МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РЕАЛІЗАЦІЯ МЕТОДУ МУЛЬТИПЛІКАТИВНИХ ІТЕРАЦІЙ ДЛЯ ВИКОНАННЯ НЕВІД'ЄМНОЇ СИМЕТРИЧНОЇ МАТРИЧНОЇ ФАКТОРИЗАЦІЇ»**

Виконав: студент 4 курсу, групи 6.1229
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
Д. Ю. Камінський
(ініціали та прізвище)
Керівник старший викладач кафедри комп'ютерних наук, к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Чопоров С.В.
(підпис)

" ____ " _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Камінському Дмитру Юрійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Реалізація методу мультиплікативних ітерацій для виконання невід'ємної симетричної матричної факторизації

керівник роботи Добровольський Геннадій Анатолійович, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд методів кластеризації.
2. Факторизація.
3. Програмна реалізація метода.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.		виконано
2.	Збір вихідних даних.		виконано
3.	Обробка методичних та теоретичних		виконано
	джерел.		
4.	Розробка першого та другого розділу.		виконано
5.	Розробка третього розділу.		виконано
6.	Оформлення та нормоконтроль		виконано
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.		

Студент _____ Д. Ю. Камінський
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г. А. Добровольський
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Реалізація методу мультиплікативних ітерацій для виконання невід'ємної симетричної матричної факторизації»: 41 с., 1 рис., 2 табл., 24 джерел.

ВИЯВЛЕННЯ СПІЛЬНОТ, КЛАСТЕРНИЙ АНАЛІЗ, КЛАСТЕРИЗАЦІЯ, МАТРИЧНА ФАКТОРИЗАЦІЯ, МЕТОДИ МАТРИЧНОЇ ФАКТОРИЗАЦІЇ, МЕТОД ГОЛОВНИХ КОМПОНЕНТ, НЕВІД'ЄМНА МАТРИЧНА ФАКТОРИЗАЦІЯ.

Об’єкт дослідження – методи матричної факторизації.

Мета роботи: реалізація мультиплікативних ітерацій методами матричної алгебри.

Для того щоб можна було виконувати кластеризацію в неметричних просторах з невідомою кількістю кластерів.

Метод дослідження – аналітичний, порівняльний.

Однією із типових задач дослідження даних є кластеризація. У загальному випадку важко встановити спосіб розподілу та кількість кластерів. А більшість відомих методів потребує кількість кластерів в якості вхідного параметру. Тому важливими є методи кластеризації, які самостійно визначають необхідну кількість кластерів. Одним із відомих методів є симетрична розріджена невід’ємна матрична факторизація. Метою роботи є реалізація мультиплікативних ітерацій методами матричної алгебри, що дозволить прискорити процес кластеризації порівняно із наївним підходом.

SUMMARY

Bachelor’s qualifying theses «Implementation of Multiplicative Iterations for the Symmetric Nonnegative Matrix Factorization»: 41 pages, 1 figure, 2 tables, 24 references.

COMMUNITY DETECTION, CLUSTER ANALYSIS, CLUSTERIZATION, MATRIX FACTORIZATION, METHODS OF MATRIX FACTORIZATION, PRINCIPAL COMPONENT ANALYSIS, NON-NEGATIVE MATRIX FACTORIZATION.

Object of the study – methods of matrix factorization.

Aim of the study: implementation of multiplicative iterations by matrix algebra methods.

Methods of research – analytical and comparative.

One of the typical problems of data research is clustering. In the general case, it is difficult to establish the method of distribution and the number of clusters. And most of the known methods require the number of clusters as an input parameter. Therefore, clustering methods that independently determine the required number of clusters are important. One of the well-known methods is symmetric sparse non-negative matrix factorization. The purpose of the work is to implement multiplicative iterations using matrix algebra methods, which will speed up the clustering process compared to the naive approach.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

WCSS – Within-Cluster Sum of Square

DBSCAN – Density-based spatial clustering of applications

SOM – Self-organizing map

NMF – Non-negative matrix factorization

SVD – ingular-value decomposition

PCA – principal component analysis,

СНМФ – симетрична невід’ємна матрична факторизація,

ПК – персональний компьютер

ОП – оперативна пам’ять

ВСТУП

Кластеризація є однією із типових задач дослідження даних. Її суть – у розділенні даних на групи (кластери). Причому у загальному випадку кількість кластерів та принцип розподілу невідомі. Більшість відомих методів кластеризації отримують кількість кластерів як вхідний параметр, примушуючи дослідника його вгадувати, виходячи з інтуїтивних міркувань, що породжує сумніви щодо оптимальності побудованих кластерів. Тому цінними є методи кластеризації, які самостійно визначають оптимальну (у якомусь сенсі) кількість кластерів. Одним із відомих методів є симетрична розріджена невід'ємна матрична факторизація, яка поєднує наближення головних компонент та невід'ємну матричну факторизацію. Метою роботи є реалізація описаних формулами (2.1 – 2.3) в роботі [1] мультиплікативних ітерацій методами матричної алгебри, що дозволить прискорити процес кластеризації порівняно із наївним підходом.

Задачами роботи є:

а) ознайомлення із відомими методами кластеризації з метою виявлення різних підходів до визначення оптимальної кількості кластерів;

б) ознайомлення із методом головних компонент;

в) ознайомлення із методом симетричної розрідженої невід'ємної матричної факторизації;

г) реалізація мультиплікативних ітерацій методами матричної алгебри пакета numpy;

д) тестування швидкості роботи отриманої реалізації порівняно із наївним підходом.

