро та зсув фільтрів у згортковій нейронній мережі; N_f – кількість фільтрів у згортковій нейронній мережі та $2K+1$ – це розмір вікна. ReLU – нелійнійна функція активації. Третім шаром є мережа уваги для визначення найбільш важливих слів у контексті тексту, цей етап вже описано для NRMS у

формулах (2.12) та (2.13). Фінальна репрезентація слова – це вже комбінація отриманих представлень слів після згорткової нейронної мережі та їх відповідних ваг уваги (2.17):

$$r^t = \sum_{j=1}^{M} {\alpha_j}^t {c_j}^t. \tag{2.17}$$

Вже після цього також відбувається кодування відповідних полів про категорію та саб-категорію новини. Перша за все кожну категорію ми перетворюємо у векторний вигляд за допомогою Label Encoding, потім після цього додається один щільний шар для того, щоб навчатися кореткно використовувати категорію новини та її саб-категорію під час тренування моделі, відповідно, формули (2.18), (2.19):

$$r^c = ReLU(V_c \times e^c + v_c), \tag{2.18}$$

$$r^{sc} = ReLU(V_s \times e^{sc} + v_{sc}). \tag{2.19}$$

Останній етап репрезентації новини, це вже згортка із механізмом уваги (attentive pooling), враховуючи всі попередні отримані векторні представлення заголовку, короткого опису тексту, для якого механізм перетворення аналогічний до заголовку, категорії та саб-категорій. Перша за все обраховується вага кожного параметру, формули (2.20), (2.21):

$$\alpha_t = {q_v}^T tanh(U_v \times r_t + u_v), \tag{2.20}$$

$$\alpha_t = \frac{exp(\alpha_t)}{exp(\alpha^c) + exp(\alpha^{sc}) + exp(\alpha_t) + exp(\alpha_b)}. \tag{2.21}$$

Фінальна репрезентація новини вже основується на порахованих вагах для кожного компоненту (заголовку, короткого опису тексту, категорії та саб-категорії), формула (2.22)

$$r = \alpha_c r^c + \alpha_{sc} r^{sc} + \alpha_t r^t + \alpha_b r^b. \quad (2.22)$$

Створення репрезентації користувача основується на основі репрезентації кожної новини із його історії, для цього використовується попередньо розглянутий механізм, для виділення основних новин використовується механізм уваги аналогічний до того, що використовувався для виділення основних характеристик новин для складення її репрезентації.

Отже, переваги NAML – це здатність об'єднувати кілька джерел інформації про новину, що дає змогу значно покращити розуміння її змісту. Крім того, багаторівневий механізм уваги забезпечує більш адаптивне формування профілю користувача. Важливою перевагою також є підхід, який забезпечує гнучку інтеграцію різнорідної семантичної інформації (заголовків, категорій, підкатегорій та коротких описів текстів), що підвищує точність представлення як новин, так і користувача. З іншого боку, модель вимагає більшого обсягу анотацій для новин і є обчислювально складнішою за простіші архітектури, як-от NRMS.

На основі розглянутих моделей у розділі 2.2 складено порівняльну таблицю (табл. 2.1), яка базується на архітектурних особливостях: рівні семантичного розуміння, використанні зовнішніх знань, обчислювальній складності та гнучкості використання моделі.

Інтеграція зовнішніх знань – ключова особливість DKN, що відрізняє її від інших моделей, поєднуючи текстові ознаки з інформацією із графів знань, модель здатна глибше інтерпретувати зміст новин, а використання KCNN дає змогу ефективно виявляти важливі локальні контексти. Завдяки цьому DKN ефективна навіть у випадках коротких заголовків.

Таблиця 2.1 – Порівняльна таблиця моделей згаданих у розділі 2.2

Модель	*Механізм уваги*	*Зовнішні знання*	*Обчисл. складність*	*Гнучкість*	*Семантичне розуміння*
DKN	Так	Так	Помірна	Висока	Високе
NRMS	Так	Ні	Низька	Висока	Високе
NAML	Так	Ні	Висока	Середня	Високе

На відміну від DKN, модель NRMS фокусується лише на внутрішніх текстових зв’язках, а NAML потребує повної анотації новин (заголовок, опис, категорія, підкатегорія), що ускладнює масштабування в реальних умовах. При цьому DKN вирізняється нижчою обчислювальною складністю порівняно з багаторівневими підходами, які реалізовані в NAML та NRMS.

Завдяки вдалому балансу між точністю, гнучкістю, семантичним розумінням і ефективністю, DKN є доцільним вибором для побудови персоналізованої системи рекомендацій новин.

2.3 Метрики точності в рекомендаційних системах

Для об'єктивної оцінки якості побудованої рекомендаційної системи важливо застосовувати відповідні метрики, які відображають здатність моделі правильно формувати прогноз рекомендацій.

Залежно від підходу до побудови моделі, метрики для її оцінювання можна умовно поділити на класифікаційні та ранжувальні. Класифікаційні метрики використовуються переважно у випадках, коли завдання моделі полягає у бінарному передбаченні (наприклад, натисне користувач на елемент чи ні), тоді як ранжувальні використовуються для випадків коли модель як результат надає

впорядкований список. Далі розглянемо 3 класифікаційних: ROC AUC, Precision@K, Recall@K та 1 ранжувальну: NDCG@K.

ROC AUC – це класифікаційна метрика, яка оцінює здатність моделі відрізняти позитивні приклади від негативних. ROC-крива показує залежність між частотою істинно позитивних спрацювань (True Positive Rate, TPR) та хибно позитивних (False Positive Rate, FPR) при зміні порогу класифікації. AUC (Area Under Curve) – це площа під цією кривою, чим вона ближча до 1, тим краща здатність моделі до класифікації (формула 2.23)

$$AUC = \frac{1}{|P|\,|N|} \sum_{(x_p, x_n) \in P \times N} I\left(\hat{y}_{x_p} > \hat{y}_{x_n}\right). \tag{2.23}$$

У формулі (2.23) використано такі позначення: P – множина позитивних прикладів, N – множина негативних прикладів $\hat{y}_x$ – передбачене значення моделі для об'єкта x, I – індикаторна функція.

Перевагою ROC AUC є її незалежність від конкретного порогу класифікації. Це особливо важливо у випадках із класовим дисбалансом, коли інші показники, як-от accuracy, можуть бути оманливими. Водночас, ROC AUC не враховує порядок елементів у списку рекомендацій, що обмежує її застосування для оцінювання моделей, орієнтованих на ранжування.

Precision@K є ранжувальною метрикою, що використовується для оцінювання точності рекомендаційної системи у межах перших K позицій рекомендованого списку. Вона вимірює частку релевантних об’єктів серед перших K рекомендацій (2.24).

$$Precision@K = \frac{r}{K}, \tag{2.24}$$

де r – кількість релевантних елементів серед перших K рекомендованих.

Основною перевагою метрики Precision@K є її зосередженість на точності верхньої частини списку. Дана метрика застосовується для вирішення проблем, у яких важливо показати користувачеві лише найбільш релевантні варіанти.

Recall@K оцінює повноту рекомендацій, тобто яку частку від усіх релевантних об’єктів модель змогла запропонувати у межах перших K позицій рекомендованого списку. Вона показує, наскільки добре система охоплює всі релевантні для користувача елементи (формула 2.25).

$$Recall@K = \frac{r}{R}. \tag{2.25}$$

У формулі (2.25), використано такі позначення: r – кількість релевантних елементів; R – кількість усіх можливих релевантних елементів. Обидва значення обраховуються серед перших K елементів впорядкованого списку. Здатність оцінювати повноту охоплення релевантного контенту системою є перевагою Recall@K метрики, але при цьому вона не враховує точність рекомендацій. Тому ця метрика повинна застосовуватися тільки у парі із Precision@K для більш зрозумілої картини стосовно ефективності моделі.

NDGC@K – це ранжувальна метрика, яка враховує як релевантність рекомендованих об’єктів, так і їхню позицію у списку. Вона базується на концепції, що релевантні елементи, розташовані ближче до початку списку, мають більшу цінність для користувача, ніж ті, що знаходяться на нижчих позиціях. Таким чином, NDCG@K оцінює якість ранжування рекомендацій з урахуванням їх корисності і порядку. Обрахування метрики здійснюється у два етапи, перш за все обраховується DCG@K (2.26).

$$DCG@K = \sum_{i=1}^{K} \frac{rel_i}{log_2(i+1)}, \tag{2.26}$$

де rel_i – релевантність елемента на позиції i у списку,
K – розмір оцінюваної вибірки.

DCG зменшується логарифмічно з ростом позиції, що моделює зниження уваги користувача до елементів, розміщених далі. Для нормалізації DCG використовується ідеальний DCG (IDCG@K) – максимальне можливе значення DCG при ідеальному впорядкуванні релевантних елементів (формула 2.27):

$$IDCG@K = \sum_{i=1}^{|REL|} \frac{rel_i^*}{log_2(i+1)}. \quad (2.27)$$

Вже нормалізований показник обраховується за формулою (2.28):

$$NDCG@K = \frac{DCG@K}{IDCG@K}. \quad (2.28)$$

Значення NDCG@K знаходиться в діапазоні від 0 до 1, де 1 відповідає ідеальному ранжуванню. Перевага метрики полягає у здатності відображати не лише наявність релевантних рекомендацій, а й їхнє розташування у списку, особливо потрібно для оцінки якості рекомендацій із ранжувальною проблемою. Водночас NDCG@K потребує визначення ступеня релевантності для кожного елемента, що іноді може бути складно в задачах із бінарною релевантністю.

Отже, класифікаційні метрики, ефективні для задач бінарного передбачення, але не враховують порядок рекомендацій. У випадку задачі прогнозування кліків (CTR), метрика ROC AUC є найбільш релевантною, оскільки вона оцінює здатність моделі розрізняти між класами незалежно від порогу класифікації. Це особливо важливо при дисбалансі класів, характерному для CTR, де подія кліку відбувається набагато рідше за відсутність кліку. Також у нашому випадку клас ранжувальних метрик не потрібен оскільки це вже

надмірно, оскільки нам не потрібно враховувати порядок розташування елементів при прогнозі, нам цікавий лише факт натиску чи не натиску на новину.

Висновки до розділу 2

У другому розділі розглянуто методи побудови та оцінки рекомендаційних систем. Перш за все розглянуто традиційні стратегії: контентну та колаборативну фільтрацію. Основна увага була приділена сучасним моделям на основі глибокого навчання, серед яких виділяються DKN, NRMS та NAML. Зокрема, DKN інтегрує дані із графів знань, NRMS використовує механізм самоуваги для аналізу контексту, а NAML поєднує відразу декілька характеристик тексту, такі як заголовки, короткий опис тексту, категорію та саб-категорію новини. Проведений порівняльний аналіз дозволив виявити переваги та обмеження кожної архітектури й обґрунтувати доцільність вибору моделі DKN для реалізації системи. У кінці розділу розглянуто метрики оцінки якості рекомендацій: класифікаційні та ранжувальні.

РОЗДІЛ 3 РОЗРОБКА РЕКОМЕНДАЦІЙНИХ СИСТЕМ ТА ОЦІНКА ЕФЕКТИВНОСТІ

3.1 Попередній аналіз даних

Попередній аналіз даних (EDA) є важливим етапом аналізу будь-якого дослідження. Основна мета розвідувального аналізу – дослідити дані на предмет розподілу, викидів та аномалій, щоб спрямувати конкретну перевірку вашої гіпотези. Він також надає інструменти для генерування гіпотез шляхом візуалізації та розуміння даних, зазвичай за допомогою графічного представлення [20].

Із рисунку 3.1 можна зрозуміти, що для тренування буде використаний часовий проміжок: 9 листопада – 14 листопада 2019 року, для тестування – наступний тиждень, тобто 16 листопада – 22 листопада 2019 року. Для валідації буде використаний вже лише 15 листопада

```
Train DF:
Min Time: 2019-11-09 00:00:00
Max Time: 2019-11-14 23:59:59

Test DF:
Min Time: 2019-11-16 00:00:05
Max Time: 2019-11-22 23:59:58

Val DF:
Min Time Val DF: 2019-11-15 00:00:00
Max Time Val DF: 2019-11-15 23:59:43
```

Рисунок 3.1 – Часові періоди для датасетів

Для навчання моделей потрібні дані про те, як користувачі взаємодіють з попередніми новинами. Важливо оцінити кількість користувачів, які не мають такої історії. Рисунок 3.2 показує, що близько 2% користувачів не мають історії

взаємодії. Така кількість відсутніх даних не є критичною для побудови моделі, особливо при використанні машини факторизації, оскільки для цього типу моделей такі дані не можна буде використати для тренування. Також, для моделі DKN (Deep Knowledge Network) відсутність історії також не є суттєвою проблемою, завдяки її архітектурі, яка дає змогу їй враховувати також і порожню історію для тренування

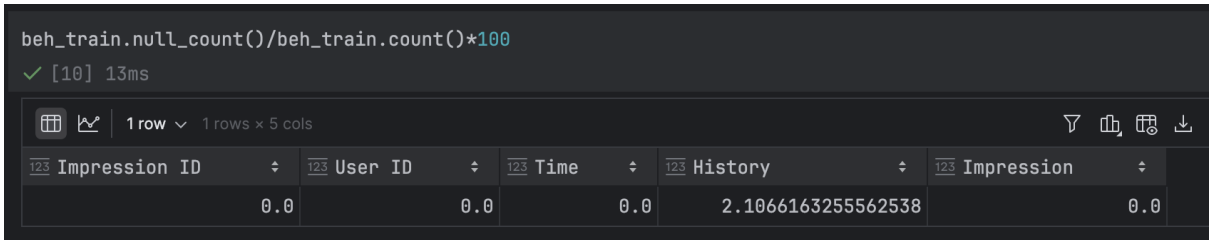

Рисунок 3.2 – Кількість нульових значень для всіх атрибутів

Атрибут цих даних Impression буде використаний для безпосереднього навчання, оскільки він відповідає за те, чи нажав користувач нажав на відповідну запропоновану новину, коли він її побачив під час прочитання останньої новини із History. Варто перевірити наскільки багато хибних рекомендацій є в початковому датасеті. Як можна побачити із рисунку 3.3, то в середньому кожен користувач натискав на запропоновану новину в 1 із 10 випадків, що говорить про розрідженість цих даних

```
Median True Values: 5.63%
Mean True Values: 10.81%
```

Рисунок 3.3 – Розрідженість даних для атрибуту Impression

Тепер коли ми перевірили дані, які стосуються користувачів, можна перейти до новин. Перше, що необхідно перевірити це кількість новин за кожною із категорій. Із рисунку 3.4 можна побачити, що дві найбільш популярні категорії

– це спорт та новини (включає новини про економічну ситуацію у світі, політику, і т.д.), які собою сумарно займають близько 50% всіх новин у датасеті. Це свідчить про наявність значного дисбалансу у категоріях новин, що може вплинути на навчання моделей. Для побудови ефективних рекомендаційних систем FFM та DKN важливо врахувати цей дисбаланс

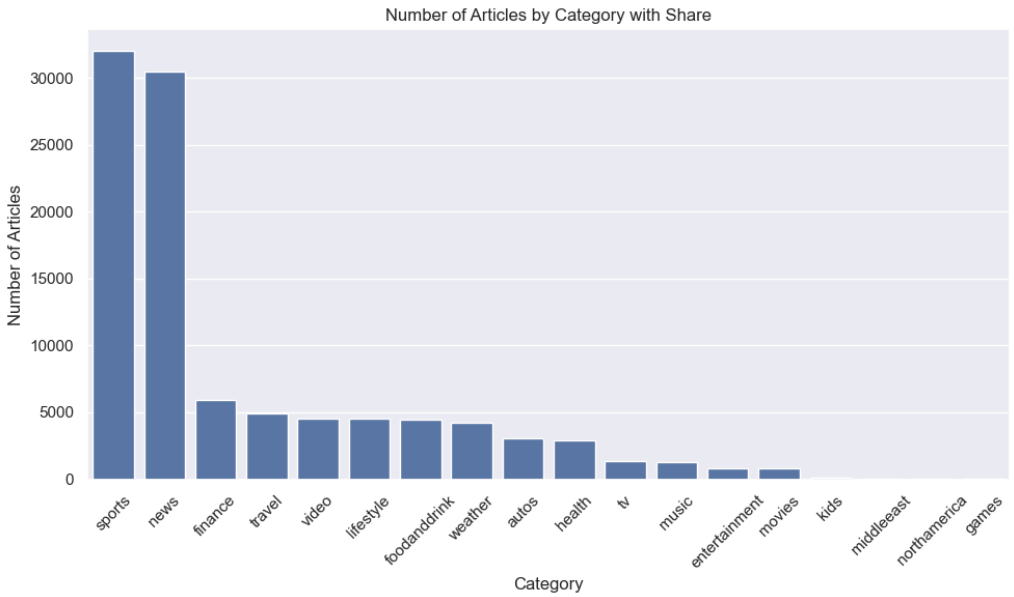

Рисунок 3.4 – Розподіл новин за категоріями

Враховуючи такий дисбаланс у категоріях новин, потрібно також перевірити, чи наявний такий дисбаланс із саб-категоріями. Можна помітити із рисунку 3.5, що для саб-категорій такий дисбаланс теж наявний, близько 25% новин – це новини про новини у США або ж про американський футбол. Отже замінити атрибут категорій виключно саб-категоріями для тренування не вийде, оскільки тут існує дисбаланс, хоча не такий сильний, як для атрибуту категорій. Тому при побудові моделей потрібно використовувати балансування вибірок для коректного тренування моделі.

Під час навчання рекомендаційних моделей однією з поширених проблем є проблема холодного старту, яка виникає у випадках появи нових користувачів або нового контенту. У зв’язку з цим, доцільно проаналізувати “тривалість

життя” кожної новини – тобто період, протягом якого вона залишається актуальною для користувачів. Попередній аналіз дає підстави припускати, що цей показник є досить низьким, оскільки щодня з’являються нові інформаційні приводи, а отже, новини швидко втрачають популярність.

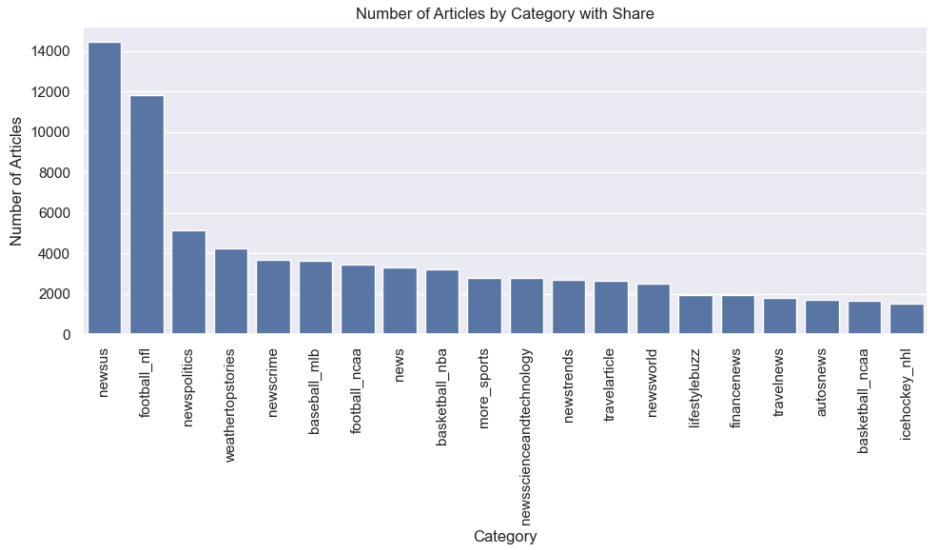

Рисунок 3.5 – Розподіл новин за саб-категоріями

Тривалість життя новини може бути визначена на основі атрибута *Impression*, який відображає факти взаємодії користувачів із конкретною новиною. Для цього обчислюється різниця між часом першої та останньої взаємодії з новиною. Як показано на рисунку 3.6, більшість новин мають життєвий цикл тривалістю менше ніж два дні. Це свідчить про те, що під час тренування моделі виникатиме проблема холодного старту, оскільки система повинна мати здатність ефективно прогнозувати рекомендації навіть для нових новин, що тільки-но з’явилися в інформаційному просторі.

Розвідувальний аналіз даних дає змогу глибше зрозуміти особливості структури та розподілу наявного датасету, що є критично важливим для побудови ефективних рекомендаційних моделей. Встановлено, що дані мають суттєві особливості, зокрема: наявність користувачів без історії взаємодії, розрідженість позитивних прикладів у кліках (Impression), а також значний

дисбаланс у категоріях і саб-категоріях новин. Також оцінка “життєвого циклу” новин засвідчила актуальність проблеми холодного старту і для наших даних.

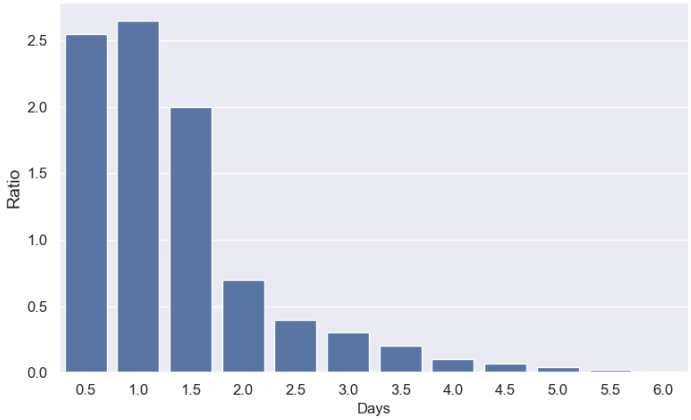

Рисунок 3.6 – Тривалість життя новин

3.2 Підготовка даних

У рамках дослідження передбачено розробку двох моделей: FFM (*Field-aware Factorization Machine*) та DKN (*Deep Knowledge Network*). Для кожної з них необхідно реалізувати окремий алгоритм попередньої обробки даних, оскільки вони використовують різні підходи до представлення інформації. Зокрема, для моделі DKN буде додатково застосовано векторне представлення текстових характеристик новин за допомогою ембедингів. Це дозволить продемонструвати переваги врахування семантичного змісту контенту під час тренування моделі. У зв’язку з цим структура даного розділу передбачає поділ на два підрозділи – відповідно до кожної з моделей.

3.2.1 FFM

Насамперед здійснено обробку даних, що стосуються новин, оскільки ця частина містить найменшу кількість змінних і є структурно найпростішою. Для забезпечення коректного функціонування моделей на етапі навчання необхідно закодувати категорії та підкатегорії новин у числовому форматі. Зважаючи на розрідженість вихідної матриці після попередньої обробки, прийнято рішення застосувати метод one-hot encoding для подання зазначених атрибутів.

Одновимірне кодування (one-hot encoding) – це алгоритм, який передбачає представлення кожної категорії у вигляді двійкового вектора. У цьому процесі для кожної унікальної категорії створюється бінарний вектор, усі елементи якого дорівнюють нулю, окрім того, що відповідає категорії даного спостереження, який дорівнює одиниці. У результаті виходить матриця бінарних векторів, що представляють категоріальні змінні в наборі даних [21].

Наступним етапом є обробка даних, що стосуються користувачів. Насамперед необхідно отримати ідентифікатори новин, з якими взаємодіяв кожен користувач. Для цього слід розгорнути список взаємодій, який зберігається у вигляді масиву. Застосовано функцію explode() з бібліотеки Polars, що дає змогу трансформувати вкладені списки у форматі «один до багатьох» у табличне представлення [22]. Після цього з кожного розгорнутого запису можна отримати відповідну мітку новини, необхідну для подальшого аналізу та побудови моделі.

Після отримання необхідних міток новин наступним кроком є формування словника ознак на основі кожного рядка даних. Це необхідно для побудови розрідженої матриці, яка згодом буде використана для навчання моделі. Для цього здійснюється перетворення кожної релевантної колонки у текстове представлення з подальшим об’єднанням їх в один спільний атрибут. Такий підхід дає змогу спростити процес побудови єдиного векторного простору ознак та забезпечує сумісність даних із вимогами FFM.

Після того, як створено словник ознак для кожної сутності даних, можна переходити до формування розрідженої матриці ознак. Для цього використано функцію із бібліотеки scikit-learn – DictVectorizer. DictVectorizer – цей трансформатор перетворює списки відображень (dict-подібні об'єкти) назв ознак до значень ознак у масиви Numpy або матриці scipy.sparse для використання з оцінками scikit-learn [23] Після перетворення наших даних за допомогою цієї функції можна переходити до тренування моделі із цими даними.

У підсумку, на етапі обробки даних для моделі FFM послідовно закодовано категоріальні змінні новин за допомогою методу one-hot encoding, що дозволило отримати зручне двійкове представлення категорій і підкатегорій. Для роботи з користувацькими взаємодіями застосовано функцію розгортання вкладених списків (explode), що спростило структуру даних та полегшило отримання необхідних міток новин. Далі сформовано словник ознак та створено розріджену матрицю за допомогою DictVectorizer, що забезпечило сумісність з алгоритмами навчання і підготувало дані до подальшого тренування моделі.

3.2.2 DKN

На початковому етапі обробки даних для моделі DKN (Deep Knowledge-Aware Network) необхідно трансформувати інформацію про взаємодію користувачів з новинами. Загальний підхід до попередньої обробки має схожість із підготовкою даних для моделі Factorization Machine, проте існують ключові відмінності, зокрема у форматі зберігання та представлення даних.

Перш за все, здійснюється розгортання атрибута, який містить історію взаємодій користувача із новинами (Impression). Це дає змогу представити кожну окрему взаємодію як окремий рядок, що значно спрощує подальшу обробку та уможливлює побудову послідовних входів, необхідних для роботи моделі DKN.

Після попереднього розгортання взаємодій здійснюється кодування кожного користувача у числовий формат. Це є необхідною умовою для коректного функціонування моделі, оскільки більшість сучасних нейромережевих архітектур, включно з DKN, не можуть працювати безперпосередньо з категоріальними змінними у текстовому вигляді. Кодування здійснюється шляхом присвоєння кожному користувача унікального номеру, відповідність потім зберігається у словник для подальшого використання на стадії валідації.

Із розділу 3.1 ми дізналися, що ми мали достатньо незбалансований датасет по частині нажатих новин, частка яких в середньому випадку для кожної взаємодії складала близько 10%. Тому для збалансованого навчання нам потрібно вирівняти цей показник. Для цього використано Негативну вибірку (Negative Sampling). Для кожної позитивної новини ми створюємо приклад із однією позитивною і декількома негативними новинами. Замість всіх негативних новин, ми використовуємо лише частину, що по-перше значно пришвидшує навчання через зменшення кількість даних і також забезпечує збалансованість. Після цих кроків ми створюємо фінальний датасет із даними щодо користувачів. Тепер можна переходити до обробки новинних атрибутів.

Першим кроком, ми формуємо словник сутностей (entity2int) з метою кодування унікальних об’єктів із тексту новин у числові індекси, які будуть використовуватися як додаткові ознаки для моделі.

Цей крок полягає у витяганні з текстів заголовків та анотацій конкретних сутностей (наприклад, імен, локацій, організацій), які відображають зміст новин на більш глибокому рівні. Для кожного ентитету обчислюється частота поява із урахуванням рівня впевненості його розпізнавання, що дає змогу відфільтрувати ненадійні або рідкісні сутності.

Після цього кожній сутності, що пройшла фільтрацію за порогом частоти, присвоюється унікальний цілий індекс, утворюючи словник entity2int.

Наступним кроком на основі згенерованих словників entity2int та word2int формується векторне подання (ембеддинги). Ці ембеддинги виступають

числовими ознаками, які подаються на вхід нейронній мережі, що дає змогу ефективно моделювати семантичні зв’язки між текстовими елементами та знаннями про контент. Таким чином, застосування ембедингів сприяє покращенню якості рекомендаційної системи шляхом врахування глибших сенсових зв’язків у даних.

Далі витягуються позиції слів у заголовку, які мають відповідні прив’язки до сутностей у базі (titles2int) у базі знань. Наприклад, якщо слово "Spain" відповідає сутності з індексом Q29, то у відповідній позиції масиву міститиметься індекс Q29.

На наступному етапі препроцесингу даних здійснюється генерація матриці векторних представлень слів (word embeddings), необхідних для подальшої роботи моделі. Для цього використовується попередньо навчена модель векторних представлень слів, зокрема GloVe, яка надає якісні вектори на основі великого корпусу текстових даних. Основною метою цього етапу є формування повного набору ембедингів для словника слів, отриманого на попередньому кроці, щоб кожне слово у корпусі мало відповідне числове представлення.

Спершу завантажується словник word2int, який містить унікальні індекси для кожного слова, відібраного за частотою у заголовках. Паралельно імпортуються pretrained вектори слів із файлу, pretrained embedings за допомогою GloVe.

GloVe – це алгоритм навчання без вчителя (unsupervised learning) для отримання векторних представлень слів [24].

Надалі здійснюється інтеграція словника зі pretrained embeddings, при цьому зберігаються лише ті слова, для яких доступні готові вектори. У випадках, коли слово відсутнє у pretrained embeddings, його вектор ініціалізується випадковими значеннями з нормального розподілу. Такий підхід дає змогу моделі адаптуватися до нових або рідкісних слів, які не враховані у GloVe.

В результаті препроцесингу даних створено набір даних для моделі DKN. Розгортання історії взаємодій користувачів з новинами до заздалегідь визначеної довжини одним з основних кроків попередньої обробки, що гарантує послідовну

структуру вхідних даних. Для подальшої подачі даних у нейронну мережу унікальні ідентифікатори новин, слів та об'єктів були перекодовані у відповідні числові індекси за допомогою словників word2int та entity2int.

Окрему увагу приділено балансуванню датасету за рахунок негативної вибірки – до кожної позитивної взаємодії користувача з новиною додавались приклади новин, на які користувач не натиснув, що дає змогу моделі навчатися розрізняти релевантні та нерелевантні приклади. У процесі формування словників додано параметр, який дав змогу відсіювати слова, які з’являлися у заголовках новин, менше заданого параметра, що зменшило шум у даних та обмежило розмірність та гнучко підлаштовувати параметр для тренування моделі

На завершальному етапі сформовано векторні представлення для всіх згаданих слів і сутностей на основі попередньо натренованих ембедингів GloVe. Це дає змогу моделі враховувати семантичні зв’язки між словами. Ембединги слів і сутностей збережено у форматі NumPy, що забезпечує швидке та ефективне завантаження їх під час навчання моделі.

3.3 Побудова рекомендаційних систем

3.3.1 FFM

Для побудови рекомендаційної системи в цьому підрозділі використано пакет xLearn. xLearn – це високопродуктивний, простий у використанні та масштабований пакет машинного навчання, який можна використовувати для вирішення масштабних завдань машинного навчання, особливо для задач на великих розріджених даних, що дуже часто зустрічається в таких задачах, як прогнозування CTR та рекомендаційних системах [25].

На відміну від звичайних Factorization Machines (FM), які моделюють взаємодії між усіма парами ознак за допомогою латентних векторів, FFM

враховує поле кожної ознаки. Це означає, що кожна ознака має окремий латентний вектор для кожного поля, з яким вона може взаємодіяти [26].

У процесі навчання моделі були підібрані оптимальні значення гіперпараметрів. Зокрема, швидкість навчання (Learning Rate) встановлено на рівні 0.2. Підвищення цього параметра призводило до коливань значень функції втрат без досягнення мінімуму, тоді як зниження до 0.1 уповільнило процес навчання до неприйнятного рівня.

Крім того, виявлено схильність моделі до перенавчання у разі відсутності регуляризації. Тому до функції втрат додано L2-регуляризаційний член із коефіцієнтом 0.002, що дозволило покращити узагальнювальну здатність моделі.

Кількість епох навчання встановлено на рівні 20, що забезпечило збалансоване співвідношення між якістю моделі та часом тренування.

3.3.2 DKN

Deep Knowledge-aware Network (DKN) – це нейронна мережа, яка розроблена для задач персоналізованої рекомендації новин. Вона інтегрує глибинне семантичне представлення новинних текстів із зовнішніми знаннями про сутності, що дає змогу моделі краще враховувати контекстуальну інформацію. Крім того, DKN використовує механізм уваги, орієнтований на користувача, що забезпечує персоналізацію рекомендацій на основі історії читання конкретного користувача. Спочатку буде розглянуто KCNN.

Першим кроком, модель DKN створює векторне представлення новин за допомогою модуля Knowledge-aware Convolutional Neural Network (KCNN). Основна концепція цього підходу полягає в інтеграції зовнішніх знань про пов'язані речі (сутності) з лінгвістичною інформацією з тексту новини (заголовка). KCNN створює багатоканальне представлення кожного слова, що охоплює ембедінги, сутності із заголовків на відміну від традиційних CNN, які

просто використовують слова. Базуючись на інформації із зовнішньої бази знань (наприклад, Вікіданих), модель краще відображає семантичні зв'язки між словами завдяки такому розширеному представленню.

Після генерації багатоканального тензора (розмірності 2 або 3 × NW ×ED), ле NW – це кількість слів, а ED – розмірність ембедингу, його піддають серії згорток з різним розміром вікна. Це дає змогу моделі ідентифікувати локальні патерни в тексті, які корелюють з різною довжиною фраз. Однією з відмінних рис KCNN є використання адитивної уваги, а не традиційного max-pooling. Цей метод дає змогу моделі динамічно оцінювати значущість кожного фрагмента тексту, створюючи більш повне уявлення про новини, яке точніше враховує семантику матеріалу. Отже, кожна новина перетворюється за допомогою KCNN на вектор фіксованої розмірності, який поєднує дані про сутності у заголовку та слова використані у заголовку.

Після цього модель переходить до побудови персоналізованого вектору користувача. Для цього використовується модуль Attention, що реалізує механізм уваги, орієнтований на новину-кандидата. Основна ідея полягає в тому, щоб зважити новини, які користувач раніше прочитав, відповідно до їх релевантності поточній новині-кандидату. Це дає змогу моделі адаптивно фокусуватись на найбільш схожих або тематично пов’язаних новинах, уникаючи однакового впливу всіх переглянутих статей.

З технічної точки зору, кожна новина, яку читає користувач, об'єднується (за допомогою конкатенації) з новиною-кандидатом, щоб сформувати пари двовимірних векторів. Ці пари пропускаються через невелику двошарову нейронну мережу, яка обчислює вагу – наскільки важливою є ця новина для оцінки кандидата. Потім ці ваги нормалізуються за допомогою softmax, і отримується розподіл уваги між новинами. Кожен вектор прочитаних новин множиться на свою вагу, а результати підсумовуються, щоб сформувати остаточний вектор користувача, що має відношення до конкретної новини кандидата.

Таким чином, Attention Network забезпечує динамічне узгодження інтересів користувача з кожним кандидатом новини.

Визначення ймовірності того, що користувач натисне на певного кандидата, є останнім етапом у моделі DKN. Модуль DNN Click Predictor, базова багатошарова нейронна мережа, допомагає в реалізації цієї процедури. Загалом мета даного кроку – це визначення релевантності на основі поєднання вектора користувача, побудованого за допомогою мережі уваги, і вектора кандидата, створеного модулем KCNN.

Для цього два вектори, новини та користувача, конкатенуються в один вектор подвоєної розмірності (2×D), який подається на вхід DNN. Структура нейромережі включає два послідовних лінійних шари з активацією ReLU між ними: перший шар зменшує розмірність до прихованого шару (sqrt(2D)), другий – до скалярного значення. Після цього результуючий тензор "сплющується", утворюючи ймовірність кліку у межах [0,1] для кожного елемента.

Таким чином, DNN Click Predictor виконує функцію бінарного класифікатора, який оперує високорівневими векторними ознаками як з боку новинного контенту, так і з боку користувача. Об’єднуючи ці ознаки, модель генерує скалярне значення, що інтерпретується як оцінка ймовірності кліку користувача на певну новину. Цей механізм дає змогу моделі приймати остаточне рішення щодо релевантності кожної новини для конкретного користувача, що є ключовим етапом у процесі формування персоналізованих рекомендацій.

Всі ці етапи створення моделі можна переглянути на рисунку 3.7, де на більш високорівневому рівні описано архітектуру моделі із найголовнішими елементами.

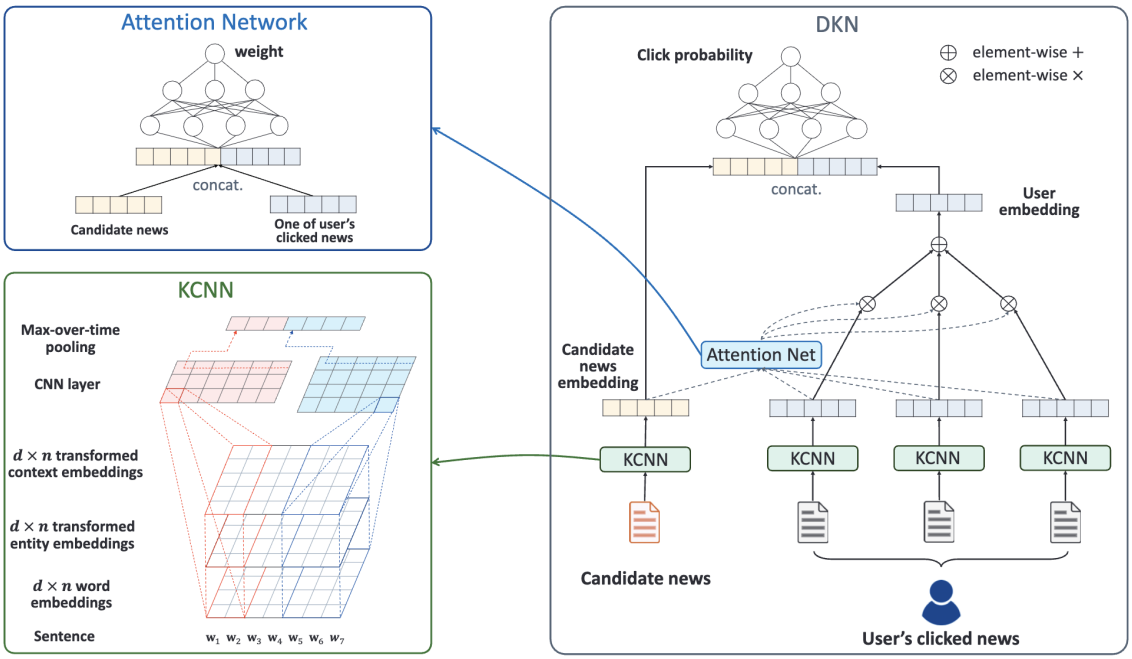

Рисунок 3.7 – Архітектура моделі DKN [17]

Тепер після пояснення основної архітектури DKN, можна переходити до розгляду основних гіперпараметрів, які напряму впливали на навчання моделі.

1. num_epochs = 20 – кількість епох або ж “проходів” по всьому навчальному датасету. Таке значення обрано, щоб запобігти перенавчання через велику кількість епох.
2. batch_size = 128 – кількість прикладів у кожному міні-пакеті. Визначає обсяг даних, який одночасно обробляється, і впливає на стабільність градієнта та швидкість навчання.
3. learning_rate = 0.0001 – темп, з яким модель оновлює ваги під час зворотного поширення (backpropagation).
4. dropout_probability = 0.2 – регуляризація, яка випадково "вимикає" частину нейронів під час тренування для зменшення перенавчання.
5. negative_sampling_ratio = 2 – кожному позитивному прикладу (новині, на яку користувач натиснув) протиставляються два негативних

(ненатиснутих), що допомагає нам зменшити дисбаланс класів натиснутих новин та не натиснутих.