1 ОГЛЯД МЕТОДІВ КЛАСТЕРИЗАЦІЇ

1.1 Кластеризація

Кластеризація або кластерний аналіз – це група методів, яка застосовується в машинному навчанні, аналізі даних та статистиці, й допомагає групувати подібні дані або об'єкти разом у кластери на основі їх подібності та відмінностей. Метою кластеризації є пошук закономірностей або структури у даних, які можуть допомогти краще зрозуміти основні явища чи підвищити точність прогностичних моделей.

При кластерному аналізі дані групуються в кластери на основі схожості або відмінності їх ознак або атрибутів. Даний показник, найчастіше, виміряють за допомогою різних метрік відстані між об’єктами, наприклад: евклідова відстань, подібність косинусів, відстань Жаккара. Існує багато різних методів кластеризації, які відрізняються між собою способом визначення кластера. Наприклад, кластеризація зверху вниз, метод к-середніх або кластеризація знизу вгору, методи DBSCAN та agglomerative clustering.

Однак існує кілька проблем, пов'язаних з проведенням кластерного аналізу:

е) визначення необхідної кількості кластерів;

ж) вибір правильної метрики відстані;

з) обробка високовимірних даних;

и) робота з неоднозначними даними;

к) інтерпретація та перевірка результатів.

Однією з основних проблем кластерного аналізу є визначення відповідної кількості кластерів. Хоча існує кілька методів визначення оптимальної кількості кластерів, таких як ліктьовий метод або силуетний аналіз, ці методи не є надійними та іноді можуть давати суперечливі результати.

Ще однією проблемою є вибір правильної метрики відстані для вимірювання схожості або відмінності між об’єктами. Різні метрики відстані можуть давати різні результати кластеризації, і важливо вибрати ту, яка найкраще відповідає даним та меті кластеризації.

Кластеризація високовимірних даних може бути складною, оскільки прокляття розмірності ускладнює пошук значущих кластерів у даних. Для вирішення цієї проблеми використовуються методи зменшення розмірності, такі як PCA або t-SNE.

У реальних наборах даних можуть бути присутні шум або неоднозначність даних, що може вплинути на результати кластеризації. В таких випадках використовуються методи виявлення винятків, очищення даних або зміна метрики відстані.

При інтерпретації та перевірки результатів кластеризації використовується кілька доступних показників, наприклад, оцінка силуету або індекс Девіса-Булдіна. Але в кінцевому підсумку інтерпретація результатів вимагає експертних знань предметної області та даних.

Визначення відповідної кількості кластерів є важливим етапом кластерного аналізу. Існують різні методи визначення відповідної кількості кластерів, розглянемо деякі з них: ліктьовий, силуетний методи, статистика розриву, ієрархічна кластеризація, знання предметної області.

У ліктьовому методі кількість кластерів наноситься на графік відносно суми квадратів усередині кластера (WCSS – Within-Cluster Sum of Square). Ділянка утворює лікоть, і кількість кластерів у точці ліктя вважається відповідною кількістю кластерів.

У силуетному методі коефіцієнт силуету вимірює, наскільки об'єкт схожий на власний кластер у порівнянні з іншими кластерами. Кількість кластерів, що максимізує середній коефіцієнт силуету, вважається відповідною кількістю кластерів.

При статистиці розриву порівнюється загальна варіація всередині кластера для різних значень k з їх очікуваними значеннями при нульовому еталонному

розподілі даних. Відповідна кількість кластерів – це значення k, яке максимізує статистику розриву.

Ієрархічна кластеризація використовується для визначення відповідної кількості кластерів шляхом вивчення дендрограми. Відповідна кількість кластерів – це кількість кластерів на рівні, де дендрограма розгалужується на окремі кластери. Під час побудови дендрограми використовуються розділювальні або агломератові (об'єднувальні) підходи [2].

Розділювальні підходи називають ще підходами «згори-вниз». Спочатку всі дані знаходяться у єдиному кластері, потім відбувається рекурсивне розбиття при русі вниз по ієрархії.

Агломератові (об'єднувальні) підходи відносяться до підходів «знизу-вгору». Спочатку кожен об’єкт має власний кластер, а далі пари кластерів об'єднуються при підйомі по ієрархії.

Знання предметної області або доменні знання також можуть бути використані для визначення відповідної кількості кластерів. Наприклад, якщо кластеризація проводиться на основі купівельної поведінки клієнтів, яка залежить від віку або статі.

Універсального підходу до визначення відповідної кількості кластерів не існує, і різні методи можуть давати різні результати, в залежності від конкретного набору даних або задачі. На практиці може бути використано кілька методів, результати яких в подальшому порівнюються для визначення найбільш підходящої кількості кластерів. Тому важливими є методи кластеризації [3], які самостійно визначають оптимальну кількість кластерів.

1.2 Кластеризація «згори-вниз»: метод k-середніх

При кластеризації «згори-вниз» всі дані спочатку знаходяться у єдиному кластері, далі відбувається рекурсивне розбиття при русі вниз по ієрархії. Оптимальною вважається кластеризація за допомогою методу k-середніх,

популярному алгоритму машинного навчання без нагляду, який використовується для групування даних в k кластерів на основі їх схожості.