6. num_clicked_news_a_user = 50 – максимальна кількість новин із історії кліків користувача, яка враховується під час побудови персонального профілю. Це встановлено, оскільки були аномальні випадки, коли у користувача близько 300 новин в історії, для пришвидшення навчання, обрано 50.

Отже, розглянуто архітектуру моделі, яка складається з трьох основних компонентів: KCNN для створення векторного представлення новин, модуля уваги для побудови персоналізованого вектору користувача та DNN Click Predictor для прогнозування ймовірності кліку. Така модульна побудова забезпечує гнучкість та точність у врахуванні як змістової інформації новин, так і індивідуальних інтересів користувача. Також були обрані гіперпараметри для забезпечення стабільного та ефективного навчання моделі, а також для зменшення перенавчання та покращення узагальнення.

3.4 Оцінка рекомендаційних систем

3.4.1 FFM

Для оцінки якості роботи моделі FFM використано метрику ROC AUC (Area Under the Receiver Operating Characteristic Curve), що відображає здатність моделі правильно розпізнавати позитивні та негативні класи при зміні порогу класифікації.

Значення AUC для тестової вибірки склало 0.514, що лише трохи вище випадкового прогнозу (0.5). Це свідчить про низьку прогностичну здатність моделі на цьому наборі даних. Що скоріше пов’язано із поганим узагальненням цієї моделі, оскільки, як буде показано далі перенавчання під час тренування не відбувалося.

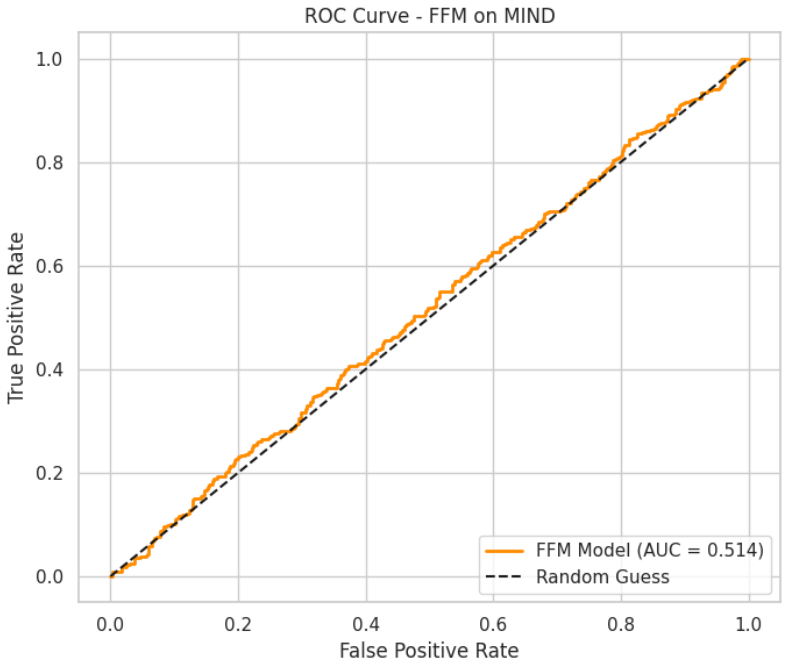

Рисунок 3.8 – ROC AUC для FFM моделі

На рисунку 3.9, можна помітити статистику тренування моделі видно, що AUC на тренувальній вибірці поступово зростає з 0.4772 на першій епосі до 0.5512 на двадцятій. Аналогічно тестовий AUC збільшується з 0.4698 до 0.5312 за 20 епох.

Із рисунку 3.10, можна побачити, що навіть при збільшенні кількості епох, скоріше за все метрики досягли плато, оскільки на останніх епохах видно, що покращення метрик на кожній епохі почало ставити все менше плюс до цього почав збільшуватися фактор перенавчання, при збільшенні регуляризаційного параметру, постраждала б вже здатність моделі до узагальнення тому прийнято рішення зупинитися на 20 епохах для даної моделі.

```
[ ACTION     ] Start to train ...
[------------] Epoch      Train AUC           Test AUC         Time cost (sec)
[   5%       ]     1           0.4772              0.4698              1812.32
[   10%      ]     2           0.4826              0.4749              1795.18
[   15%      ]     3           0.4869              0.4787              1830.44
[   20%      ]     4           0.4927              0.4835              1822.11
[   25%      ]     5           0.4975              0.4860              1819.36
[   30%      ]     6           0.5001              0.4884              1807.91
[   35%      ]     7           0.5054              0.4913              1799.87
[   40%      ]     8           0.5082              0.4940              1833.15
[   45%      ]     9           0.5105              0.4955              1825.73
[   50%      ]    10           0.5110              0.4951              1836.19
[   55%      ]    11           0.5143              0.4972              1798.22
[   60%      ]    12           0.5181              0.5007              1829.30
[   65%      ]    13           0.5220              0.5038              1810.75
[   70%      ]    14           0.5264              0.5071              1837.48
[   75%      ]    15           0.5300              0.5105              1842.33
[   80%      ]    16           0.5356              0.5149              1801.44
[   85%      ]    17           0.5417              0.5191              1835.19
[   90%      ]    18           0.5456              0.5230              1828.60
[   95%      ]    19           0.5478              0.5273              1834.01
[   100%     ]    20           0.5512              0.5312              1819.88
[ ACTION     ] Training finished at epoch 20
```

Рисунок 3.9 – Зміна метрики AUC під час тренування моделі FFM

Оцінка продуктивності моделі FFM за допомогою метрики ROC AUC продемонструвала низьку прогностичну здатність, з значенням AUC близько 0.51, що свідчить про недостатнє узагальнення моделі. Під час процесу навчання спостерігалося поступове зростання показників AUC як на тренувальній, так і на тестовій вибірках, однак досягнуто плато приблизно на 20-й епосі. З урахуванням зростаючого ризику перенавчання при подальшій оптимізації моделі прийнято рішення припинити тренування на даному етапі та перейти до дослідження альтернативних моделей з кращими узагальнювальними властивостями.

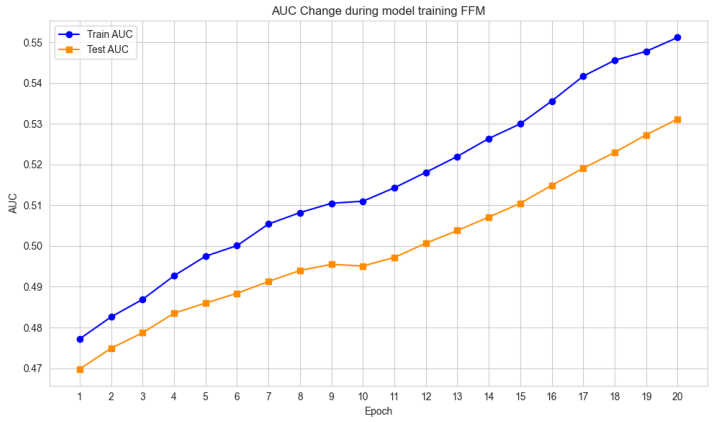

Рисунок 3.10 – Зміна AUC метрик під час тренування для FFM моделі

3.4.2 DKN

Перш за все нам потрібно за аналогією із попереднім підрозділом оцінити ROC AUC. Із рисунку 3.11, можна побачити, що площа під ROC-кривою (AUC) прогнозу DKN для валідаційних даних становить 0,753, що свідчить про її здатність розрізняти позитивні та негативні випадки.

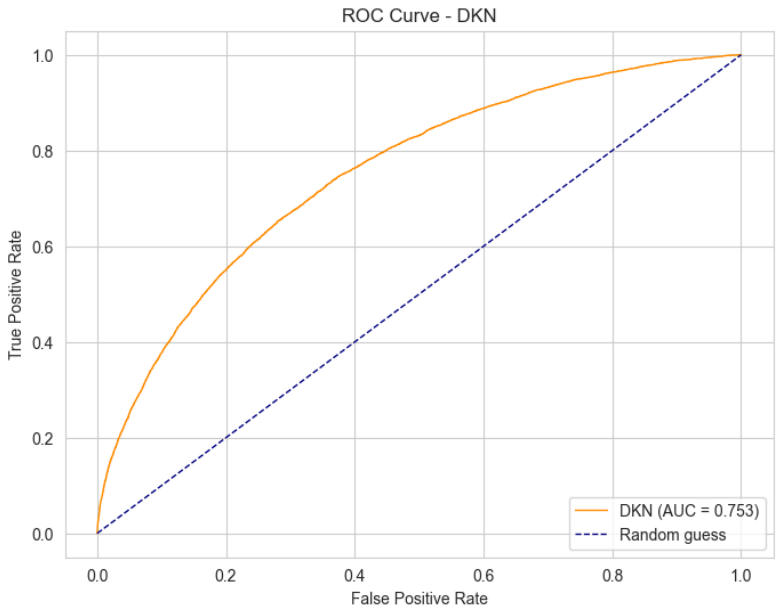

Рисунок 3.11 – ROC AUC для DKN моделі

Перевага моделі DKN полягає в тому, що вона використовує механізм уваги на основі семантичної інформації. Це дає змогу моделі краще розуміти контекст новинних статей, що має вирішальне значення для рекомендацій новин, де тематика та зміст мають вирішальне значення. Ефективність моделі FFM у цьому завданні суттєво обмежена, оскільки, на відміну від DKN, вона використовує лише попередньо закодовані числові дані та ігнорує семантичну структуру тексту.

Також для того, щоб дослідити можливість перенавчання або недонавчання моделі зібрано дані про зміну метрики AUC під час навчання моделі.

Як видно із рисунка 3.12, AUC як навчальної, так і тестової вибірок зростає з часом. Показник починається приблизно з 0,50 і поступово зростає протягом фази навчання, досягаючи 0,78 для навчальної вибірки і 0,76 для тестової вибірки наприкінці останньої епохи.

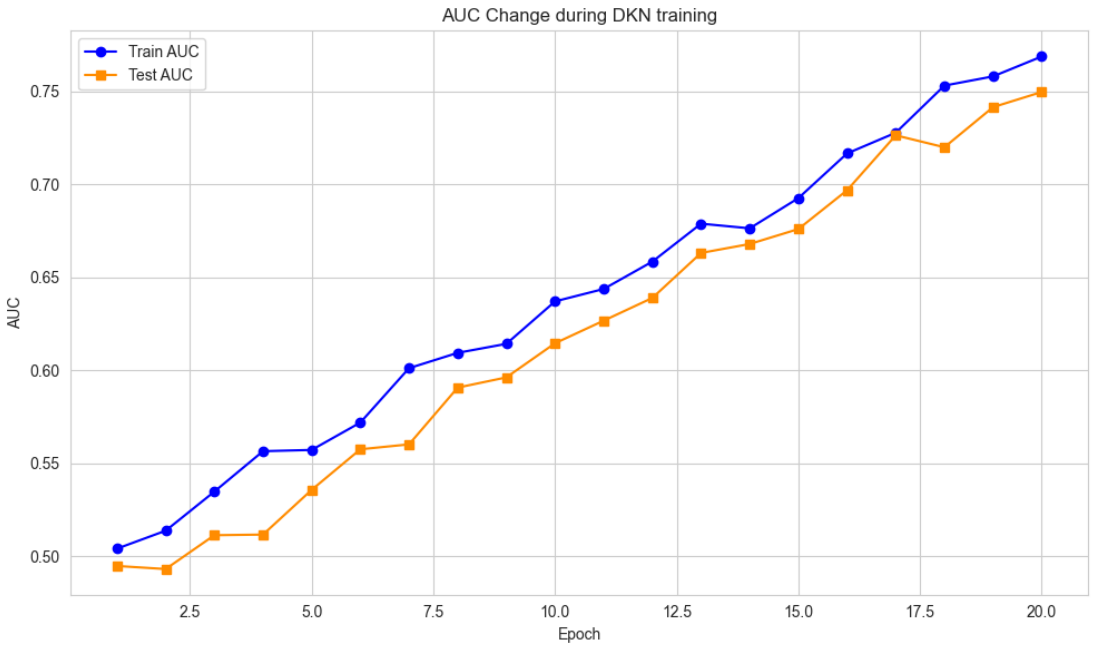

Рисунок 3.12 – Зміна AUC метрик під час тренування для DKN моделі

На противагу цьому результату, модель FFM продемонструвала ознаки як недонавчання (низька точність на навчальних даних), так і перенавчання (суттєва різниця між AUC на навчальній та тестовій вибірках зі збільшенням кількості епох), на додаток до того, що AUC наближається до 0,5. Це свідчить про недостатню здатність моделі FFM до узагальнення, особливо коли текстові та семантичні дані присутні, але не беруться до уваги.

Отже, порівняльний аналіз моделей FFM та DKN показав значну різницю в їхній ефективності. FFM досягла AUC лише 0.514, що близько до випадкового

прогнозу, і свідчить про слабку здатність моделі до узагальнення. Незважаючи на зростання метрик у процесі навчання, прогрес сповільнився після 20 епох, а подальше навчання могло призвести до перенавчання при тому, що модель і так була недонавчена.

Натомість модель DKN показала значно кращий результат із AUC на рівні 0.753, продемонструвавши стабільне покращення на всіх етапах тренування. Завдяки використанню семантичної інформації та механізму уваги, DKN краще враховує зміст новин, що робить її більш ефективною для задач рекомендації контенту.

Висновки до розділу 3

У третьому розділі виконано розробку системи персоналізованої рекомендації новин із урахуванням текстових характеристик контенту. Спочатку проведено розвідувальний аналіз даних, який виявив суттєву розрідженість взаємодій користувачів, значний дисбаланс за категоріями новин і короткий життєвий цикл новин. Ці фактори визначили специфіку подальшої роботи з даними та вибір методів моделювання.

Для побудови рекомендаційних моделей здійснено окрему підготовку даних для двох підходів. Для моделі FFM проведено кодування категоріальних змінних методом one-hot encoding, розгортання історії користувачів та створення розрідженої матриці ознак. Для моделі DKN розроблено складніший пайплайн, що включав балансування класів за допомогою негативної вибірки, кодування сутностей та слів, а також формування векторного подання тексту за допомогою попередньо навчених ембедингів GloVe, що дозволило врахувати глибоку семантику новин.

Побудовані моделі показали різний рівень якості: FFM продемонструвала низький рівень узагальнення з AUC на валідаційних даних близько 0.514, що

майже дорівнює випадковому прогнозу, натомість DKN досягла значного покращення з AUC на валідаційних даних 0.753, завдяки використанню механізму уваги та використання семантичного контексту. Така різниця свідчить про важливість включення тексту у побудову рекомендаційних систем новин.

Таким чином, результати розділу підтверджують, що застосування глибоких нейронних мереж із урахуванням семантики тексту і зовнішніх знань значно підвищує ефективність рекомендацій.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ ДЛЯ СИСТЕМИ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ НОВИН НА ОСНОВІ ТЕКСТОВИХ ХАРАКТЕРИСТИК КОНТЕНТУ

У цьому розділі система персоналізованої рекомендації контенту на основі текстових характеристик розглядається не лише як суто технічне рішення, а й як об'єкт, що потребує економічного обґрунтування. Розроблений програмний продукт може бути впроваджений у практичну діяльність онлайн-платформ, новинних порталів або інформаційних систем, що працюють із великими обсягами текстового контенту. Він також може виступати як окремий компонент у складі комплексних інформаційно-аналітичних систем.

Для оцінки доцільності впровадження та вибору оптимальних рішень у процесі розробки системи використовується функціонально-вартісний аналіз (ФВА). Цей підхід дає змогу проаналізувати співвідношення між витратами на створення програмного забезпечення та його функціональністю, а також допомагає визначити найефективніший шлях реалізації продукту з точки зору ресурсів, технологій та очікуваної користі.

ФВА є важливим інструментом при управлінні розробкою сучасних IT-продуктів. Він дає змогу не лише отримати точну оцінку вартості розробки, але й виявити можливості для оптимізації витрат без шкоди для якості. Це особливо актуально при створенні високотехнологічних систем, які потребують адаптації до динамічних вимог ринку та користувачів.

Методологія ФВА передбачає детальний аналіз функцій системи, визначення їхньої важливості, співставлення з потребами кінцевого користувача та виділення тих складових, які мають найбільший вплив на витрати. На основі цього аналізу формуються рішення щодо зменшення витрат за рахунок раціонального використання ресурсів, при цьому не знижуючи функціональні можливості або якість продукту.

4.1 Постановка задачі проектування

Застосування методу функціонально-вартісного аналізу (ФВА) дало змогу провести всебічну оцінку техніко-економічних характеристик розробки системи персоналізованої рекомендації контенту на основі текстових характеристик. Усі рішення приймались з урахуванням взаємодії окремих підсистем, що забезпечує узгоджене функціонування продукту в цілому. Тому аналіз охоплює як оцінку функціональності системи, так і її ефективність у реальних умовах експлуатації – зокрема, здатність точно формувати персоналізовані рекомендації на основі текстових характеристик контенту.

Сформульовано такі технічні вимоги до програмного забезпечення.

1. Система має забезпечувати високу точність у визначенні релевантності контенту для конкретного користувача на основі аналізу текстових даних.
2. Алгоритми повинні працювати швидко, з мінімальними затримками, наближеними до режиму реального часу, щоб забезпечити актуальність рекомендацій.
3. Розробка та впровадження рішення мають бути економічно доцільними, з оптимальними витратами на створення, інтеграцію та подальше технічне обслуговування системи.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 полягає у створенні програмного продукту, що вирішує задачу персоналізованої рекомендації контенту на основі текстових характеристик. Рішення має забезпечувати ефективну генерацію рекомендацій

відповідно до інтересів користувача. Таким чином можна вирізнити такі підфункції:

F_1 – вибір найбільш придатної мови програмування;

F_2 – визначення оптимальних методів, моделей і бібліотек для завдання;

F_3 – проведення оцінювання точності та якості роботи системи.

Кожна з цих функцій має декілька варіантів реалізації.

Функція F_1.

1. Python.
2. R.
3. Javascript.

Функція F_2.

1. Застосування готових бібліотек для попередньої обробки тексту.
2. Використання бібліотек для навчання моделі за допомогою мережі FFM.
3. Реалізація власної реалізації моделі DKN.

Функція F_3.

1. Проведення тестування системи з використанням реальних або наближених до реальних сценаріїв користувацької взаємодії.
2. Використання відкритих датасетів, наприклад MIND, які містять історію взаємодії користувачів з контентом та еталонні рекомендації.

Морфологічна карта системи (рис. 4.1) відображає варіанти реалізацій функцій описаних вище.

За допомогою морфологічної карти є можливість побудувати позитивно-негативну матрицю з відображенням усіх варіантів основних функцій(табл. 4.1).

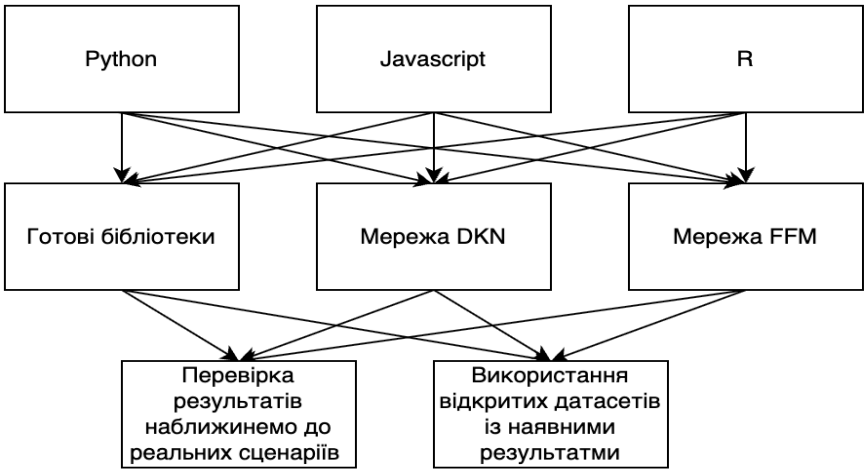

Рисунок 4.1 – Морфологічна карта

Таблиця 4.1 – Позитивно-негативна матриця

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_1	1	Найбільша кількість бібліотек для ML і NLP. Велика підтримка спільноти. Простий синтаксис	Менш ефективний у високонавантажених системах. Менше контроль над продуктивністю
	2	Потужна мова для статистичного аналізу. Добре підходить для експериментів і досліджень	Обмежені можливості для продакшн-рішень. Менше бібліотек для глибокого навчання
	3	Можливість інтеграції з веб-додатками. Можна створювати браузерні прототипи	Не призначена для складної ML-обробки. Обмежена підтримка NLP/ML-бібліотек

Кінець таблиці 4.1

Функції	*Варіанти реалізації*	*Переваги*	*Недоліки*
F_2	1	Швидкий старт розробки. Багато готових моделей. Активна підтримка спільноти	Обмеження у гнучкості. Залежність від зовнішніх змін
	2	Добре працює з категоріальними і текстовими ознаками. Висока швидкість навчання	Не враховує складну семантику тексту. Менш ефективна для довгих текстів
	3	Можливість глибокої кастомізації. Урахування знань із графу. Висока точність персоналізації	Складність реалізації. Високі обчислювальні витрати. Вимоги до якісних знань про сутності у тексті
F_3	1	Можливість оцінити якість у контексті реального використання. Підвищення адаптивності системи	Потреба у великій кількості сценаріїв. Часозатратне тестування
	2	Дає змогу порівнювати з іншими методами. Еталонна метрика якості: AUC	Можуть бути невідповідними для конкретного контексту. Ризик використання застарілих даних

На основі проведеного аналізу позитивно-негативної матриці встановлено, що окремі варіанти реалізації функцій програмного продукту не задовольняють вимоги до точності, масштабованості або відповідності цілям системи персоналізованої рекомендації контенту. У зв’язку з цим ці варіанти були виключені з подальшої розробки.

Функція F_1.

Мова програмування Python була обрана як найбільш придатна, оскільки вона вирізняється високою читабельністю, простотою синтаксису, а також широким набором бібліотек, орієнтованих на обробку текстових даних та реалізацію систем машинного навчання.

Функція F_2.

Підходять 1 або 2 варіанти, оскільки підходи різняться, проте не мають одне перед одним жодних переваг, їхнє використання є рівноцінним.

Функція F_3.

В даному випадку обидва варіанти оцінювання є раціональними. Проте пріоритет падає на використання відкритих датасетів, оскільки вони дозволяють отримати еталонні метрики якості та порівнювати результати з іншими підходами. Варіант із тестуванням на реальних сценаріях потребує більше часу на валідацію, що є важливим фактором при обмежених термінах розробки.

Маємо такі варіанти реалізації ПП:

$F_1 1 – F_2 2 – F_3 2$,

$F_1 1 – F_2 1 – F_3 2$.

Оцінювання якості згаданої функцій виконується з допомогою параметрів нижче.

4.3 Обґрунтування системи параметрів програмного продукту

З огляду на наявні сучасні технології та інструменти, здійснено аналіз ключових факторів, що визначають ефективність персоналізованої рекомендаційної системи новин, які безпосередньо впливають на вибір архітектурного підходу для реалізації проєкту.

Під час аналізу можливих технологічних рішень розглянуто доцільність використання різних мов програмування – зокрема Python, JavaScript та R – а також варіанти створення рекомендаційної моделі: від самостійної реалізації архітектури глибокого навчання до застосування існуючих бібліотек, орієнтованих на обробку текстових даних. Для обґрунтованого вибору визначено такі ключові критерії порівняння:

- $X1$ – приблизний обсяг програмного коду, необхідного для реалізації системи;
- $X2$ – тривалість навчання моделі на текстовому корпусі новин;
- $X3$ – час, потрібний для генерації персоналізованих рекомендацій у робочому середовищі;
- $X4$ – точність рекомендацій, що відображає ефективність моделі в ідентифікації інтересів користувача.

Оцінювані параметри класифікуються на гірші, середні та кращі рівні, вибір яких здійснюється відповідно до вимог замовника та специфіки використання програмного забезпечення (див. табл. 4.2).

Дані в таблиці 4.3 дозволяють побудувати графічні характеристики для оцінки та порівняння параметрів (рис. 4.2 – рис. 4.5).

Таблиця 4.2 – Основні параметри програмного продукту

Назва параметра	*Умовні позначення*	*Одиниці виміру*	*Значення параметра*		
			Гірші	*Середні*	*Кращі*
Обсяг коду, необхідний для реалізації системи	*X*1	Кількість рядків коду	2000	1500	1000
Тривалість навчання моделі на новинному корпусі	*X*2	Дні	5	4	2
Час генерації рекомендацій у середовищі користувача	*X*3	Секунди	3	2	1
Точність рекомендацій	*X*4	Значення AUC	0.75	0.7	0.65

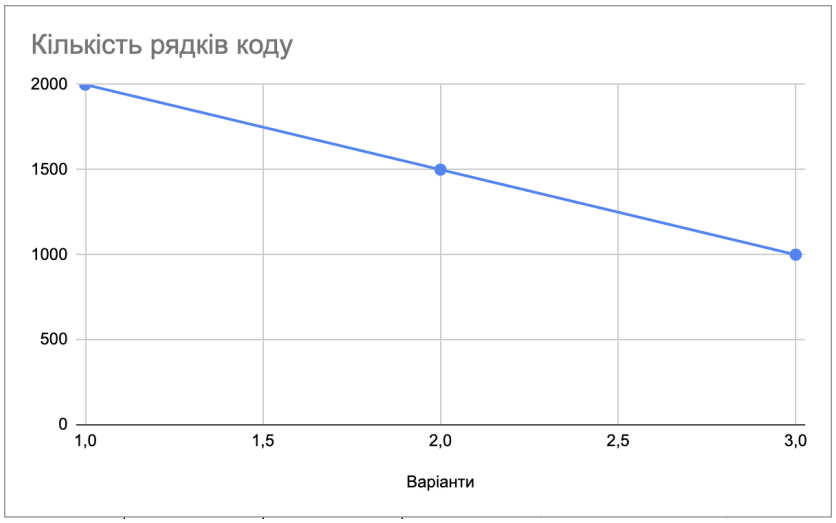

Рисунок 4.2 – *X*1, Обсяг коду, необхідний для реалізації системи

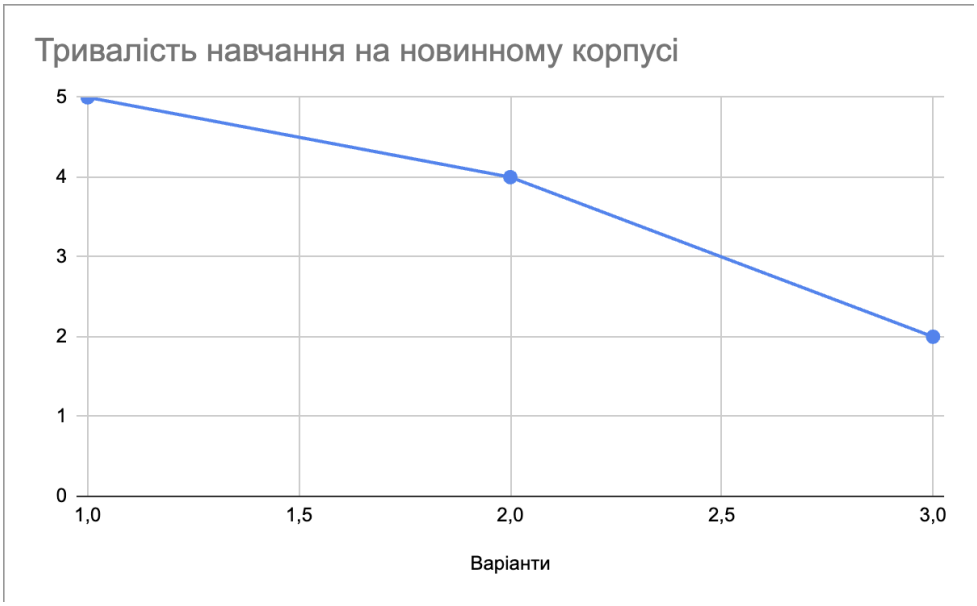

Рисунок 4.3 – *X*2,Тривалість навчання моделі на новинному корпусі

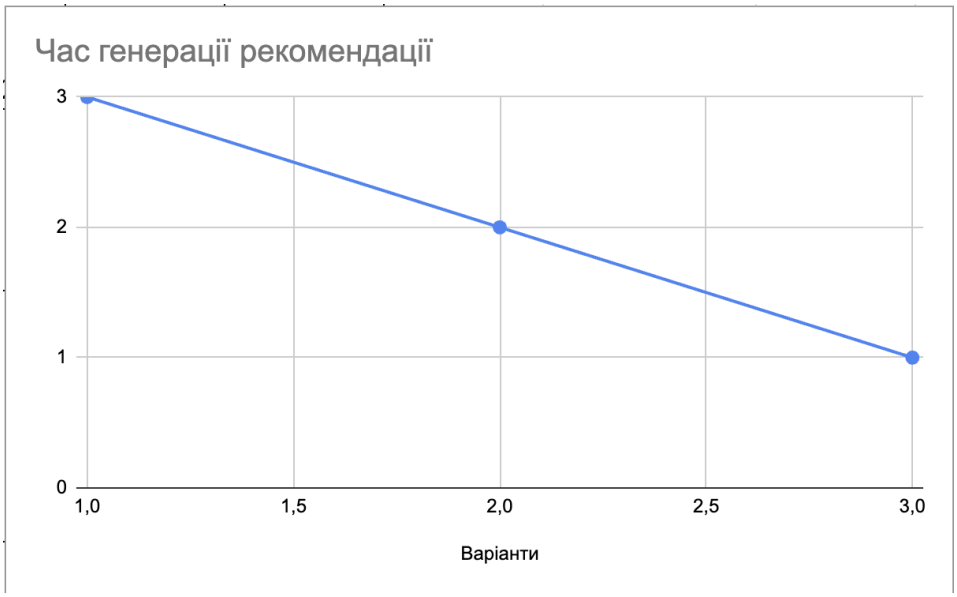

Рисунок 4.4 – *X*3, Час генерації рекомендацій у середовищі користувача

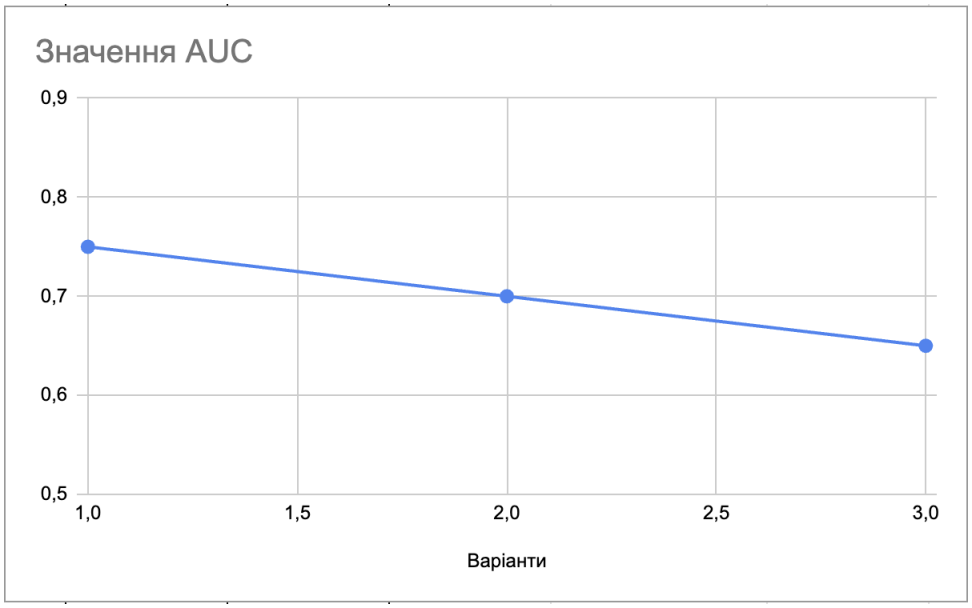

Рисунок 4.5 – *X*4, Точність рекомендації

4.4 Аналіз експертного оцінювання параметрів

З метою забезпечення обґрунтованого вибору оптимальної конфігурації програмного продукту для персоналізованих рекомендацій новин, проведено комплексну експертну оцінку важливості кожного з визначених параметрів. Основна мета цієї оцінки – визначити ступінь впливу кожного з техніко-функціональних показників на ефективність системи, зокрема на її здатність забезпечувати релевантність та точність контенту для кінцевого користувача.

Оцінювання проводилося за допомогою методу попарного порівняння, у якому брали участь чотири незалежні експерти з галузі машинного навчання, розробки програмного забезпечення та аналітики даних.

Процедура формування вагових коефіцієнтів включала такі етапи:

- проведення ранжування параметрів за ступенем їхнього впливу на кінцеву якість та продуктивність системи;

- оцінювання надійності експертних висновків, що забезпечує достовірність отриманих результатів;
- попарне порівняння параметрів, яке дає змогу встановити відносну важливість кожного з них у загальній структурі функціональності системи
- агрегування результатів та розрахунок підсумкових вагових коефіцієнтів, що відображають пріоритетність кожного критерію в контексті розробки та впровадження системи рекомендацій;

Результати рангової оцінки експертами наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Обсяг коду, необхідний для реалізації системи	кількість рядків коду	2	2	2	3	4	3	3	19	1,5	2,25
X2	Тривалість навчання моделі на новинному корпусі	дні	3	1	4	2	2	1	2	15	-2,5	6,25
X3	Час генерації рекомендацій у середовищі користувача	секунди	4	4	3	4	3	4	4	26	8,5	72,25
X4	Точність рекомендацій	Значення AUC	1	3	1	1	1	2	1	10	-7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	137

Для оцінки достовірності експертних рейтингів визначаються такі параметри:

1. Сумарний ранг кожного показника та загальний ранг усіх показників:

2.

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – кількість експертів,

n – число параметрів.

3. Середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5. \quad (4.2)$$

4. Різниця між рангами кожного параметра та середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума, що визначає відхилення кожного параметру повинна дорівнювати 0.

5. Загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} {\Delta_i}^2 = 137. \quad (4.4)$$

Коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 * 197}{7^2(4^3 - 4)} = 0.754 > W_k = 0{,}56. \quad (4.5)$$

Отримані результати ранжування можна вважати надійними, адже розрахований коефіцієнт узгодженості перевищує нормативний показник, значення якого дорівнює 0,56. Таким чином утворимо таблицю 4.4, де попарно порівняємо всі параметри.

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	*Експерти*							*Кінцева оцінка*	*Числове значення*
	1	*2*	*3*	*4*	*5*	*6*	*7*		
*X*1 і *X*2	>	<	>	<	<	<	<	<	0,5
*X*1 і *X*3	>	>	>	>	<	>	>	>	1,5
*X*1 і *X*4	<	>	<	<	<	<	<	<	0,5
*X*2 і *X*3	>	>	<	>	>	>	>	>	1,5
*X*2 і *X*4	<	>	<	<	<	>	<	<	0,5
*X*3 і *X*4	<	<	<	<	<	<	<	<	0,5

Ступінь переваги *i*–го параметра над *j*–тим визначається числовим параметром a_{ij} і визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j. \\ 0.5 \text{ при } X_i < X_j \end{cases} \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами (4.7) та (4.8).

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \quad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij}. \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки такі значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і

наступних кроках відносні оцінки розраховуються за наступними формулами (4.9) та (4.10).

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \tag{4.9}$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j. \tag{4.10}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	*Параметри* x_j				*Перша ітер.*		*Друга ітер.*	
	X1	*X2*	*X3*	*X4*	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$
*X*1	1	0,5	1,5	0,5	3,5	0,21875	12,25	0,207627
*X*2	1,5	1	1,5	0,5	4,5	0,28125	16,25	0,275424
*X*3	0,5	0,5	1	0,5	2,5	0,15625	9,25	0,15678
*X*4	1,5	1,5	1,5	1	5,5	0,34375	21,25	0,360169
Всього:					16	1	59	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X*1 (обсяг коду, необхідний для реалізації системи), *X*2 (тривалість навчання моделі на новинному корпусі) та *X*4 (точність рекомендацій) відповідають технічним вимогам умов функціонування програмного продукту.

Абсолютне значення параметра *X*3 (час генерації рекомендацій у середовищі користувача) обрано таким чином, щоб уникнути найгірших сценаріїв продуктивності.

Коефіцієнт технічного рівня для кожного варіанта реалізації програмного продукту розраховується відповідно до даних таблиці 4.6:

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j}, \qquad (4.11)$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах

За даними з таблиці 4.6 за формулою (4.12)

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}]. \qquad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 3{,}99 + 11{,}88 + 3{,}6 = 19{,}5,$$
$$K_{K2} = 3{,}99 + 9 + 3{,}6 = 16{,}6\ .$$

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	*Варіант реалізації функції*	*Параметри*	*Абсолютне значення параметра*	*Бальна оцінка параметра*	*Коефіцієнт вагомості параметра*	*Коефіцієнт рівня якості*
*F*1	1	*X*1	700	19	0,21	3,99
*F*2	1	*X*4	95	33	0,36	11,88
	2	*X*4	80	25	0,36	9
*F*3	1	*X*3	80	24	0,15	3,6

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для оцінки вартості розробки програмного продукту передусім необхідно здійснити розрахунок трудомісткості проєкту.

Усі варіанти реалізації передбачають виконання двох основних завдань.

1. Розробка проєкту програмного продукту.
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи Б, завдання 2 до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 2.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_Р$ = 78 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.045. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 78 \cdot 1.045 \cdot 1 \cdot 0.81 = 66.02 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =29 людино-днів, $К_П$ =1.08, $К_{СК}$ = 1, $К_{СТ}$ =0.7:

$$T_2 = 2910 \cdot \ 1.08 \cdot 1 \cdot 0.7 = 21{,}92 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_0 = (66.02 + 21.922) \cdot 8 = 703.52 \quad \text{людино-годин.}$$

В розробці беруть участь один аналітик даних з окладом 53000 грн. Визначимо середню зарплату за годину за формулою (4.14).

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн,} \qquad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$С_Ч = \frac{53000}{18 \cdot 8} = 368{,}05 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою (4.16).

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (4.16)$$

де $С_Ч$– величина погодинної оплати праці аналітика даних;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$С_{ЗП} = 368.05 \cdot 703.520 \cdot 1.125 = 291296{,}85 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 291296{,}85 \cdot 0.22 = 64085{,}31 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного аналітика даних з окладом 53000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 53000 \cdot 0{,}2 = 127200 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 127200 \cdot (1 + 0.2) = 152640 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 152640 \cdot 0,22 = 22580,8 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 49000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.23 \cdot 0.25 \cdot 49000 = 15067,5 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.23 \cdot 49000 \cdot 0.03 = 1808,10 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою, описаною нижче.