Метою методу є розподіл n об’єктів на k кластерів, так щоб кожен об’єкт відносився до кластера з найближчим до нього середнім значенням. Метод ґрунтується на мінімізації суми квадратів відстаней між кожним об’єктом та центром кластера (центроїдом):

$$\sum_{i=1}^{n} d(x_i, m_j(x_i))^2, \tag{1.1}$$

де d – метрика, яка використовується для обчислення схожості;

x_i – i-ий об'єкт даних;

$m_j(x_i)$– центр кластера, якому на j-ій ітерації приписаний елемент x_i.

Основною задачею методу k-середніх є мінімізація суми квадратичних відстаней між кожним об'єктом даних і присвоєним йому центроїдом.

У початковий момент роботи довільним чином обираються центри кластерів, їх кількість k можна визначити за допомогою різних методів, наприклад ліктьового або силуетного. Далі ітеративно обчислюється відстань кожного об'єкту до центрів з додаванням кожного об'єкта до кластера з найближчим центром. Для кожного з отриманих кластерів обчислюються нові значення центрів, при цьому мінімізується функція, що обчислює якість поточного розбиття множини на k кластерів: сумарне квадратичне відхилення елементів кластерів від центрів цих кластерів буде найменшим. Після чого повторюється процедура перерозподілу об'єктів між кластерами.

Таким чином метод містить такі основні кроки:

л) обрається k початкових центрів кластерів (центроїдів);

м) кожен об'єкт зіставляється з кластером, відстань до центра якого мінімальна;

н) проводиться перевірка, що в кожному кластері міститься хоча б один об'єкт. Для цього кожний порожній кластер доповнюється довільним об'єктом, що розташовано «далеко» від центра кластера;

о) центр кожного кластера замінюється середнім;

п) за потреби перелічені зверху пункти повторюються.

Однією з переваг методу *k*-середніх є його простота, ефективність та зрозумілість. Однак метод залежить від початкового розміщення центроїдів, і, в залежності від вибору, результати можуть змінюватись. Також, він може погано спрацьовувати на наборах даних з несферичними або різними за розмірами кластерами. Крім того, кількість кластерів повинна бути заздалегідь визначена дослідником, що не завжди є можливим.

Поширеність методу k-середніх зумовлено його головними перевагами:

р) простотою у використанні;

с) гнучкістю;

т) швидкою збіжністю;

у) зрозумілістю;

ф) можливістю перевірки статистичної значимості відмінностей між виділеними кластерами.

Але метод суттєво обмежується такими недоліками:

х) результати кластеризації за методом k-середніх значною мірою залежать від вибору початкової конфігурації центроїдів (для усунення цієї проблеми можна застосовувати методику поступового збільшення значення числа кластерів) [1];

ц) кількість кластерів повинна бути заздалегідь визначена дослідником;

ч) робота алгоритму суттєво уповільнюється під час кластеризації великих обсягів даних (можливим вирішенням даної проблеми є використання вибірки даних);

ш) алгоритм може сходитися до локального мінімуму цільової функції [1];

щ) алгоритм чутливий до викидів, які можуть викривлювати середнє;

ы)порушення умови зв'язності елементів одного кластера [2].

1.3 Кластеризація «знизу-вгору»: DBSCAN, agglomerative clustering

При кластеризації «знизу-вгору» кожен об’єкт спочатку відноситься до свого власного кластера, а далі пари кластерів об'єднуються при підйомі по ієрархії. До такого типу кластеризації відноситься просторова кластеризація з шумом на основі щільності, найбільш поширеними є методи DBSCAN та агломеративна кластеризація (agglomerative clustering).

Основною ідеєю DBSCAN є групування в групи точек, які щільно розташовані в просторі даних, тобто мають багатьох сусідів, при цьому точки, які мало розподілені, тобто чиї сусіди розташовані занадто далеко, розглядаються як шум (викиди).

Для початку обирається випадкова точка в наборі даних, а потім знаходяться всі точки в заданому радіусі навколо неї. Якщо кількість точок в межах цього радіуса перевищує порогове значення, виділена точка вважається ядром і утворюється новий кластер. Потім алгоритм рекурсивно розширює кластер, знаходячи всі сусідні точки в межах одного радіуса, і додаючи їх до кластера.

Точки, які самі по собі не є точками ядра, але знаходяться в його радіусі, вважаються прикордонними та додаються до того ж кластеру. Точки, які не є точками ядра і не знаходяться в радіусі будь-якої основної точки, вважаються точками шуму або викидом й ігноруються. Процес триває до тих пір, поки всі точки не будуть віднесені до кластерів або ідентифіковані як шумові.

Перевагами DBSCAN є:

э) можливість ідентифікувати кластери довільної форми і розміру, що робить його більш універсальним, і дозволяє застосовувати до більш широкого діапазону наборів даних;

ю) ефективна обробка шуму і винятків, що важливо в реальних наборах даних, де наявність шуму є поширеним явищем;

я) швидкість і масштабованість, що робить його зручним при використанні на великих наборах даних.

Проблемами DBSCAN є:

аа) визначення оптимальних значень для радіуса і порогових параметрів може істотно вплинути на результати та якість кластеризації;

бб) якість кластеризації залежить від функції відстані, що ускладнює кластеризацію багатовимірних даних;

вв) складність при кластеризації наборів даних з великим перепадом щільності, оскільки важко підібрати поєднання радіуса і порогових параметрів, яке б відповідало різним кластерам.

Взагалі, DBSCAN добре підходить для різних наборів даних, особливо тих, що мають шум і винятки.