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 105 – 21 – 16) \cdot 8 \cdot 0.87 =$$

$$= 1552,08 \text{ години,}$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1552,08 \cdot 0,3 \cdot 0,5 \cdot 9.43 = 2195,42 \text{ грн},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою, описаною нижче.

$$С_Н = Ц_{ПР} \cdot 0.67 = 15067,5 \cdot 0,67 = 10095,23 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н. \quad (4.17)$$

$$С_{ЕКС} = 291296,85 + 22809,6 + 15067,5 + 1808,10 + 2195,42 + 10095,23 = 343272,70 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 343272,70 / 1552,08 = 221,17 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т. \quad (4.18)$$

$$С_М = 221,17 \cdot 703,52 = 155597,14 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67. \quad (4.19)$$

$$С_Н = 291296,85 \cdot 0,67 = 195168,89 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н. \quad (4.20)$$

$$С_{ПП} = 291296,85 + 22580,8 + 155597,14 + 195168,89 = 664643,68 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою (4.21).

$$К_{ТЕР j} = К_{Кj} / С_{Фj}. \quad (4.21)$$

$$КТЕР1 = 19,5 / 414697,55 = 4,7 \cdot 10^{-5}.$$

На основі проведеного функціонально-вартісного аналізу визначено трудомісткість виконання двох ключових етапів розробки програмного

забезпечення: розробки проєкту програмного продукту та розробки програмної оболонки.

Трудомісткість першого завдання склала 66,02 людино-дня. При цьому враховано використання довідкової інформації, а також застосування стандартних модулів, що відображено через поправочний коефіцієнт 1,045 та коефіцієнт використання стандартних модулів 0,8.

Друге завдання характеризується трудомісткістю 21,92 людино-дня. Його реалізація передбачає використання алгоритмів третьої групи складності, що відповідає поправочному коефіцієнту 1,08.

Сумарна трудомісткість обох завдань у перерахунку на людино-години становить 703.52. Це дає змогу здійснити попередню оцінку необхідних трудових ресурсів та тривалості виконання робіт.

Загальна вартість розробки програмного продукту оцінена в 664643,68 гривень. Ця сума включає заробітну плату виконавця, витрати на машинний час, обов’язкові соціальні внески та накладні витрати підприємства.

Значення коефіцієнта техніко-економічного рівня, що становить $4{,}7 \times 10^{-5}$, свідчить про високу вартість розробки відносно технічних характеристик та очікуваної комерційної віддачі. Це обумовлює доцільність проведення додаткового аналізу інвестиційної ефективності та потенціалу оптимізації витрат.

Висновки до розділу 4

На основі аналізу вартості розробки системи персоналізованих рекомендацій, що ґрунтується на текстових характеристиках контенту, можна зробити висновок про значний потенціал такого програмного продукту як з практичної, так і з комерційної точки зору. Застосування функціонально-вартісного аналізу дало змогу всебічно оцінити економічну ефективність

системи, зокрема з точки зору оптимізації витрат на розробку, навчання моделі та експлуатацію. Розроблені альтернативні варіанти реалізації ключових функцій системи дозволяють гнучко адаптувати її під різні технічні вимоги та бюджетні обмеження. У підсумку, система не лише відповідає сучасним вимогам до точності рекомендацій та швидкості їх генерації, але й є економічно доцільною завдяки оптимальному співвідношенню витрат і якості. Враховуючи високу вартість розробки у порівнянні з технічними параметрами і комерційною вартістю, важливим є подальший пошук шляхів для зниження витрат і підвищення ефективності інвестицій, що може бути вирішено через оптимізацію вибраних технологій та підходів до розробки.

ВИСНОВКИ

У результаті проведеної роботи реалізовано систему персоналізованої рекомендації новин на основі текстових характеристик контенту. розглянуто сучасні методи побудови рекомендаційних систем, акцент зроблено на моделях із використанням глибокого навчання для аналізу текстових характеристик. Порівняння різних підходів дозволило виокремити найбільш ефективне рішення.

Під час роботи над системою були проведені експерименти з різними архітектурами, виконано аналіз точності рекомендацій відповідно до сучасних метрик. Це дозволило оцінити якість моделі та зробити обґрунтований вибір на користь найбільш ефективного підходу.

Створено повноцінний програмний продукт, що демонструє практичну реалізацію обраної моделі, та здійснено оцінку його ефективності в умовах реальних сценаріїв застосування. Також виконано техніко-економічний аналіз реалізованого рішення та обґрунтовано доцільність його впровадження з урахуванням витрат і очікуваного результату.

Окрему увагу приділено роботі з текстовими характеристиками новин, що дало змогу моделювати зміст на більш глибокому семантичному рівні. Такий підхід покращив релевантність рекомендацій та дозволив краще адаптуватися до інтересів користувача. Отримані результати можуть бути використані як основа для подальших досліджень у сфері персоналізації контенту та розвитку інтелектуальних інформаційних систем.

Загалом, виконана робота продемонструвала ефективність використання сучасних підходів глибокого навчання у задачах рекомендації, а також виявлено потенціал подальшого розвитку таких систем у динамічному інформаційному середовищі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ofcom. News consumption in the UK 2024. URL: https://www.ofcom.org.uk/siteassets/resources/documents/research-and-data/tv-radio-and-on-demand-research/tv-research/news/news-consumption-2024/news-consumption-in-the-uk-2024-report.pdf?v=379621 (date of access: 05.03.2025).
2. Zhu J. Research on user behavior and content consumption trends in the era of new media. *Frontiers in business, economics and management*. 2024. Vol. 15, no. 2. P. 265–268. URL: https://doi.org/10.54097/k34e0531 (date of access: 07.03.2025).
3. Rich E. User modeling via stereotypes*. Cognitive science. 1979. Vol. 3, no. 4. P. 329–354. URL: https://doi.org/10.1207/s15516709cog0304_3 (date of access: 11.03.2025).
4. Linden G., Smith B., York J. Amazon.com recommendations: item-to-item collaborative filtering. IEEE internet computing. 2003. Vol. 7, no. 1. P. 76–80. URL: https://doi.org/10.1109/mic.2003.1167344 (date of access: 27.03.2025).
5. Covington P., Adams J., Sargin E. Deep neural networks for youtube recommendations. RecSys '16: tenth ACM conference on recommender systems, Boston Massachusetts USA. New York, NY, USA, 2016. URL: https://doi.org/10.1145/2959100.2959190 (date of access: 02.04.2025).
6. Feature-combination hybrid recommender systems for automated music playlist continuation / A. Vall et al. User modeling and user-adapted interaction. 2019. Vol. 29, no. 2. P. 527–572. URL: https://doi.org/10.1007/s11257-018-9215-8 (date of access: 07.04.2025).
7. Gomez-Uribe C., Hunt N. The netflix recommender system: algorithms, business value,and innovation. ACM transactions on management information systems. 2016. Т. 6, № 4. С. 1–1 (date of access: 07.04.2025).

8. Thiyagarajan K. Exploring multimodal large language models for next-generation recommendation systems. International journal of computer trends and technology. 2025. Vol. 73, no. 2. P. 64–70. URL: https://doi.org/10.14445/22312803/ijctt-v73i2p108 (date of access: 08.04.2025).
9. Al-Ghuribi S. M., Mohd Noah S. A. A comprehensive overview of recommender system and sentiment analysis. Arxiv. URL: https://doi.org/10.48550/arXiv.2109.08794 (date of access: 08.04.2025).
10. Є. М. Проблеми сучасних рекомендаційних систем та методи їх рішення. Системи управління, навігації та зв'язку. Збірник наукових праць. 2018. Т. 4. С. 120–124 (дата звернення: 12.04.2025).
11. M N. Personalized product recommendation system for amazon. International journal for research in applied science and engineering technology. 2025. Vol. 13, no. 4. P. 1992–1997. URL: https://doi.org/10.22214/ijraset.2025.68607 (date of access: 13.04.2025).
12. Bravo B., Indra I. Article recommendations with item-based collaborative filtering on online news portals. Journal of information systems and informatics. 2024. Vol. 6, no. 3. P. 1962–1972. URL: https://doi.org/10.51519/journalisi.v6i3.851 (date of access: 17.04.2025).
13. B.Thorat P., M. Goudar R., Barve S. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. International journal of computer applications. 2015. Vol. 110, no. 4. P. 31–36. URL: https://doi.org/10.5120/19308-0760 (date of access: 28.04.2025).
14. Lundquist E. Factorization machines for item recommendation with implicit feedback data. Medium. URL: https://medium.com/data-science/factorization-machines-for-item-recommendation-with-implicit-feedback-data-5655a7c749db (date of access: 30.04.2025).
15. Rendle S. Factorization machines. 2010 IEEE international conference on data mining. 2010. URL: https://analyticsconsultores.com.mx/wp-

content/uploads/2019/03/Factorization-Machines-Steffen-Rendle-Osaka-University-2010.pdf (date of access: 02.05.2025).

16. Deep learning based recommender system / S. Zhang et al. ACM computing surveys. 2019. Vol. 52, no. 1. P. 1–38. URL: https://doi.org/10.1145/328502 (date of access: 03.05.2025).
17. DKN: deep knowledge-aware network for news recommendation / H. Wang et al. The 27th international conference on world wide web. 2018. URL: https://arxiv.org/pdf/1801.08284 (date of access: 04.05.2025).
18. Neural News Recommendation with Multi-Head Self-Attention / C. Wu та ін. Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP), м. Hong Kong, China. Stroudsburg, PA, USA, 2019. URL: https://doi.org/10.18653/v1/d19-1671 (date of access: 04.05.2025).
19. Neural news recommendation with attentive multi-view learning / C. Wu та ін. Twenty-Eighth international joint conference on artificial intelligence {IJCAI-19}, м. Macao, China, 10–16 серп. 2019 р. California, 2019. URL: https://doi.org/10.24963/ijcai.2019/53 (date of access: 07.05.2025).
20. Natrella M. E-Handbook of statistical methods. Nist/sematech. 2010.
21. Missing data preprocessing in credit classification: one-hot encoding or imputation? / L. Yu et al. Emerging markets finance and trade. 2020. P. 1–11. URL: https://doi.org/10.1080/1540496x.2020.1825935 (date of access: 12.05.2025).
22. Polars documentation. https://docs.pola.rs/api/python/stable/reference/index.html. URL: https://docs.pola.rs/api/python/stable/reference/dataframe/api/polars.DataFrame.explode.html (date of access: 12.05.2025).
23. DictVectorizer. https://scikit-learn.org. URL: https://scikit-learn.org/0.15/modules/generated/sklearn.feature_extraction.DictVectorizer.html (date of access: 12.05.2025).

24. Pennington J., Socher R., Manning C. D. GloVe: global vectors for word representation. 2013. URL: https://aclanthology.org/D14-1162.pdf (date of access: 13.05.2025)
25. XLearn. https://xlearn-doc.readthedocs.io/en/latest/. URL: https://xlearn-doc.readthedocs.io/en/latest/ (date of access: 15.05.2025).
26. Field-aware factorization machines for CTR prediction / Y. Juan et al. 2016.
27. Недашківська Н.І. Дистанційний курс «Інтелектуальний аналіз даних». Платформа дистанційного навчання «Сікорський»: (Сертифікат серія ДК №0256 затверджено Методичною радою КПІ ім. Ігоря Сікорського, протокол № 4 від 01.02.2024 р.)
28. Недашківська Н.І. Дистанційний курс «Інтелектуальний аналіз даних і прийняття рішень». Платформа дистанційного навчання «Сікорський»: (Сертифікат серія ДК №0257 затверджено Методичною радою КПІ ім. Ігоря Сікорського, протокол № 4 від 01.02.2024 р.)
29. Henrik Brink Joseph W. Richards Mark Fetherolf. Real-World Machine Learning. Manning Publications. 2016. 336 p.
30. Wes McKinney. Python for Data Analysis. O'Reilly Media Inc., 2013. 482 p.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

Файл EDA.ipynb

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import polars as pl
#%%
beh_train = pl.read_csv('MINDlarge_train/behaviors.tsv',
                        separator='\t',
                        row_index_offset=0,
                        new_columns=['Impression ID',
                                     'User ID',
                                     'Time',
                                     'History',
                                     'Impression'])

beh_train = beh_train.with_columns(
    pl.col("Time").str.replace(r" (\d):", r" 0$1:", literal=False).alias('Time')
)

beh_train = beh_train.with_columns(
    pl.col("Time").str.replace(r"(\d)/(\d){1}/", r"$1/0$2/",
literal=False).alias('Time')
)

beh_train = beh_train.with_columns(
    pl.col('Time').str.strptime(
        pl.Datetime,
        format="%m/%d/%Y %I:%M:%S %p"
    ).alias('Time')
)
beh_train
#%%
print(f'Min Time: {beh_train.select(pl.col('Time').min())}')
print(f'Max Time: {beh_train.select(pl.col('Time').max())}')
#%%
news_train = pl.read_csv('MINDlarge_train/news.tsv',
                         separator='\t',
                         row_index_offset=0,
                         new_columns=['News ID',
                                      'Category',
                                      'Sub-Category',
                                      'Title',
                                      'Abstract',
                                      'URL',
                                      'Title Entities',
                                      'Abstract Entities'],
                         encoding="utf8-lossy",
                         quote_char=None)

news_train = news_train.with_columns(
    pl.col('Title Entities').map_batches(
        lambda x: x.str.replace_many(["[", "]"], '')).alias('Title Entities')
)
```

```
news_train = news_train.with_columns(
    pl.col('Abstract Entities').map_batches(
        lambda x: x.str.replace_many(["[", "]"], '')).alias('Abstract Entities')
)

news_train = news_train.with_columns(
    pl.col('News ID').str.replace_many(['"'], ''),
    pl.col('Category').str.replace_many(['"'], ''),
    pl.col('Sub-Category').str.replace_many(['"'], ''),
    pl.col('Title').str.replace_many(['"'], ''),
    pl.col('Abstract').str.replace_many(['"'], '')
)

news_train
#%%
count_df = news_train.select(pl.col('News ID'), pl.col('Category'))
count_df = count_df.group_by('Category').agg(
    pl.col('News ID').count().alias('Number of Articles'),
)
count_df = count_df.with_columns(
    (pl.col('Number of Articles') * 100 / pl.sum('Number of
Articles')).round(2).alias('Share')
)
count_df
news_only = news_train.filter(pl.col('Category') == 'news')
news_only
#%%
print(f'Number of users: {beh_train['User ID'].unique().shape[0]}\n'
      f'Number of article: {news_train['News ID'].unique().shape[0]}\n'
      f'Number of impressions: {beh_train.shape[0]}')
#%% md
#### Survival News Check
#%%
sur_time = beh_train.select(['History', 'Time'])


def explode_list_column(df, col, separator=' '):
    new_col_name = f'New {col}'
    new_df = (
        df
        .filter(pl.col(col).is_not_null())
        .with_columns([
            pl.col(col).str.split(separator).alias(new_col_name)
        ])
    )

    calc_df = new_df.sample(fraction=0.5)

    calc_df = (
        calc_df
        .explode(new_col_name)
    )

    return calc_df


calc_df = explode_list_column(beh_train, 'History').select(pl.col('New
History'),
                                                           pl.col('Time'))

calc_df = (
    calc_df.group_by('New History').agg([
        pl.col('Time').min().alias('Min Time'),
```

```
        pl.col('Time').max().alias('Max Time')
    ])
)

calc_df = calc_df.with_columns(
    (pl.col('Max Time') - pl.col('Min Time')).alias('Time Difference'),
    (pl.col('Max Time') - pl.col('Min
Time').dt.date()).dt.total_hours().alias('Hours')
)

plot_df = calc_df.select(pl.col('Hours'))

plot_df = plot_df.to_pandas()
#%%
def define_days(x):
    return int(x / 24)


plot_df['days'] = plot_df['Hours'].map(define_days)
sns.histplot(data=plot_df['days'], bins=5)
#%%
null_share = beh_train.select([
    (pl.col("History").is_null().sum() * 100 / pl.len()).alias("null_share")
])
print(f'Null Share: {round(null_share[0, 'null_share'], 2)}%')
#%% md
#### Impression Sparsity
#%%
imp_df = explode_list_column(beh_train, 'Impression')

imp_df = imp_df.with_columns(
    pl.col('New Impression').str.split('-').list.to_struct(fields=['News
Impression', 'Bool']).alias("New Impression")
).unnest('New Impression')

imp_df = imp_df.cast({'Bool': pl.Int8})

sparsity = (imp_df
            .select(['Impression ID', 'News Impression', 'Bool'])
            .group_by('Impression ID')
                .agg(pl.col('News Impression').count().alias('Impression
Count'),
                     pl.col('Bool').sum().alias('Bool Count'))
            )

sparsity = sparsity.with_columns(
    (pl.col('Bool Count') * 100.0/pl.col('Impression
Count')).round(2).alias('Share of Not-Null')
)

median = sparsity.select(pl.median('Share of Not-Null'))
mean = sparsity.select(pl.mean('Share of Not-Null'))

print(f'Median Sparsity: {median[0, 'Share of Not-Null']}\nMean Sparsity:
{mean[0, 'Share of Not-Null']}')
#%%
sparsity
#%%
count_df = news_train.select(pl.col('News ID'), pl.col('Sub-Category')
,pl.col('Category'))
count_df = count_df.group_by('Category', 'Sub-Category').agg(
    pl.col('News ID').count().alias('Number of Articles'),
)
```

```
count_df = count_df.with_columns(
    (pl.col('Number of Articles') * 100 / pl.sum('Number of
Articles')).round(2).alias('Share')
)
count_df
```

Файл fm-model.ipynb

```
import data manipulation libraries
import pandas as pd
import numpy as np
import polars as pl

#visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

#Preprocessing
from sklearn.feature_extraction import DictVectorizer
from sklearn.preprocessing import LabelEncoder

#Other
import psutil
import json
import gc

#Test Efficeincy
from sklearn.metrics import roc_curve, roc_auc_score

#model
import xlearn as xl
#%%
#Function to take only n% of the dataframe
def take_sample_date_respectively(df, day_column, sample_percent):
    sampled = df.group_by(day_column).map_groups(
        lambda group: group.sample(n = int(len(group) * sample_percent),
with_replacement = False) if len(group) > 0 else group
    )
    return sampled

#Function for column Explode
def explode_list_column(df, col, separator=' '):
    new_df = (
        df
        .filter(pl.col(col).is_not_null())
        .with_columns([
            pl.col(col).str.split(separator)
        ])
    )

    calc_df = (
        new_df
        .explode(col)
    )

    return calc_df

# Dictionary based on the History Column
def build_history_feature_dict_batch(row):
    features = {}
    if row:  # Check if row is not None or NaN
        for nid in row.split(' '):
            features[f'history_{nid}'] = 1
    return str(features)
```

```
#%% md
### Тренувальні дані
#%%
def data_preprocess_beh(df):
    df = df.with_columns(
        pl.col("Time").str.replace(r" (\d):", r" 0$1:",
literal=False).alias('Time')
    )

    df = df.with_columns(
        pl.col("Time").str.replace(r"(\d)/(\d){1}/", r"$1/0$2/",
literal=False).alias('Time')
    )

    df = df.with_columns(
        pl.col('Time').str.strptime(
            pl.Datetime,
            format="%m/%d/%Y %I:%M:%S %p"
        ).alias('Time')
    )
    return df


beh_train = pl.read_csv('/kaggle/input/mind-large-train/behaviors.tsv',
                        separator='\t',
                        row_index_offset=0,
                        new_columns=['Impression ID',
                                     'User ID',
                                     'Time',
                                     'History',
                                     'Impression'])

beh_train = data_preprocess_beh(beh_train)
beh_train
#%%
#Preprocess News Function
def data_preprocess_news(df):
    df = df.with_columns(
        pl.col('Title Entities').map_batches(
            lambda x: x.str.replace_many(["[", "]"], '')).alias('Title
Entities')
    )

    df = df.with_columns(
        pl.col('Abstract Entities').map_batches(
            lambda x: x.str.replace_many(["[", "]"], '')).alias('Abstract
Entities')
    )

    df = df.with_columns(
        pl.col('News ID').str.replace_many(['"'], ''),
        pl.col('Category').str.replace_many(['"'], ''),
        pl.col('Sub-Category').str.replace_many(['"'], ''),
        pl.col('Title').str.replace_many(['"'], ''),
        pl.col('Abstract').str.replace_many(['"'], '')
    )
    return df

news_train = pl.read_csv('/kaggle/input/mind-large-train/news.tsv',
                         separator='\t',
                         row_index_offset=0,
                         new_columns=['News ID',
```