Агломеративна кластеризація (agglomerative clustering), також відноситься до ієрархічної або агрегатної кластеризації. Застосовується в машинному навчанні та аналізі даних. Базується на основі ітераційного злиття пар найбільш схожих об’єктів даних або кластерів разом, поки всі об’єкти або кластери не будуть об'єднані в єдиний кластер. На початку кожен об’єкт даних знаходиться у власному кластері, а потім кластери, розділені найменшою відстанню, послідовно об'єднуються. Схожість між двома кластерами обчислюється за допомогою різних метрик відстані, залежно від типу даних.

На кожному кроці проводиться злиття двох кластерів. Відстань між ними визначається однією парою елементів кожного кластера, які мають найменшу відстань. Об’єднання проводиться поки всі елементи не опиняться в одному кластері. Такий підхід відомий як кластеризація найближчих сусідів.

Отриману ієрархію кластерів можна візуалізувати за допомогою дендрограми, яка показує послідовність злиття кластерів.

Агломеративна кластеризація використовується для сегментації зображень, сегментації ринку, аналізу експресії генів. До переваг відносять відносну простоту у реалізації, та можливість роботи з великими наборами даних. До недоліків те, що може потребувати значних обчислювальних затрат, і

є чутливим до вибору метрики відстані та критеріїв зв'язку, що використовуються для визначення схожості між кластерами.

DBSCAN – це алгоритм кластеризації на основі щільності, який групує точки даних на основі їх близькості та щільності, тоді як агломеративна кластеризація – це ієрархічний алгоритм кластеризації, який ітераційно об'єднує найближчі кластери разом на основі метрики відстані.

Просторова кластеризація даних з шумом на основі щільності (DBSCAN) і агломеративна кластеризація – це обидва алгоритми кластеризації, що використовуються в навчанні без нагляду.

1.4 Глибинне навчання

Глибинне навчання – це тип машинного навчання, який передбачає використання нейронних мереж з декількома шарами для аналізу та навчання на основі даних. Може поєднувати алгоритми навчання з вчителем і без.

При глибинному навчанні нейронні мережі складаються з безлічі шарів, які обробляють вхідні дані і передають їх на наступний рівень. Кожен наступний шар отримує вихідні дані попереднього шару, а вихід одного шару стає входом для наступного.

Таким чином глибинне навчання характеризується наявністю декількох шарів нелінійної обробки. Навчанням з вчителем або без ознак кожного шару, формуя ієрархію від низького до високого рівня. Це дозволяє нейронній мережі моделювати все більш складні представлення даних [4].

Чим складніше дані, що моделюються, тим складніше їх представлення. Автоматичне виділення важливих особливостей, є однією з причин вдалого застосування машинного навчання.

Часто дані представляються у вигляді графа. Адже графи – це потужний і гнучкий спосіб представлення даних [5, 6]. Завдяки такому підходу представлення даних є більш виразним та дозволяє поглянути з іншого боку на

різні проблеми [7].

За допомогою графа можна відобразити будь-які складні дані. Таке представлення можна застосовувати як до розробки ліків, так і до аналізу рекомендацій у соціальних мережах. Графи стали основою для численних систем, яка дозволяє обробляти, зберігати та отримувати доступ до знань про взаємодіючі об'єкти [7].

Графи застосовуються у різних галузях, наприклад, соціальній, суспільній, природознавчій, фізичній [8, 9], медичній [10]. З їх допомогою можна будувати графи знань, а також використовувати в багатьох інших галузях [11, 12].

Глибинне навчання застосовується в багатьох областях, наприклад, комп'ютерний зір, обробка природної мови, розпізнавання мови. Це дозволило досягти значних успіхів у розпізнаванні зображень, мовлення, машинному перекладі.

Досягнуті успіхи стимулювали розробку таких потужних інструментів та фреймворків, як Keras, TensorFlow, PyTorch, що дозволяють легко будувати та навчати складні нейронні мережі.

1.5 Самоорганізаційна карта Кохонена

Самоорганізаційна карта Кохонена (SOM) – нейронна мережа, що використовує некероване навчання. Застосовується для створення низьковимірного, як правило двовимірного, представлення (карти) набору даних вищої розмірності зі збереженням їх топологічної структури.

Самоорганізаційна карта Кохонена, як і більшість інших нейронних мереж, працює у двох режимах: навчанні та відображенні. Під час навчання, за допомогою вхідних даних, створюється карта (низьковимірне представлення вхідних даних). При цьому може використовується, наприклад, конкурентний процес або векторне квантування, тоді як при відображенні новий вхідний вектор

автоматично класифікується. Мета навчання полягає в тому, щоб різні частини мережі реагувати однаково на певні вхідні шаблони [13].

Карта складається з компонентів, які називаються вузлами або нейронами. Простір карти, у більшості випадків двовимірний, визначається заздалегідь як скінчення область, де вузли (нейрони) розташовані у правильній гексагональній або прямокутній сітці.

Кожному вузлу приписується вектор з вагою, який має таку ж розмірність, що і вхідні вектори. Незважаючи на те, що вузли в просторі карти залишаються фіксованими, під час тренування вектори ваги переміщуються у напрямку вхідних даних (відбувається зменшення метрики відстані) без зміни топології.

Після навчання, за допомогою карти, вектор з вхідного простору можна класифікувати обчислюючи метричну відстань до найближчих вузлів, вузол з найменшою відстанню і буде найближчим.