```
                                        'Category',
                                        'Sub-Category',
                                        'Title',
                                        'Abstract',
                                        'URL',
                                        'Title Entities',
                                        'Abstract Entities'],
                            encoding="utf8-lossy",
                            quote_char=None)

news_train = data_preprocess_news(news_train)

news_train
beh_test = pl.read_csv('/kaggle/input/mind-large-test/behaviors.tsv',
                       separator='\t',
                       row_index_offset=0,
                       new_columns=['Impression ID',
                                    'User ID',
                                    'Time',
                                    'History',
                                    'Impression'])

beh_test = data_preprocess_beh(beh_test)
beh_test
#%%
news_test = pl.read_csv('/kaggle/input/mind-large-train/news.tsv',
                        separator='\t',
                        row_index_offset=0,
                        new_columns=['News ID',
                                     'Category',
                                     'Sub-Category',
                                     'Title',
                                     'Abstract',
                                     'URL',
                                     'Title Entities',
                                     'Abstract Entities'],
                        encoding="utf8-lossy",
                        quote_char=None)

news_test = data_preprocess_news(news_test)
news_train = pl.concat([news_train, news_test]).unique()
news_train

#Label Encoding of column
def encode_news_column(df, column, encoder = None, return_encoder = False):
    if encoder is None:
        data = pl.Series(df.select(column)).to_list()
        encoder = LabelEncoder().fit(data)

        df = df.with_columns(
            pl.col(column).map_batches(lambda x: encoder.transform(x))
        )
        if return_encoder:
            return df, encoder
        else:
            return df
    else:
        df = df.with_columns(
            pl.col(column).map_batches(lambda x: encoder.transform(x))
        )
        return df

news_train = encode_news_column(news_train, 'Category')
```

```
news_train
#%% md
### Preparing Test & Train Behaviours Data for FM
#%%
def form_hist_json(beh, sample_frac):
    beh = beh.with_columns(
        pl.col('Time').dt.date().alias('Day')
    )
    beh_sample = take_sample_date_respectively(beh, 'Day', sample_frac)
    beh_sample = beh_sample.with_columns(
        pl.col('History').map_elements(build_history_feature_dict_batch,
return_dtype=pl.Utf8).alias('Hist_json')
    )
    return beh_sample

def unnest_imp(beh_sample):
    beh_explode = explode_list_column(beh_sample, 'Impression')

    beh_explode = beh_explode.with_columns(
        pl.col('Impression').str.splitn('-', 2)
        .struct.rename_fields(['Impression News ID', 'Impression'])
    ).unnest('Impression')
    return beh_explode

def forming_json_cols(beh_explode):
    beh_explode = beh_explode.rename({'Impression News ID': 'News ID'})
    beh_explode = beh_explode.join(news_train, on='News ID')
    beh_explode = beh_explode.select('User ID', 'Time', 'History', 'News ID',
'Impression', 'Category',
                                                    'Sub-Category', 'Hist_json',
'Day')
    beh_explode = beh_explode.rename({'News ID': 'Impression News ID'})
    beh_explode = beh_explode.drop_nans()

    features = beh_explode.with_columns(
        pl.col('Hist_json').str.replace_many(['{', '}', "'"], ['', '', '"']),
        ('"candidate_' + pl.col('Impression News ID').cast(pl.Utf8) + '":
1').alias('Imp_json'),
        ('"Impression": ' + pl.col('Impression')).alias('Impression_Label'),
        ('"cat_' + pl.col('Category').cast(pl.Utf8) + '": 1').alias('Cat_json'),
        ('"sub-cat_' + pl.col('Sub-Category').cast(pl.Utf8) + '":
1').alias('Sub_Cat_json'),
        ('"time_' + pl.col('Day').cast(pl.Utf8) + '": 1').alias('Day_json')
    )
    features = features.drop_nulls()
    return features

def forming_final_dict(features):
    features = features.with_columns(
        pl.concat_str(
            [
                pl.col('Imp_json'),
                pl.col('Cat_json'),
                pl.col('Sub_Cat_json'),
                pl.col('Impression_Label'),
                pl.col("Hist_json"),
                pl.col('Day_json')
            ],
            separator=', '
        ).alias('JSON')
    )

    features = features.with_columns(pl.lit('{').alias('{'),
```

```
                                        pl.lit('}').alias('}'))

    features = features.with_columns(
        pl.concat_str([pl.col('{'), pl.col('JSON'), pl.col('}')] , separator =
'').alias('NEW_JSON')
    )
    features = features.drop_nulls()
    features = features['NEW_JSON'].to_list()

    return features

def form_dictionary(series):
    batch_size = 2_000_000

    total = len(series)
    train_dict = []

    for batch_index, start in enumerate(range(0, total, batch_size), start=1):
        end = min(start + batch_size, total)

        batch = series[start:end]

        parsed_batch = [json.loads(s) for s in batch]

        process = psutil.Process()
        memory_usage_mb = process.memory_info().rss / 1e6

        train_dict.extend(parsed_batch)

        del parsed_batch, batch
        gc.collect()

    return train_dict

def dict_vectorizer(dictionary):
    dictionary = dictionary[:max_dict_len]
    dict_vect = DictVectorizer(dtype = np.int8)
    dictionary = dict_vect.fit_transform(dictionary)
    df = pd.DataFrame(dictionary.toarray(), columns =
dict_vect.get_feature_names_out())
    return df
#%%
beh_model = pl.concat([beh_train, beh_test])
beh_model = data_preproces_to_dict_beh(beh_model, 0.05)
beh_model = form_dictionary(beh_model)
beh_model = dict_vectorizer(beh_model)
print('Df for FM Created')
print(f'All Data: 4310935')
#%%
beh_model_train = beh_model[beh_model['time_2019-11-15'] == 0]
beh_model_test = beh_model[beh_model['time_2019-11-15'] == 1]
#%%
beh_model_train.to_csv('beh_model_train.csv')
beh_model_test.to_csv('beh_model_test.csv')
#%%
ffm_model = xl.create_ffm()
ffm_model.setTrain('train.csv')
ffm_model.setValidate('validate.csv')

param = {'task':'binary', 'lr':0.2,
         'lambda':0.002, 'metric':'auc', 'epoch':20}

ffm_model.fit(param, 'model.out')
```

```
#%%
ffm_model.setTest("beh_model_test.csv")
ffm_model.predict("model.out", "output.csv")

predict = pd.read_csv("output.csv")
y_test = beh_model_test.to_pandas()['Impression']

fpr, tpr, thresholds = roc_curve(y_test, predict)
auc_score = roc_auc_score(y_test, predict)

plt.figure(figsize=(7, 6))
plt.plot(fpr, tpr, label=f"FFM Model (AUC = {auc_score:.3f})",
color='darkorange', linewidth=2)
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - FFM on MIND")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()

print(f"AUC Score: {auc_score:.3f}")
```

Файл __init__.py:

```
import torch
from model.DKN.KCNN import KCNN
from model.DKN.attention import Attention
from model.general.click_predictor.DNN import DNNClickPredictor


class DKN(torch.nn.Module):
    """
    Deep knowledge-aware network.
    Input 1 + K candidate news and a list of user clicked news, produce the
click probability.
    """
    def __init__(self,
                 config,
                 pretrained_word_embedding=None,
                 pretrained_entity_embedding=None,
                 pretrained_context_embedding=None):
        super(DKN, self).__init__()
        self.config = config
        self.kcnn = KCNN(config, pretrained_word_embedding,
                         pretrained_entity_embedding,
                         pretrained_context_embedding)
        self.attention = Attention(config)
        self.click_predictor = DNNClickPredictor(
            len(self.config.window_sizes) * 2 * self.config.num_filters)

    def forward(self, candidate_news, clicked_news):
        """
        Args:
            candidate_news:
                [
                    {
                        "title": batch_size * num_words_title,
                        "title_entities": batch_size * num_words_title
                    } * (1 + K)
                ]
            clicked_news:
                [
```

```
                    {
                        "title": batch_size * num_words_title,
                        "title_entities": batch_size * num_words_title
                    } * num_clicked_news_a_user
                ]
        Returns:
            click_probability: batch_size
        """
        # batch_size, 1 + K, len(window_sizes) * num_filters
        candidate_news_vector = torch.stack(
            [self.kcnn(x) for x in candidate_news], dim=1)
        # batch_size, num_clicked_news_a_user, len(window_sizes) * num_filters
        clicked_news_vector = torch.stack([self.kcnn(x) for x in clicked_news],
                                          dim=1)
        # batch_size, 1 + K, len(window_sizes) * num_filters
        user_vector = torch.stack([
            self.attention(x, clicked_news_vector)
            for x in candidate_news_vector.transpose(0, 1)
        ],
                                  dim=1)
        size = candidate_news_vector.size()
        # batch_size, 1 + K
        click_probability = self.click_predictor(
            candidate_news_vector.view(size[0] * size[1], size[2]),
            user_vector.view(size[0] * size[1],
                             size[2])).view(size[0], size[1])
        return click_probability

    def get_news_vector(self, news):
        """
        Args:
            news:
                {
                    "title": batch_size * num_words_title,
                    "title_entities": batch_size * num_words_title
                }
        Returns:
            (shape) batch_size, len(window_sizes) * num_filters
        """
        # batch_size, len(window_sizes) * num_filters
        return self.kcnn(news)

    def get_user_vector(self, clicked_news_vector):
        """
        Args:
            clicked_news_vector: batch_size, num_clicked_news_a_user,
len(window_sizes) * num_filters
        Returns:
            (shape) batch_size, num_clicked_news_a_user, len(window_sizes) *
num_filters
        """
        # batch_size, num_clicked_news_a_user, len(window_sizes) * num_filters
        return clicked_news_vector

    def get_prediction(self, candidate_news_vector, clicked_news_vector):
        """
        Args:
            candidate_news_vector: candidate_size, len(window_sizes) *
num_filters
            clicked_news_vector: num_clicked_news_a_user, len(window_sizes) *
num_filters
        Returns:
            click_probability: 0-dim tensor
```

```
        """
        # candidate_size, len(window_sizes) * num_filters
        user_vector = self.attention(candidate_news_vector,

clicked_news_vector.expand(candidate_news_vector.size(0), -1, -1))
        # candidate_size
        click_probability = self.click_predictor(candidate_news_vector,
                                                 user_vector)
        return click_probability
```

Файл attenttion.py:

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class Attention(torch.nn.Module):
    """
    Attention Net.
    Input embedding vectors (produced by KCNN) of a candidate news and all of
user's clicked news,
    produce final user embedding vectors with respect to the candidate news.
    """
    def __init__(self, config):
        super(Attention, self).__init__()
        self.config = config
        self.dnn = nn.Sequential(
            nn.Linear(
                len(self.config.window_sizes) * 2 * self.config.num_filters,
                16), nn.Linear(16, 1))

    def forward(self, candidate_news_vector, clicked_news_vector):
        """
        Args:
            candidate_news_vector: batch_size, len(window_sizes) * num_filters
            clicked_news_vector: batch_size, num_clicked_news_a_user,
len(window_sizes) * num_filters
        Returns:
            user_vector: batch_size, len(window_sizes) * num_filters
        """
        # batch_size, num_clicked_news_a_user
        clicked_news_weights = F.softmax(self.dnn(
            torch.cat((candidate_news_vector.expand(
                self.config.num_clicked_news_a_user, -1, -1).transpose(
                    0, 1), clicked_news_vector),
                      dim=2)).squeeze(dim=2),
                                         dim=1)

        # batch_size, len(window_sizes) * num_filters
        user_vector = torch.bmm(clicked_news_weights.unsqueeze(dim=1),
                                clicked_news_vector).squeeze(dim=1)
        return user_vector
```

Файл KCNN.py:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from model.general.attention.additive import AdditiveAttention

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


class KCNN(torch.nn.Module):
```

```
    """
    Knowledge-aware CNN (KCNN) based on Kim CNN.
    Input a news sentence (e.g. its title), produce its embedding vector.
    """
    def __init__(self, config, pretrained_word_embedding,
                 pretrained_entity_embedding, pretrained_context_embedding):
        super(KCNN, self).__init__()
        self.config = config
        if pretrained_word_embedding is None:
            self.word_embedding = nn.Embedding(config.num_words,
                                               config.word_embedding_dim,
                                               padding_idx=0)
        else:
            self.word_embedding = nn.Embedding.from_pretrained(
                pretrained_word_embedding, freeze=False, padding_idx=0)
        if pretrained_entity_embedding is None:
            self.entity_embedding = nn.Embedding(config.num_entities,
                                                 config.entity_embedding_dim,
                                                 padding_idx=0)
        else:
            self.entity_embedding = nn.Embedding.from_pretrained(
                pretrained_entity_embedding, freeze=False, padding_idx=0)
        if config.use_context:
            if pretrained_context_embedding is None:
                self.context_embedding = nn.Embedding(
                    config.num_entities,
                    config.entity_embedding_dim,
                    padding_idx=0)
            else:
                self.context_embedding = nn.Embedding.from_pretrained(
                    pretrained_context_embedding, freeze=False, padding_idx=0)
        self.transform_matrix = nn.Parameter(
            torch.empty(self.config.entity_embedding_dim,
                        self.config.word_embedding_dim).uniform_(-0.1, 0.1))
        self.transform_bias = nn.Parameter(
            torch.empty(self.config.word_embedding_dim).uniform_(-0.1, 0.1))

        self.conv_filters = nn.ModuleDict({
            str(x): nn.Conv2d(3 if self.config.use_context else 2,
                              self.config.num_filters,
                              (x, self.config.word_embedding_dim))
            for x in self.config.window_sizes
        })
        self.additive_attention = AdditiveAttention(
            self.config.query_vector_dim, self.config.num_filters)

    def forward(self, news):
        """
        Args:
          news:
            {
                "title": batch_size * num_words_title,
                "title_entities": batch_size * num_words_title
            }

        Returns:
            final_vector: batch_size, len(window_sizes) * num_filters
        """
        # batch_size, num_words_title, word_embedding_dim
        word_vector = self.word_embedding(news["title"].to(device))
        # batch_size, num_words_title, entity_embedding_dim
        entity_vector = self.entity_embedding(
            news["title_entities"].to(device))
```

```
        if self.config.use_context:
            # batch_size, num_words_title, entity_embedding_dim
            context_vector = self.context_embedding(
                news["title_entities"].to(device))

        # batch_size, num_words_title, word_embedding_dim
        transformed_entity_vector = torch.tanh(
            torch.add(torch.matmul(entity_vector, self.transform_matrix),
                      self.transform_bias))

        if self.config.use_context:
            # batch_size, num_words_title, word_embedding_dim
            transformed_context_vector = torch.tanh(
                torch.add(torch.matmul(context_vector, self.transform_matrix),
                          self.transform_bias))

            # batch_size, 3, num_words_title, word_embedding_dim
            multi_channel_vector = torch.stack([
                word_vector, transformed_entity_vector,
                transformed_context_vector
            ],
                                               dim=1)
        else:
            # batch_size, 2, num_words_title, word_embedding_dim
            multi_channel_vector = torch.stack(
                [word_vector, transformed_entity_vector], dim=1)

        pooled_vectors = []
        for x in self.config.window_sizes:
            # batch_size, num_filters, num_words_title + 1 - x
            convoluted = self.conv_filters[str(x)](
                multi_channel_vector).squeeze(dim=3)
            # batch_size, num_filters, num_words_title + 1 - x
            activated = F.relu(convoluted)
            # batch_size, num_filters
            # Here we use a additive attention module
            # instead of pooling in the paper
            pooled = self.additive_attention(activated.transpose(1, 2))
            # pooled = activated.max(dim=-1)[0]
            # # or
            # # pooled = F.max_pool1d(activated,
activated.size(2)).squeeze(dim=2)
            pooled_vectors.append(pooled)
        # batch_size, len(window_sizes) * num_filters
        final_vector = torch.cat(pooled_vectors, dim=1)
        return final_vector
```

Файл additive.py:

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class AdditiveAttention(torch.nn.Module):
    """
    A general additive attention module.
    Originally for NAML.
    """
    def __init__(self,
                 query_vector_dim,
                 candidate_vector_dim,
                 writer=None,
                 tag=None,
```

```
                 names=None):
        super(AdditiveAttention, self).__init__()
        self.linear = nn.Linear(candidate_vector_dim, query_vector_dim)
        self.attention_query_vector = nn.Parameter(
            torch.empty(query_vector_dim).uniform_(-0.1, 0.1))
        # For tensorboard
        self.writer = writer
        self.tag = tag
        self.names = names
        self.local_step = 1

    def forward(self, candidate_vector):
        """
        Args:
            candidate_vector: batch_size, candidate_size, candidate_vector_dim
        Returns:
            (shape) batch_size, candidate_vector_dim
        """
        # batch_size, candidate_size, query_vector_dim
        temp = torch.tanh(self.linear(candidate_vector))
        # batch_size, candidate_size
        candidate_weights = F.softmax(torch.matmul(
            temp, self.attention_query_vector),
                                      dim=1)
        if self.writer is not None:
            assert candidate_weights.size(1) == len(self.names)
            if self.local_step % 10 == 0:
                self.writer.add_scalars(
                    self.tag, {
                        x: y
                        for x, y in zip(self.names,
                                        candidate_weights.mean(dim=0))
                    }, self.local_step)
            self.local_step += 1
        # batch_size, candidate_vector_dim
        target = torch.bmm(candidate_weights.unsqueeze(dim=1),
                           candidate_vector).squeeze(dim=1)
        return target
```

Файл multihead_self.py:

```
import torch
import torch.nn as nn
import numpy as np

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


class ScaledDotProductAttention(nn.Module):
    def __init__(self, d_k):
        super(ScaledDotProductAttention, self).__init__()
        self.d_k = d_k

    def forward(self, Q, K, V, attn_mask=None):
        scores = torch.matmul(Q, K.transpose(-1, -2)) / np.sqrt(self.d_k)
        scores = torch.exp(scores)
        if attn_mask is not None:
            scores = scores * attn_mask
        attn = scores / (torch.sum(scores, dim=-1, keepdim=True) + 1e-8)

        context = torch.matmul(attn, V)
        return context, attn
```

```
class MultiHeadSelfAttention(nn.Module):
    def __init__(self, d_model, num_attention_heads):
        super(MultiHeadSelfAttention, self).__init__()
        self.d_model = d_model
        self.num_attention_heads = num_attention_heads
        assert d_model % num_attention_heads == 0
        self.d_k = d_model // num_attention_heads
        self.d_v = d_model // num_attention_heads

        self.W_Q = nn.Linear(d_model, d_model)
        self.W_K = nn.Linear(d_model, d_model)
        self.W_V = nn.Linear(d_model, d_model)

        self._initialize_weights()

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight, gain=1)

    def forward(self, Q, K=None, V=None, length=None):
        if K is None:
            K = Q
        if V is None:
            V = Q
        batch_size = Q.size(0)

        q_s = self.W_Q(Q).view(batch_size, -1, self.num_attention_heads,
                               self.d_k).transpose(1, 2)
        k_s = self.W_K(K).view(batch_size, -1, self.num_attention_heads,
                               self.d_k).transpose(1, 2)
        v_s = self.W_V(V).view(batch_size, -1, self.num_attention_heads,
                               self.d_v).transpose(1, 2)

        if length is not None:
            maxlen = Q.size(1)
            attn_mask = torch.arange(maxlen).to(device).expand(
                batch_size, maxlen) < length.to(device).view(-1, 1)
            attn_mask = attn_mask.unsqueeze(1).expand(batch_size, maxlen,
                                                      maxlen)
            attn_mask = attn_mask.unsqueeze(1).repeat(1,
                                                      self.num_attention_heads,
                                                      1, 1)
        else:
            attn_mask = None

        context, attn = ScaledDotProductAttention(self.d_k)(q_s, k_s, v_s,
                                                            attn_mask)
        context = context.transpose(1, 2).contiguous().view(
            batch_size, -1, self.num_attention_heads * self.d_v)
        return context
```