2 ФАКТОРИЗАЦІЯ

2.1 Задача факторизації

Однією із задач аналізу даних є кластеризація, тобто розділення даних на групи (кластери). Найчастіше при цьому принцип розподілу на кластери та їхня кількість невідомі. Значна частина методів кластеризації потребують визначення кількості кластерів на початку аналізу, змушуючи вказувати його, спираючись на досвід дослідника. Такий підхід не дає однозначної відповіді щодо оптимальності кількості побудованих кластерів. Тому важливими є методи кластеризації, які самостійно визначають кількість кластерів [16, 14, 15].

Одним із методів, що дозволяють вирішити цю проблему, є симетрична розріджена невід'ємна матрична факторизація [17]. Вона поєднує наближення головних компонент та невід'ємну матричну факторизацію. Метою роботи є створення програмного продукту, який реалізує, описані в роботі Добровольского Г.А. [1], формули мультиплікативних ітерацій методами матричної алгебри, що дозволить прискорити процес кластеризації порівняно із наївним підходом.

Задача факторізації найчастіше вирішується методом градієнтного спуску. Якщо взяти ПК з процесором *IntelCore i5* та *8Гб* ОП, то типова тривалість факторізації буде близько години на матрицях розміру 5000×5000.

Тому потрібно розглянути інший метод факторізації, реалізувати його, оптимізуючи швидкість виконання, та порівняти з методом градієнтного спуску.

2.2 Методи матричної факторизації

Факторизацією називають розкладання на множники – це декомпозиція об'єкта у добуток інших об'єктів, або множників, які після перемноження дадуть

вихідний об'єкт.

Матрична факторизація – це метод, який використовується для розкладання великої матриці на менші матриці, які описують найважливіші ознаки вихідної матриці. Ціль такого розкладу в тому, щоб знайти дві або більше матриць, результат множення яких був би максимально наближений до вихідної матриці. Такий підхід дозволяє зменшити розмірність даних та полегшити їх аналіз.

Використовується в лінійній алгебрі та машинному навчанні. Існують різні методи матричної факторизації, наприклад, сингулярний розклад матриць (SVD), невід'ємна матрична факторизація (NMF) та метод головних компонент (PCA).

При сингулярному розкладі матриць (SVD) матриця розкладається на три матриці: ліву сингулярну матрицю, діагональну матрицю сингулярних значень і праву сингулярну матрицю. SVD використовується для зменшення розмірності та стиснення набору даних, виявлення закономірностей.

Метод головних компонент (PCA) дозволяє зменшити розмірність набору даних за допомогою виділення їх найважливіших функцій або компонентів. В PCA на матриці даних застосовується SVD, та вибираються верхні основні компоненти.

Розклад на невід'ємні матриці (NMF) – це метод, який використовується для розкладання невід'ємних матриць на менші матриці, які також будуть невід'ємні. Використовується для виявлення закономірностей в даних, які нелегко зафіксувати традиційними методами факторизації матриці [16, 17].

SVD можна використовувати в самих різних сферах: обробка зображень, стиснення даних, машинне навчання, наприклад.

При стисненні зображень за допомогою SVD можна зменшити кількість сингулярних значень, які використовуються для їх реконструкції.

SVD можна використовувати для побудови рекомендаційних систем шляхом аналізу оцінок користувачів продуктів або послуг. Метод можна використати для виявлення закономірностей в даних і прогнозування того, які

продукти або послуги, ймовірно, зацікавлять користувача.

Також SVD може бути використаний при обробці природної мови для виявлення найважливіших понять або тем у великому корпусі тексту. Метод можна використати для кластеризації схожих документів, ідентифікації ключових словосполучень або слів, побудови семантичних моделей мови тощо.

В інтелектуальному аналізі даних для виявлення закономірностей і тенденцій у великих наборах даних також можна застосувати SVD. Метод може бути використаний для зменшення розмірності даних, що полегшує їх аналіз і візуалізацію.

2.3 Метод головних компонент

Метод головних компонент (PCA) вважається одним з основних підходів до зменшення розмірності даних, та втрати при цьому найменшої кількості інформації. Зменшення розмірності великих наборів відбувається за рахунок проектування даних у нову систему координат. PCA обчислює власні векторі і власні значення коваріаційної матриці даних. Власні вектори представляють головні компоненти, а власні значення представляють величину дисперсії, захопленої кожним компонентом. Після цього головні компоненти сортуються в порядку убивання їх власних значень, а для представлення даних в новій системі координат обираються перші k основних компонентів.

Використання PCA передбачає лінійну залежність між змінними і може погано спрацьовувати в іншому випадку. Також треба зазначити, що PCA не завжди може видавати значущі результати у висококорельованих наборах даних, та бути чутливим до винятків у даних.

PCA широко використовується в аналізі даних та машинному навчанні для зменшення розмірності великих наборів даних шляхом виявлення найважливіших ознак, які сприяють мінливості даних. Таким чином PCA дозволяє зменшити кількість вимірів, зберігаючи при цьому більшу частину

інформації.

PCA також використовують для візуалізації високовимірних даних, проектуючи їх на низьковимірний простір. Це дозволяє дослідити зв'язки між змінними, виявляти закономірності та кластери в даних.

Крім того PCA застосовується для попередньої обробки даних перед застосуванням підходів машинного навчання. Зменшуючи розмірність даних, PCA дозволяє підвищити продуктивність моделей машинного навчання за рахунок зменшення обчислювальної складності.

Також PCA використовують під час обробки природної мови для зменшення розмірності векторів слів. Виявивши головні компоненти векторів слова, можна зменшити розмірність вхідного простору, зберігаючи найважливішу інформацію про слова.

При стисненні зображення PCA застосовують для зменшення розмірів зображення, спрощуючи його зберігання та передачу. Зображення представляється у вигляді лінійного поєднання його головних компонент, які відповідають найважливішим ознакам зображення, а компоненти з низькою дисперсією відкидаються.

2.4 Методи виявлення спільнот

Машинне навчання успішне використовується в різних галузях. Сфера його застосування стрімко росте. Важливою складовою є автоматичне виділення важливих особливостей при моделюванні даних, причому чим складніше дані, тим складніше їх представлення. Нерідко дані моделюються у вигляді графа, такий підхід вважається досить гнучким та дозволяє краще візуалізувати представлення. Моделювати можна практично будь-які дані, як графи вузлів у поєднанні зі зв’язками.

Різні дослідження останніх років показують, що можна отримати глибше розуміння мережевих систем, досліджуючи їх структуру. Структура мережі – це

опис поведінки системи та того, як мережа допомагає вивчати, аналізувати та прогнозувати поведінку кожного вузла, а також загальну динаміку системи [18]. При дослідженні структури мережі, зазвичай систему ділять на компоненти, які можуть перекриватися. Таким чином виявляються спільноти, які складаються з групи вузлів із більш складними відносинами між собою, ніж з іншими спільнотами.

Виявлення спільнот помагає зрозуміти формування та еволюцію мережевих складних систем. Багато ранніх досліджень було зосереджено на виявленні спільнот без перекриття, де кожен вузол призначався одній спільноті, наприклад, методи, засновані на теорії ігор [19]. В реальному житті більше розповсюджені спільноти, що перекриваються, в яких вузол може брати участь у кількох спільнотах одночасно.

Наприклад, в соціальних мережах вузлами будуть користувачі та групи. Імовірно, що користувачі скоріше стануть друзями, якщо матимуть схожі хобі, спільних друзів або спільні групи. Крім того друзі найчастіше матимуть однакові хобі та належатимуть до одних груп. На основі цього аналізу кожен вузол пов’язують із вектором ідентичності.

Виявлення спільнот є важливим інструментом, що допомагає зрозуміти формування та розвиток мережевих складних систем: механізм, що складається з виявлення спільнот, що не перекриваються, і виявлення спільнот, що перекриваються. Ранні дослідження, в основному, сконцентровано на виявленні спільнот без перекриття, тобто кожен вузол належить лише одній спільноті [20, 19]. В реальності більш поширені спільноти, що перекриваються, тобто кожен вузол може належати кільком спільнотам одночасно. Алгоритмами виявлення спільнот, що перекриваються, є: метод клікової перколяції CPM (the clique percolation method), метод послідовної клікової перколяції SCP (sequential clique percolation method), жадібне клікове розширення GCE (greedy clique expansion), метод локальної оптимізації статистики порядку OSLOM (order statistics local optimization method), BigClam і NRL (network representation learning) тощо [19]. Тобто роботи по вдосконаленню процесу виявлення спільнот ведуться у різних

напрямках.

2.5 Постановка задачі

Метою роботи є створення програмного продукту, який реалізує, описані в роботі Добровольского Г.А. [1], формули мультиплікативних ітерацій методами матричної алгебри.

Задача симетричної невід'ємної матричної факторизації (СНМФ) [16, 17]:

$$A=HH^T \tag{2.1}$$

де A – матриця суміжності графа,

H – матриця, множник, якій потрібно знайти для вирішення задачі факторізації.

Кожен стовпчик матриці H відповідає окремому кластеру, а кожен елемент стовпчика визначає міру належності елемента заданому кластеру.

Матриця суміжності – це один із засобів представлення графу у вигляді матриці. У задачах факторізації використовується симетрична матриця суміжності, яка містить нулі на головній діагоналі.

Міра належності H_{ab} – умовна імовірність $P(a|b)$, того, що слово a належить кластеру b.

Для вирішення задачі СНМФ було запропоновано ітерацію (2.2).

$$H_{ab}(n+1) = H_{ab}(n)\frac{\sum_j P_{aj}H_{jb}(n)}{\sum_{j,p} H_{ap}(n)H_{jp}(n)H_{jb}(n)} + \frac{\lambda}{4}\frac{H_{ab}(n)}{\sum_{j,p} H_{ap}(n)H_{jp}(n)H_{jb}(n)}, \tag{2.2}$$

де a – кількість слів в словнику,

b – максимальна кількість кластерів,

P – матриця імовірностей,

λ – параметр регуляризації,

H – матриця, множник, якій потрібно знайти для вирішення задачі факторізації.

Параметр регуляризації λ часто використовується в задачах машинного навчання, та означає додавання деякої додаткової умови, що дозволяє уникнути перенавчання чи знайти рішення для некоректно поставленої задачі.

Подробиці обчислення та необхідні викладки потрібні для створення даної формули приведені у роботі Добровольского Г.А. [1].

Ітерації формули (2.2) за умовою λ = 0 мають мультиплікативну форму, близьку до невід'ємної матричної факторизації [21]:

$$H_{ab}(n+1) = H_{ab}(n)\frac{\sum_j P_{aj}H_{jb}(n)}{\sum_{j,p} H_{ap}(n)H_{jp}(n)H_{jb}(n)}, \quad (2.3)$$

У формулах (2.2) і (2.3) матриця A_{aa} симетрична і складається з невід'ємних елементів, шуканий множник H_{ab} теж складається з невід'ємних елементів. Параметр регуляризації λ управляє ступенем розрідженості/генералізації моделі.