Файл self.py:

```
import torch
import torch.nn.functional as F


class SelfAttention(torch.nn.Module):
    """
    A general self attention module.
    Originally for Hi-Fi Ark.
    """
    def __init__(self):
```

```
        super(SelfAttention, self).__init__()

    def forward(self, candidate_vector):
        """
        Args:
            candidate_vector: batch_size, candidate_size, candidate_vector_dim
        Returns:
            (shape) batch_size, candidate_size, candidate_vector_dim
        """
        # batch_size, candidate_size, candidate_size
        weights = F.softmax(torch.bmm(candidate_vector,
                                      candidate_vector.transpose(1, 2)),
                            dim=2)
        # batch_size, candidate_size, candidate_vector_dim
        self_attended_vector = torch.bmm(weights, candidate_vector)
        return self_attended_vector
```

Файл similiarity.py:

```
import torch
import torch.nn.functional as F


class SimilarityAttention(torch.nn.Module):
    """
    A general attention module based on similarity w.r.t. another vector
    """
    def __init__(self):
        super(SimilarityAttention, self).__init__()

    def forward(self, wrt_vector, candidate_vector):
        """
        Args:
            wrt_vector: batch_size, candidate_vector_dim
            candidate_vector: batch_size, candidate_size, candidate_vector_dim
        Returns:
            (shape) batch_size, candidate_vector_dim
        """
        # batch_size, candidate_size
        candidate_weights = F.softmax(torch.bmm(
            candidate_vector, wrt_vector.unsqueeze(dim=2)).squeeze(dim=2),
                                      dim=1)
        # batch_size, candidate_vector_dim
        target = torch.bmm(candidate_weights.unsqueeze(dim=1),
                           candidate_vector).squeeze(dim=1)
        return target
```

Файл DNN.py:

```
import torch
import torch.nn as nn
from math import sqrt


class DNNClickPredictor(torch.nn.Module):
    def __init__(self, input_size, hidden_size=None):
        super(DNNClickPredictor, self).__init__()
        if hidden_size is None:
            hidden_size = int(sqrt(input_size))
        self.dnn = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, 1),
        )
```

```
    def forward(self, candidate_news_vector, user_vector):
        """
        Args:
            candidate_news_vector: batch_size, X
            user_vector: batch_size, X
        Returns:
            (shape): batch_size
        """
        # batch_size
        return self.dnn(torch.cat((candidate_news_vector, user_vector),
                                  dim=1)).squeeze(dim=1)
```

Файл config,.py:

```
import os

model_name = os.environ['MODEL_NAME'] if 'MODEL_NAME' in os.environ else 'NRMS'
# Currently included model
assert model_name in [
    'NRMS', 'NAML', 'LSTUR', 'DKN', 'HiFiArk', 'TANR', 'Exp1'
]


class BaseConfig():
    """
    General configurations appiled to all models
    """
    num_epochs = 2
    num_batches_show_loss = 100  # Number of batchs to show loss
    # Number of batchs to check metrics on validation dataset
    num_batches_validate = 1000
    batch_size = 128
    learning_rate = 0.0001
    num_workers = 4  # Number of workers for data loading
    num_clicked_news_a_user = 50  # Number of sampled click history for each
user
    num_words_title = 20
    num_words_abstract = 50
    word_freq_threshold = 1
    entity_freq_threshold = 2
    entity_confidence_threshold = 0.5
    negative_sampling_ratio = 2  # K
    dropout_probability = 0.2
    # Modify the following by the output of `src/dataprocess.py`
    num_words = 1 + 70975
    num_categories = 1 + 274
    num_entities = 1 + 12957
    num_users = 1 + 50000
    word_embedding_dim = 300
    category_embedding_dim = 100
    # Modify the following only if you use another dataset
    entity_embedding_dim = 100
    # For additive attention
    query_vector_dim = 200


class NRMSConfig(BaseConfig):
    dataset_attributes = {"news": ['title'], "record": []}
    # For multi-head self-attention
```

```
    num_attention_heads = 15


class NAMLConfig(BaseConfig):
    dataset_attributes = {
        "news": ['category', 'subcategory', 'title', 'abstract'],
        "record": []
    }
    # For CNN
    num_filters = 300
    window_size = 3


class LSTURConfig(BaseConfig):
    dataset_attributes = {
        "news": ['category', 'subcategory', 'title'],
        "record": ['user', 'clicked_news_length']
    }
    # For CNN
    num_filters = 300
    window_size = 3
    long_short_term_method = 'ini'
    # See paper for more detail
    assert long_short_term_method in ['ini', 'con']
    masking_probability = 0.5


class DKNConfig(BaseConfig):
    dataset_attributes = {"news": ['title', 'title_entities'], "record": []}
    # For CNN
    num_filters = 50
    window_sizes = [2, 3, 4]
    use_context = False


class HiFiArkConfig(BaseConfig):
    dataset_attributes = {"news": ['title'], "record": []}
    # For CNN
    num_filters = 300
    window_size = 3
    num_pooling_heads = 5
    regularizer_loss_weight = 0.1


class TANRConfig(BaseConfig):
    dataset_attributes = {"news": ['category', 'title'], "record": []}
    # For CNN
    num_filters = 300
    window_size = 3
    topic_classification_loss_weight = 0.1


class Exp1Config(BaseConfig):
    dataset_attributes = {
        # TODO ['category', 'subcategory', 'title', 'abstract'],
        "news": ['category', 'subcategory', 'title'],
        "record": []
    }
    # For multi-head self-attention
    num_attention_heads = 15
    ensemble_factor = 1  # Not use ensemble since it's too expensive
```

Файл data_preprocess.py:

```
from config import model_name
import pandas as pd
import swifter
import json
import math
from tqdm import tqdm
from os import path
from pathlib import Path
import random
from nltk.tokenize import word_tokenize
import numpy as np
import csv
import importlib

try:
    config = getattr(importlib.import_module('config'), f"{model_name}Config")
except AttributeError:
    print(f"{model_name} not included!")
    exit()


def parse_behaviors(source, target, user2int_path):
    """
    Parse behaviors file in training set.
    Args:
        source: source behaviors file
        target: target behaviors file
        user2int_path: path for saving user2int file
    """
    print(f"Parse {source}")

    behaviors = pd.read_table(
        source,
        header=None,
        names=['impression_id', 'user', 'time', 'clicked_news', 'impressions'])
    behaviors.clicked_news.fillna(' ', inplace=True)
    behaviors.impressions = behaviors.impressions.str.split()

    user2int = {}
    for row in behaviors.itertuples(index=False):
        if row.user not in user2int:
            user2int[row.user] = len(user2int) + 1

    pd.DataFrame(user2int.items(), columns=['user',
                                            'int']).to_csv(user2int_path,
                                                           sep='\t',
                                                           index=False)
    print(
        f'Please modify `num_users` in `src/config.py` into 1 + {len(user2int)}'
    )

    for row in behaviors.itertuples():
        behaviors.at[row.Index, 'user'] = user2int[row.user]

    for row in tqdm(behaviors.itertuples(), desc="Balancing data"):
        positive = iter([x for x in row.impressions if x.endswith('1')])
        negative = [x for x in row.impressions if x.endswith('0')]
        random.shuffle(negative)
        negative = iter(negative)
        pairs = []
        try:
            while True:
                pair = [next(positive)]
```

```
                for _ in range(config.negative_sampling_ratio):
                    pair.append(next(negative))
                pairs.append(pair)
            except StopIteration:
                pass
            behaviors.at[row.Index, 'impressions'] = pairs

        behaviors = behaviors.explode('impressions').dropna(
            subset=["impressions"]).reset_index(drop=True)
        behaviors[['candidate_news', 'clicked']] = pd.DataFrame(
            behaviors.impressions.map(
                lambda x: (' '.join([e.split('-')[0] for e in x]), ' '.join(
                    [e.split('-')[1] for e in x]))).tolist())
        behaviors.to_csv(
            target,
            sep='\t',
            index=False,
            columns=['user', 'clicked_news', 'candidate_news', 'clicked'])


    def parse_news(source, target, category2int_path, word2int_path,
                   entity2int_path, mode):
        """
        Parse news for training set and test set
        Args:
            source: source news file
            target: target news file
            if mode == 'train':
                category2int_path, word2int_path, entity2int_path: Path to save
            elif mode == 'test':
                category2int_path, word2int_path, entity2int_path: Path to load from
        """
        print(f"Parse {source}")
        news = pd.read_table(source,
                             header=None,
                             usecols=[0, 1, 2, 3, 4, 6, 7],
                             quoting=csv.QUOTE_NONE,
                             names=[
                                 'id', 'category', 'subcategory', 'title',
                                 'abstract', 'title_entities', 'abstract_entities'
                             ])  # TODO try to avoid csv.QUOTE_NONE
        news.title_entities.fillna('[]', inplace=True)
        news.abstract_entities.fillna('[]', inplace=True)
        news.fillna(' ', inplace=True)

        def parse_row(row):
            new_row = [
                row.id,
                category2int[row.category] if row.category in category2int else 0,
                category2int[row.subcategory]
                if row.subcategory in category2int else 0,
                [0] * config.num_words_title, [0] * config.num_words_abstract,
                [0] * config.num_words_title, [0] * config.num_words_abstract
            ]

            # Calculate local entity map (map lower single word to entity)
            local_entity_map = {}
            for e in json.loads(row.title_entities):
                if e['Confidence'] > config.entity_confidence_threshold and e[
                        'WikidataId'] in entity2int:
                    for x in ' '.join(e['SurfaceForms']).lower().split():
                        local_entity_map[x] = entity2int[e['WikidataId']]
            for e in json.loads(row.abstract_entities):
```

```
            if e['Confidence'] > config.entity_confidence_threshold and e[
                    'WikidataId'] in entity2int:
                for x in ' '.join(e['SurfaceForms']).lower().split():
                    local_entity_map[x] = entity2int[e['WikidataId']]

        try:
            for i, w in enumerate(word_tokenize(row.title.lower())):
                if w in word2int:
                    new_row[3][i] = word2int[w]
                    if w in local_entity_map:
                        new_row[5][i] = local_entity_map[w]
        except IndexError:
            pass

        try:
            for i, w in enumerate(word_tokenize(row.abstract.lower())):
                if w in word2int:
                    new_row[4][i] = word2int[w]
                    if w in local_entity_map:
                        new_row[6][i] = local_entity_map[w]
        except IndexError:
            pass

        return pd.Series(new_row,
                         index=[
                             'id', 'category', 'subcategory', 'title',
                             'abstract', 'title_entities', 'abstract_entities'
                         ])

    if mode == 'train':
        category2int = {}
        word2int = {}
        word2freq = {}
        entity2int = {}
        entity2freq = {}

        for row in news.itertuples(index=False):
            if row.category not in category2int:
                category2int[row.category] = len(category2int) + 1
            if row.subcategory not in category2int:
                category2int[row.subcategory] = len(category2int) + 1

            for w in word_tokenize(row.title.lower()):
                if w not in word2freq:
                    word2freq[w] = 1
                else:
                    word2freq[w] += 1
            for w in word_tokenize(row.abstract.lower()):
                if w not in word2freq:
                    word2freq[w] = 1
                else:
                    word2freq[w] += 1

            for e in json.loads(row.title_entities):
                times = len(e['OccurrenceOffsets']) * e['Confidence']
                if times > 0:
                    if e['WikidataId'] not in entity2freq:
                        entity2freq[e['WikidataId']] = times
                    else:
                        entity2freq[e['WikidataId']] += times

            for e in json.loads(row.abstract_entities):
                times = len(e['OccurrenceOffsets']) * e['Confidence']
```

```
                if times > 0:
                    if e['WikidataId'] not in entity2freq:
                        entity2freq[e['WikidataId']] = times
                    else:
                        entity2freq[e['WikidataId']] += times

        for k, v in word2freq.items():
            if v >= config.word_freq_threshold:
                word2int[k] = len(word2int) + 1

        for k, v in entity2freq.items():
            if v >= config.entity_freq_threshold:
                entity2int[k] = len(entity2int) + 1

        parsed_news = news.swifter.apply(parse_row, axis=1)
        parsed_news.to_csv(target, sep='\t', index=False)

        pd.DataFrame(category2int.items(),
                     columns=['category', 'int']).to_csv(category2int_path,
                                                         sep='\t',
                                                         index=False)
        print(
            f'Please modify `num_categories` in `src/config.py` into 1 +
{len(category2int)}'
        )

        pd.DataFrame(word2int.items(), columns=['word',
                                                'int']).to_csv(word2int_path,
                                                               sep='\t',
                                                               index=False)
        print(
            f'Please modify `num_words` in `src/config.py` into 1 +
{len(word2int)}'
        )

        pd.DataFrame(entity2int.items(),
                     columns=['entity', 'int']).to_csv(entity2int_path,
                                                       sep='\t',
                                                       index=False)
        print(
            f'Please modify `num_entities` in `src/config.py` into 1 +
{len(entity2int)}'
        )

    elif mode == 'test':
        category2int = dict(pd.read_table(category2int_path).values.tolist())
        # na_filter=False is needed since nan is also a valid word
        word2int = dict(
            pd.read_table(word2int_path, na_filter=False).values.tolist())
        entity2int = dict(pd.read_table(entity2int_path).values.tolist())

        parsed_news = news.swifter.apply(parse_row, axis=1)
        parsed_news.to_csv(target, sep='\t', index=False)

    else:
        print('Wrong mode!')


def generate_word_embedding(source, target, word2int_path):
    """
    Generate from pretrained word embedding file
    If a word not in embedding file, initial its embedding by N(0, 1)
    Args:
```

```
        source: path of pretrained word embedding file, e.g. glove.840B.300d.txt
        target: path for saving word embedding. Will be saved in numpy format
        word2int_path: vocabulary file when words in it will be searched in
pretrained embedding file
    """
    # na_filter=False is needed since nan is also a valid word
    # word, int
    word2int = pd.read_table(word2int_path, na_filter=False, index_col='word')
    source_embedding = pd.read_table(source,
                                     index_col=0,
                                     sep=' ',
                                     header=None,
                                     quoting=csv.QUOTE_NONE,
                                     names=range(config.word_embedding_dim))
    # word, vector
    source_embedding.index.rename('word', inplace=True)
    # word, int, vector
    merged = word2int.merge(source_embedding,
                            how='inner',
                            left_index=True,
                            right_index=True)
    merged.set_index('int', inplace=True)

    missed_index = np.setdiff1d(np.arange(len(word2int) + 1),
                                merged.index.values)
    missed_embedding = pd.DataFrame(data=np.random.normal(
        size=(len(missed_index), config.word_embedding_dim)))
    missed_embedding['int'] = missed_index
    missed_embedding.set_index('int', inplace=True)

    final_embedding = pd.concat([merged, missed_embedding]).sort_index()
    np.save(target, final_embedding.values)

    print(
        f'Rate of word missed in pretrained embedding: {(len(missed_index)-
1)/len(word2int):.4f}'
    )


def transform_entity_embedding(source, target, entity2int_path):
    """
    Args:
        source: path of embedding file
        target: path of transformed embedding file in numpy format
        entity2int_path
    """
    entity_embedding = pd.read_table(source, header=None)
    entity_embedding['vector'] = entity_embedding.iloc[:,
                                                       1:101].values.tolist()
    entity_embedding = entity_embedding[[0, 'vector'
                                         ]].rename(columns={0: "entity"})

    entity2int = pd.read_table(entity2int_path)
    merged_df = pd.merge(entity_embedding, entity2int,
                         on='entity').sort_values('int')
    entity_embedding_transformed = np.random.normal(
        size=(len(entity2int) + 1, config.entity_embedding_dim))
    for row in merged_df.itertuples(index=False):
        entity_embedding_transformed[row.int] = row.vector
    np.save(target, entity_embedding_transformed)


if __name__ == '__main__':
```

```
    train_dir = './data/train'
    val_dir = './data/val'
    test_dir = './data/test'

    print('Process data for training')

    print('Parse behaviors')
    parse_behaviors(path.join(train_dir, 'behaviors.tsv'),
                    path.join(train_dir, 'behaviors_parsed.tsv'),
                    path.join(train_dir, 'user2int.tsv'))

    print('Parse news')
    parse_news(path.join(train_dir, 'news.tsv'),
               path.join(train_dir, 'news_parsed.tsv'),
               path.join(train_dir, 'category2int.tsv'),
               path.join(train_dir, 'word2int.tsv'),
               path.join(train_dir, 'entity2int.tsv'),
               mode='train')

    print('Generate word embedding')
    generate_word_embedding(
        f'./data/glove/glove.840B.{config.word_embedding_dim}d.txt',
        path.join(train_dir, 'pretrained_word_embedding.npy'),
        path.join(train_dir, 'word2int.tsv'))

    print('Transform entity embeddings')
    transform_entity_embedding(
        path.join(train_dir, 'entity_embedding.vec'),
        path.join(train_dir, 'pretrained_entity_embedding.npy'),
        path.join(train_dir, 'entity2int.tsv'))

    print('\nProcess data for validation')

    print('Parse news')
    parse_news(path.join(val_dir, 'news.tsv'),
               path.join(val_dir, 'news_parsed.tsv'),
               path.join(train_dir, 'category2int.tsv'),
               path.join(train_dir, 'word2int.tsv'),
               path.join(train_dir, 'entity2int.tsv'),
               mode='test')

    print('\nProcess data for test')

    print('Parse news')
    parse_news(path.join(test_dir, 'news.tsv'),
               path.join(test_dir, 'news_parsed.tsv'),
               path.join(train_dir, 'category2int.tsv'),
               path.join(train_dir, 'word2int.tsv'),
               path.join(train_dir, 'entity2int.tsv'),
               mode='test')
```

Файл dataset.py:

```
from torch.utils.data import Dataset
import pandas as pd
from ast import literal_eval
from os import path
import numpy as np
from config import model_name
import importlib
import torch

try:
    config = getattr(importlib.import_module('config'), f"{model_name}Config")
```

```
except AttributeError:
    print(f"{model_name} not included!")
    exit()


class BaseDataset(Dataset):
    def __init__(self, behaviors_path, news_path):
        super(BaseDataset, self).__init__()
        assert all(attribute in [
            'category', 'subcategory', 'title', 'abstract', 'title_entities',
            'abstract_entities'
        ] for attribute in config.dataset_attributes['news'])
        assert all(attribute in ['user', 'clicked_news_length']
                   for attribute in config.dataset_attributes['record'])

        self.behaviors_parsed = pd.read_table(behaviors_path)
        self.news_parsed = pd.read_table(
            news_path,
            index_col='id',
            usecols=['id'] + config.dataset_attributes['news'],
            converters={
                attribute: literal_eval
                for attribute in set(config.dataset_attributes['news']) & set([
                    'title', 'abstract', 'title_entities', 'abstract_entities'
                ])
            })
        self.news_id2int = {x: i for i, x in enumerate(self.news_parsed.index)}
        self.news2dict = self.news_parsed.to_dict('index')
        for key1 in self.news2dict.keys():
            for key2 in self.news2dict[key1].keys():
                self.news2dict[key1][key2] = torch.tensor(
                    self.news2dict[key1][key2])
        padding_all = {
            'category': 0,
            'subcategory': 0,
            'title': [0] * config.num_words_title,
            'abstract': [0] * config.num_words_abstract,
            'title_entities': [0] * config.num_words_title,
            'abstract_entities': [0] * config.num_words_abstract
        }
        for key in padding_all.keys():
            padding_all[key] = torch.tensor(padding_all[key])

        self.padding = {
            k: v
            for k, v in padding_all.items()
            if k in config.dataset_attributes['news']
        }

    def __len__(self):
        return len(self.behaviors_parsed)

    def __getitem__(self, idx):
        item = {}
        row = self.behaviors_parsed.iloc[idx]
        if 'user' in config.dataset_attributes['record']:
            item['user'] = row.user
        item["clicked"] = list(map(int, row.clicked.split()))
        item["candidate_news"] = [
            self.news2dict[x] for x in row.candidate_news.split()
        ]
        item["clicked_news"] = [
            self.news2dict[x]
```

```
            for x in row.clicked_news.split()[:config.num_clicked_news_a_user]
        ]
        if 'clicked_news_length' in config.dataset_attributes['record']:
            item['clicked_news_length'] = len(item["clicked_news"])
        repeated_times = config.num_clicked_news_a_user - \
            len(item["clicked_news"])
        assert repeated_times >= 0
        item["clicked_news"] = [self.padding
                                ] * repeated_times + item["clicked_news"]

        return item
```

Файл evaluate.py:

```
import numpy as np
from sklearn.metrics import roc_auc_score
from tqdm import tqdm
import torch
from config import model_name
from torch.utils.data import Dataset, DataLoader
from os import path
import sys
import pandas as pd
from ast import literal_eval
import importlib
from multiprocessing import Pool

try:
    Model = getattr(importlib.import_module(f"model.{model_name}"), model_name)
    config = getattr(importlib.import_module('config'), f"{model_name}Config")
except AttributeError:
    print(f"{model_name} not included!")
    exit()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


def dcg_score(y_true, y_score, k=10):
    order = np.argsort(y_score)[::-1]
    y_true = np.take(y_true, order[:k])
    gains = 2**y_true - 1
    discounts = np.log2(np.arange(len(y_true)) + 2)
    return np.sum(gains / discounts)


def ndcg_score(y_true, y_score, k=10):
    best = dcg_score(y_true, y_true, k)
    actual = dcg_score(y_true, y_score, k)
    return actual / best


def mrr_score(y_true, y_score):
    order = np.argsort(y_score)[::-1]
    y_true = np.take(y_true, order)
    rr_score = y_true / (np.arange(len(y_true)) + 1)
    return np.sum(rr_score) / np.sum(y_true)


def value2rank(d):
    values = list(d.values())
    ranks = [sorted(values, reverse=True).index(x) for x in values]
    return {k: ranks[i] + 1 for i, k in enumerate(d.keys())}
```

```
class NewsDataset(Dataset):
    """
    Load news for evaluation.
    """
    def __init__(self, news_path):
        super(NewsDataset, self).__init__()
        self.news_parsed = pd.read_table(
            news_path,
            usecols=['id'] + config.dataset_attributes['news'],
            converters={
                attribute: literal_eval
                for attribute in set(config.dataset_attributes['news']) & set([
                    'title', 'abstract', 'title_entities', 'abstract_entities'
                ])
            })
        self.news2dict = self.news_parsed.to_dict('index')
        for key1 in self.news2dict.keys():
            for key2 in self.news2dict[key1].keys():
                if type(self.news2dict[key1][key2]) != str:
                    self.news2dict[key1][key2] = torch.tensor(
                        self.news2dict[key1][key2])

    def __len__(self):
        return len(self.news_parsed)

    def __getitem__(self, idx):
        item = self.news2dict[idx]
        return item


class UserDataset(Dataset):
    """
    Load users for evaluation, duplicated rows will be dropped
    """
    def __init__(self, behaviors_path, user2int_path):
        super(UserDataset, self).__init__()
        self.behaviors = pd.read_table(behaviors_path,
                                       header=None,
                                       usecols=[1, 3],
                                       names=['user', 'clicked_news'])
        self.behaviors.clicked_news.fillna(' ', inplace=True)
        self.behaviors.drop_duplicates(inplace=True)
        user2int = dict(pd.read_table(user2int_path).values.tolist())
        user_total = 0
        user_missed = 0
        for row in self.behaviors.itertuples():
            user_total += 1
            if row.user in user2int:
                self.behaviors.at[row.Index, 'user'] = user2int[row.user]
            else:
                user_missed += 1
                self.behaviors.at[row.Index, 'user'] = 0
        if model_name == 'LSTUR':
            print(f'User miss rate: {user_missed/user_total:.4f}')

    def __len__(self):
        return len(self.behaviors)

    def __getitem__(self, idx):
        row = self.behaviors.iloc[idx]
        item = {
            "user":
            row.user,
```

```
            "clicked_news_string":
            row.clicked_news,
            "clicked_news":
            row.clicked_news.split()[:config.num_clicked_news_a_user]
        }
        item['clicked_news_length'] = len(item["clicked_news"])
        repeated_times = config.num_clicked_news_a_user - len(
            item["clicked_news"])
        assert repeated_times >= 0
        item["clicked_news"] = ['PADDED_NEWS'
                                ] * repeated_times + item["clicked_news"]

        return item


class BehaviorsDataset(Dataset):
    """
    Load behaviors for evaluation, (user, time) pair as session
    """
    def __init__(self, behaviors_path):
        super(BehaviorsDataset, self).__init__()
        self.behaviors = pd.read_table(behaviors_path,
                                       header=None,
                                       usecols=range(5),
                                       names=[
                                           'impression_id', 'user', 'time',
                                           'clicked_news', 'impressions'
                                       ])
        self.behaviors.clicked_news.fillna(' ', inplace=True)
        self.behaviors.impressions = self.behaviors.impressions.str.split()

    def __len__(self):
        return len(self.behaviors)

    def __getitem__(self, idx):
        row = self.behaviors.iloc[idx]
        item = {
            "impression_id": row.impression_id,
            "user": row.user,
            "time": row.time,
            "clicked_news_string": row.clicked_news,
            "impressions": row.impressions
        }
        return item


def calculate_single_user_metric(pair):
    try:
        auc = roc_auc_score(*pair)
        mrr = mrr_score(*pair)
        ndcg5 = ndcg_score(*pair, 5)
        ndcg10 = ndcg_score(*pair, 10)
        return [auc, mrr, ndcg5, ndcg10]
    except ValueError:
        return [np.nan] * 4


@torch.no_grad()
def evaluate(model, directory, num_workers, max_count=sys.maxsize):
    """
    Evaluate model on target directory.
    Args:
        model: model to be evaluated
```

```
        directory: the directory that contains two files (behaviors.tsv,
news_parsed.tsv)
        num_workers: processes number for calculating metrics
    Returns:
        AUC
        MRR
        nDCG@5
        nDCG@10
    """
    news_dataset = NewsDataset(path.join(directory, 'news_parsed.tsv'))
    news_dataloader = DataLoader(news_dataset,
                                 batch_size=config.batch_size * 16,
                                 shuffle=False,
                                 num_workers=config.num_workers,
                                 drop_last=False,
                                 pin_memory=True)

    news2vector = {}
    for minibatch in tqdm(news_dataloader,
                          desc="Calculating vectors for news"):
        news_ids = minibatch["id"]
        if any(id not in news2vector for id in news_ids):
            news_vector = model.get_news_vector(minibatch)
            for id, vector in zip(news_ids, news_vector):
                if id not in news2vector:
                    news2vector[id] = vector

    news2vector['PADDED_NEWS'] = torch.zeros(
        list(news2vector.values())[0].size())

    user_dataset = UserDataset(path.join(directory, 'behaviors.tsv'),
                               'data/train/user2int.tsv')
    user_dataloader = DataLoader(user_dataset,
                                 batch_size=config.batch_size * 16,
                                 shuffle=False,
                                 num_workers=config.num_workers,
                                 drop_last=False,
                                 pin_memory=True)

    user2vector = {}
    for minibatch in tqdm(user_dataloader,
                          desc="Calculating vectors for users"):
        user_strings = minibatch["clicked_news_string"]
        if any(user_string not in user2vector for user_string in user_strings):
            clicked_news_vector = torch.stack([
                torch.stack([news2vector[x].to(device) for x in news_list],
                            dim=0) for news_list in minibatch["clicked_news"]
            ],
                                              dim=0).transpose(0, 1)
            if model_name == 'LSTUR':
                user_vector = model.get_user_vector(
                    minibatch['user'], minibatch['clicked_news_length'],
                    clicked_news_vector)
            else:
                user_vector = model.get_user_vector(clicked_news_vector)
            for user, vector in zip(user_strings, user_vector):
                if user not in user2vector:
                    user2vector[user] = vector

    behaviors_dataset = BehaviorsDataset(path.join(directory, 'behaviors.tsv'))
    behaviors_dataloader = DataLoader(behaviors_dataset,
                                      batch_size=1,
                                      shuffle=False,
```

```
                                        num_workers=config.num_workers)

    count = 0

    tasks = []

    for minibatch in tqdm(behaviors_dataloader,
                          desc="Calculating probabilities"):
        count += 1
        if count == max_count:
            break

        candidate_news_vector = torch.stack([
            news2vector[news[0].split('-')[0]]
            for news in minibatch['impressions']
        ],
                                            dim=0)
        user_vector = user2vector[minibatch['clicked_news_string'][0]]
        click_probability = model.get_prediction(candidate_news_vector,
                                                 user_vector)

        y_pred = click_probability.tolist()
        y_true = [
            int(news[0].split('-')[1]) for news in minibatch['impressions']
        ]

        tasks.append((y_true, y_pred))

    with Pool(processes=num_workers) as pool:
        results = pool.map(calculate_single_user_metric, tasks)

    aucs, mrrs, ndcg5s, ndcg10s = np.array(results).T
    return np.nanmean(aucs), np.nanmean(mrrs), np.nanmean(ndcg5s), np.nanmean(
        ndcg10s)


if __name__ == '__main__':
    print('Using device:', device)
    print(f'Evaluating model {model_name}')
    # Don't need to load pretrained word/entity/context embedding
    # since it will be loaded from checkpoint later
    model = Model(config).to(device)
    from train import latest_checkpoint  # Avoid circular imports
    checkpoint_path = latest_checkpoint(path.join('./checkpoint', model_name))
    if checkpoint_path is None:
        print('No checkpoint file found!')
        exit()
    print(f"Load saved parameters in {checkpoint_path}")
    checkpoint = torch.load(checkpoint_path)
    model.load_state_dict(checkpoint['model_state_dict'])
    model.eval()
    auc, mrr, ndcg5, ndcg10 = evaluate(model, './data/test',
                                       config.num_workers)
    print(
        f'AUC: {auc:.4f}\nMRR: {mrr:.4f}\nnDCG@5: {ndcg5:.4f}\nnDCG@10:
{ndcg10:.4f}'
    )
```

Файл train,py:

```
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
from dataset import BaseDataset
import torch
```

```
import torch.nn as nn
import torch.nn.functional as F
import time
import numpy as np
from config import model_name
from tqdm import tqdm
import os
from pathlib import Path
from evaluate import evaluate
import importlib
import datetime

try:
    Model = getattr(importlib.import_module(f"model.{model_name}"), model_name)
    config = getattr(importlib.import_module('config'), f"{model_name}Config")
except AttributeError:
    print(f"{model_name} not included!")
    exit()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


class EarlyStopping:
    def __init__(self, patience=5):
        self.patience = patience
        self.counter = 0
        self.best_loss = np.Inf

    def __call__(self, val_loss):
        """
        if you use other metrics where a higher value is better, e.g. accuracy,
        call this with its corresponding negative value
        """
        if val_loss < self.best_loss:
            early_stop = False
            get_better = True
            self.counter = 0
            self.best_loss = val_loss
        else:
            get_better = False
            self.counter += 1
            if self.counter >= self.patience:
                early_stop = True
            else:
                early_stop = False

        return early_stop, get_better


def latest_checkpoint(directory):
    if not os.path.exists(directory):
        return None
    all_checkpoints = {
        int(x.split('.')[-2].split('-')[-1]): x
        for x in os.listdir(directory)
    }
    if not all_checkpoints:
        return None
    return os.path.join(directory,
                        all_checkpoints[max(all_checkpoints.keys())])


def train():
```

```
    writer = SummaryWriter(
        log_dir=

f"./runs/{model_name}/{datetime.datetime.now().replace(microsecond=0).isoformat(
)}{'-' + os.environ['REMARK'] if 'REMARK' in os.environ else ''}"
    )

    if not os.path.exists('checkpoint'):
        os.makedirs('checkpoint')

    try:
        pretrained_word_embedding = torch.from_numpy(
            np.load('./data/train/pretrained_word_embedding.npy')).float()
    except FileNotFoundError:
        pretrained_word_embedding = None

    if model_name == 'DKN':
        try:
            pretrained_entity_embedding = torch.from_numpy(
                np.load(
                    './data/train/pretrained_entity_embedding.npy')).float()
        except FileNotFoundError:
            pretrained_entity_embedding = None

        try:
            pretrained_context_embedding = torch.from_numpy(
                np.load(
                    './data/train/pretrained_context_embedding.npy')).float()
        except FileNotFoundError:
            pretrained_context_embedding = None

        model = Model(config, pretrained_word_embedding,
                      pretrained_entity_embedding,
                      pretrained_context_embedding).to(device)
    elif model_name == 'Exp1':
        models = nn.ModuleList([
            Model(config, pretrained_word_embedding).to(device)
            for _ in range(config.ensemble_factor)
        ])
    else:
        model = Model(config, pretrained_word_embedding).to(device)

    if model_name != 'Exp1':
        print(model)
    else:
        print(models[0])

    dataset = BaseDataset('data/train/behaviors_parsed.tsv',
                          'data/train/news_parsed.tsv')

    print(f"Load training dataset with size {len(dataset)}.")

    dataloader = iter(
        DataLoader(dataset,
                   batch_size=config.batch_size,
                   shuffle=True,
                   num_workers=config.num_workers,
                   drop_last=True,
                   pin_memory=True))
    if model_name != 'Exp1':
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(),
                                     lr=config.learning_rate)
```

```
    else:
        criterion = nn.NLLLoss()
        optimizers = [
            torch.optim.Adam(model.parameters(), lr=config.learning_rate)
            for model in models
        ]
    start_time = time.time()
    loss_full = []
    exhaustion_count = 0
    step = 0
    early_stopping = EarlyStopping()

    checkpoint_dir = os.path.join('./checkpoint', model_name)
    Path(checkpoint_dir).mkdir(parents=True, exist_ok=True)

    checkpoint_path = latest_checkpoint(checkpoint_dir)
    if checkpoint_path is not None:
        print(f"Load saved parameters in {checkpoint_path}")
        checkpoint = torch.load(checkpoint_path)
        early_stopping(checkpoint['early_stop_value'])
        step = checkpoint['step']
        if model_name != 'Exp1':
            model.load_state_dict(checkpoint['model_state_dict'])
            optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
            model.train()
        else:
            for model in models:
                model.load_state_dict(checkpoint['model_state_dict'])
                model.train()
            for optimizer in optimizers:
                optimizer.load_state_dict(checkpoint['optimizer_state_dict'])

    for i in tqdm(range(
            1,
            config.num_epochs * len(dataset) // config.batch_size + 1),
                  desc="Training"):
        try:
            minibatch = next(dataloader)
        except StopIteration:
            exhaustion_count += 1
            tqdm.write(
                f"Training data exhausted for {exhaustion_count} times after {i}
batches, reuse the dataset."
            )
            dataloader = iter(
                DataLoader(dataset,
                           batch_size=config.batch_size,
                           shuffle=True,
                           num_workers=config.num_workers,
                           drop_last=True,
                           pin_memory=True))
            minibatch = next(dataloader)

        step += 1
        if model_name == 'LSTUR':
            y_pred = model(minibatch["user"], minibatch["clicked_news_length"],
                           minibatch["candidate_news"],
                           minibatch["clicked_news"])
        elif model_name == 'HiFiArk':
            y_pred, regularizer_loss = model(minibatch["candidate_news"],
                                             minibatch["clicked_news"])
        elif model_name == 'TANR':
            y_pred, topic_classification_loss = model(
```

```
                minibatch["candidate_news"], minibatch["clicked_news"])
        elif model_name == 'Exp1':
            y_preds = [
                model(minibatch["candidate_news"], minibatch["clicked_news"])
                for model in models
            ]
            y_pred_averaged = torch.stack(
                [F.softmax(y_pred, dim=1) for y_pred in y_preds],
                dim=-1).mean(dim=-1)
            y_pred = torch.log(y_pred_averaged)
        else:
            y_pred = model(minibatch["candidate_news"],
                           minibatch["clicked_news"])

        y = torch.zeros(len(y_pred)).long().to(device)
        loss = criterion(y_pred, y)

        if model_name == 'HiFiArk':
            if i % 10 == 0:
                writer.add_scalar('Train/BaseLoss', loss.item(), step)
                writer.add_scalar('Train/RegularizerLoss',
                                  regularizer_loss.item(), step)
                writer.add_scalar('Train/RegularizerBaseRatio',
                                  regularizer_loss.item() / loss.item(), step)
            loss += config.regularizer_loss_weight * regularizer_loss
        elif model_name == 'TANR':
            if i % 10 == 0:
                writer.add_scalar('Train/BaseLoss', loss.item(), step)
                writer.add_scalar('Train/TopicClassificationLoss',
                                  topic_classification_loss.item(), step)
                writer.add_scalar(
                    'Train/TopicBaseRatio',
                    topic_classification_loss.item() / loss.item(), step)
            loss += config.topic_classification_loss_weight *
topic_classification_loss
        loss_full.append(loss.item())
        if model_name != 'Exp1':
            optimizer.zero_grad()
        else:
            for optimizer in optimizers:
                optimizer.zero_grad()
        loss.backward()
        if model_name != 'Exp1':
            optimizer.step()
        else:
            for optimizer in optimizers:
                optimizer.step()

        if i % 10 == 0:
            writer.add_scalar('Train/Loss', loss.item(), step)

        if i % config.num_batches_show_loss == 0:
            tqdm.write(
                f"Time {time_since(start_time)}, batches {i}, current loss
{loss.item():.4f}, average loss: {np.mean(loss_full):.4f}, latest average loss:
{np.mean(loss_full[-256:]):.4f}"
            )

        if i % config.num_batches_validate == 0:
            (model if model_name != 'Exp1' else models[0]).eval()
            val_auc, val_mrr, val_ndcg5, val_ndcg10 = evaluate(
                model if model_name != 'Exp1' else models[0], './data/val',
                config.num_workers, 200000)
```

```
            (model if model_name != 'Exp1' else models[0]).train()
            writer.add_scalar('Validation/AUC', val_auc, step)
            writer.add_scalar('Validation/MRR', val_mrr, step)
            writer.add_scalar('Validation/nDCG@5', val_ndcg5, step)
            writer.add_scalar('Validation/nDCG@10', val_ndcg10, step)
            tqdm.write(
                f"Time {time_since(start_time)}, batches {i}, validation AUC:
{val_auc:.4f}, validation MRR: {val_mrr:.4f}, validation nDCG@5:
{val_ndcg5:.4f}, validation nDCG@10: {val_ndcg10:.4f}, "
            )

            early_stop, get_better = early_stopping(-val_auc)
            if early_stop:
                tqdm.write('Early stop.')
                break
            elif get_better:
                try:
                    torch.save(
                        {
                            'model_state_dict': (model if model_name != 'Exp1'
                                                 else models[0]).state_dict(),
                            'optimizer_state_dict':
                            (optimizer if model_name != 'Exp1' else
                             optimizers[0]).state_dict(),
                            'step':
                            step,
                            'early_stop_value':
                            -val_auc
                        }, f"./checkpoint/{model_name}/ckpt-{step}.pth")
                except OSError as error:
                    print(f"OS error: {error}")


def time_since(since):
    """
    Format elapsed time string.
    """
    now = time.time()
    elapsed_time = now - since
    return time.strftime("%H:%M:%S", time.gmtime(elapsed_time))


if __name__ == '__main__':
    print('Using device:', device)
    print(f'Training model {model_name}')
    train()
```