Як приклад, матриця A_{ab} може обчислюватись, як матриця відстаней між вузлами неорієнтованого графа або як матриця парних імовірностей.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДА

3.1 Програмне оточення та бібліотеки

Для написання програми було вибрано мову Python, через легкість кодування, її популярність та необхідні для розробки бібліотеки [22].

Python є інтерпретованою мовою програмування. Це об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією [23].

До плюсів мови Python відносять:

гг) легкість написання коду, через що він підходить для початківців;

дд) велика інтернет-спільноти;

ее) наявність логічного синтаксису, який легко читати та сприймати;

жж) масштабованість, гнучкість;

зз) наявність великої кількості бібліотек.

Завдяки цим перевагам розробка буде швидшою, ніж з використанням інших мов програмування. Крім того Python це інтерпретована мова програмування, що свідчить про те, що до запуску це звичайний текстовий файл. Даний підхід дозволяє програмувати майже на всіх платформах.

Популярність python, значна підтримка, наявність різнопланових бібліотек значно полегшують процес розробки програм. При написанні програм з використанням матричних операцій застосовується пакет NumPy.

NumPy (Numerical Python extensions) – перекладається як числові розширення Python, одна з бібліотек Python, яка застосовується для математичних обчислень. Містить як базові функції, так і лінійну алгебру [24].

Вихідний код бібліотеки зберігається на GitHub, і є у вільному доступі, тому NumPy є досить популярною, також її називають open-source модулем для Python.

Бібліотека написана на мовах C та Fortran, що відносяться до мов, що компілюються. Завдяки цьому швидкість обчислення буде більша, ніж, якщо

використовувати мови, що інтерпретуються.

NumPy використовується в різних сферах, наприклад: наукових обчисленнях, Data Science, Machine Learning, візуалізації даних або для створення нових бібліотек.

Багато вчених користуються саме NumPy для вирішення складних завдань з великою кількістю даних.

Також бібліотека використовується для роботи з даними, для їх аналізу, обробки, оцінки тощо, та дозволяє візуалізувати великі набори даних

Бібліотеки SciPy, scikit-learn, PyViz також базуються на NumPy.

Бібліотеки Dask, CuPy чи XND, що працюють з новими типами масивів, написані на основі NumPy.

3.2 Вибір показника якості роботи

Оцінка складності алгоритму – це кількісна оцінка необхідних для його реалізації ресурсів. Зазвичай оцінюються необхідні для виконання час та кількість пам'яті.

При чому складність майже завжди залежить від вхідних даних, їх розміру. Наприклад матриці малого розміру будуть оброблені швидше, ніж аналогічні більшого розміру. Зрозуміло, що точний час складно оцінити, так як це залежить від типу даних, мови програмування, апаратної та програмної частин тощо. Тому важливою вважається асимптотична складність, тобто складність яка оцінюється на великих розмірах вхідних даних.

Задачею даної роботи є підтвердження залежності часу роботи програмного продукту від вхідного параметру – розміру матриці.

Створений програмний продукт реалізує, описані в роботі Добровольского Г.А. [1], формули мультиплікативних ітерацій методами матричної алгебри.

Як правило, у задачах тематичного моделювання, розмір вхідної матриці

близько 10 000×10 000 елементів, тому важливу роль грає саме швидкість обчислень.

3.3 Опис програмного продукту

Розглянемо запис формул мультиплікативних ітерацій (формули 2.1 - 2.3) з використанням та без методів матричної алгебри.

Для чистоти експерименту та подальшого тестування час роботи факторізації буде порівнюватися з методами градієнтного спуску та наївної реалізації.

При використанні наївного підходу, тобто програмування формули, як є, без використання методів матричної алгебри, формула 2.2 в програмному продукті буде містити багато обчислень за допомогою вкладених циклів. Такий підхід значно уповільнить роботу методу та буде погано спрацьовувати на великих наборах даних. Приведемо частину програми, що відповідає за розрахунок формули 2.2:

```
for i_word in words_range:
 for i_topic in topics_range:
 _denominator = 0.000001 # regularization constant
 for p in topics_range:
 for j in words_range:
 _denominator += h[i_word][p] * h[j][p] * h[j][i_topic]
 _numerator = L
 for j in words_range:
 _numerator += wwcovar[i_word][j] * h[j][i_topic]

 _factor[i_word][i_topic] = _numerator / _denominator
```

```
for i_word in words_range:
for i_topic in topics_range:
h[i_word][i_topic] = h[i_word][i_topic] * _factor[i_word][i_topic]
```

З приведеного вище коду видно, що для обчислення формули використовуються вкладені цикли. А зазвичай найбільш складними частинами програми є саме виконання циклів.

Запис формули 2.2 з використанням методів матричної алгебри буде виглядати таким чином:

```
ht=h.T
_denominator=h.dot(ht.dot(h))
_numerator=p.dot(h)

_factor = _numerator / _denominator
h = h*_factor
```

Для запису формули було використано такі бібліотечні функції, як транспонування матриці, множення матриць та множення матриці на число. Зазвичай бібліотечні функції спрацьовують швидше та ефективніше. Та й код виглядає простішим та більш зрозумілим.

При виконанні програмного коду вхідними даними будуть:

n_words – кількість слів в словнику;

n_topics – максимальна кількість кластерів;

p_max – максимальна кількість ітерацій (не обов’язковий параметр, за замовчуванням дорівнює 200);

wwcovar – матриця, яку потрібно розкласти на множники;

λ – параметр регуляризації;

H_0 – нормована матриця, заповнена рандомними приблизними значеннями (перше наближення матриці H).

На виході обчислюється:

H – матриця, множник, якій потрібно знайти для вирішення задачі факторізації.

3.4 Тестування швидкості роботи отриманої реалізації

Метою роботи було прискорення проведення мультиплікативних ітерацій в процесі матричної факторізації. Тому метою тестування буде вимірювання часу роботи функцій факторізації, в залежності від розміру вхідних даних. Та порівняння з аналогічними показниками наївної реалізації та методом градієнтного спуску.

Запуск проводився на ноутбуку з процесором intel(R) Core(TM) i5-6300U, з 8ГБ ОП, 64-разрядною операційною системою Windows 10 Pro.

Вхідні дані були однакові для всіх трьох методів:

n_words – кількість слів в словнику;

n_topics – максимальна кількість кластерів;

p_max – максимальна кількість ітерацій (не обов’язковий параметр, за замовчуванням дорівнює 200);

wwcovar – матриця, яку потрібно розкласти на множники;

λ – параметр регуляризації;

H_0 – нормована матриця, заповнена рандомними приблизними значеннями (перше наближення матриці H).

На виході обчислюється:

H – матриця, множник, якій потрібно знайти для вирішення задачі факторізації.

Перед викликом функцій матриця wwcovar заповнюється рандомними значеннями від 0 до 1 та нормується по стовпцях. В програмному коді:

```
wwcovar= wwcovar.dot(wwcovar.T)
```

wwcovar=wwcovar/numpy.linalg.norm(wwcovar, axis=0)

Після нормування, сума по стовпцях може не дати одиницю через похибку округлення. Так як програмний продукт працює з матрицями, чим більше набір даних, тим більше імовірність того, що похибка округлення збільшить помилку.

На невеликих наборах даних додатково ще проводилось вимірювання залежності похибки від номера ітерації. Таке вимірювання доречно проводити лише на малих наборах, бо друк даних значно впливає на час виконання програми.

В таблиці 3.1 приведено значення похибок на деяких ітераціях для вхідної матриці розміром 3×2.

Таблиця 3.1 – Значення похибки, в залежності від номера ітерації

№ ітерації	Значення похибки для	
	наївної реалізації	розробленого методу
49	0,258667174	1,629715383
99	1,629410758	1,629715383
149	1,629259096	1,629715382
199	1,629107545	1,629715382
249	1,628955622	1,629715338
299	1,628782781	1,629713193
349	1,628624675	1,629657582
399	1,628550553	1,629708157
449	1,628443769	1,629715381
499	1,628337058	1,629715383

Як видно з таблиці 3.1 збіжність розробленого методу повільніша, ніж у наївної реалізації, але, при цьому, швидкість значно вища. Значення швидкості приводиться у таблиці 3.2.

У таблиці 3.2 приведено результати тестування для розробленого

програмного продукту, наївної реалізації та метода градієнтного спуску. Метою тестування буде вимірювання часу роботи функцій факторізації, в залежності від розміру вхідних даних. Для кожного з методів, для різного розміру вхідної матриці, приводяться значення середній помилки та час виконання.

Таблиця 3.2 – Результати тестування

№ П/п	Метод	Розмір матриці	Помилка	Час
1	2	3	4	5
1.		3×2		
	градієнтного спуску		0,920913326	0,017500162
	наївної реалізації		8,394817204	0,045402288
	розроблений		8,508106023	0,014060736
2.		20×10		
	градієнтного спуску		2,240980423	0,143660307
	наївної реалізації		256,2063742	20,05014038
	розроблений		1885,573554	0,01810813
3.		30×20		
	градієнтного спуску		2,74013617	0,441052437
	наївної реалізації		303,1397817	187,5124207
	розроблений		8466,997319	0,03079319
4.		100×50		
	градієнтного спуску		10	0,068136692
	наївної реалізації		227,2818268	98544,80397
	розроблений		236307,7518	0,554724932
5.		300×200		
	градієнтного спуску		17,32050808	0,309055328
	розроблений		8505269,298	3,051281214

Кінець таблиці 3.2

1	2	3	4	5
6.		3000×2000		
	градієнтного спуску		54,77225575	28,89819646
	розроблений		8520120635	876,2039547
7.		5000×4500		
	градієнтного спуску		70,71067812	225,7598429
	розроблений		53026344292	5798,544063
8.		10000×10000		
	градієнтного спуску		100	595,3875923
9.		30000×20000		
	градієнтного спуску		Нестача пам’яті	

Тестування показало, що наївну реалізацію можна використовувати лише на невеликих наборах даних, бо вже на матриці 300×200 він не видав результат після 12 годин роботи. Тому в подальшому тестуванні, на більш великих вхідних даних його час роботи не обчислювався.

Метод градієнтного спуску можна використовувати на досить великих наборах даних, але й він має свою межу.

З таблиці 3.2 видно, що найменша помилка буде у метода градієнтного спуску, а найбільша у розробленого. Такі значення можуть бути через похибку округлення, так як програмний продукт використовує матричні операції. Тому чим більше набір даних, тим імовірніше, що похибка округлення збільшить помилку.

При обробці матриць невеликого розміру найшвидшим буде саме розроблений метод. Але на великих даних, через матричні операції, він буде спрацьовувати повільніше градієнтного спуску.

Після проведеного тестування можна зробити висновок, що використання матричних операцій допомогло покращити наївну реалізацію. Але, в подальшому, потребує доопрацювання.

3.5 Практичне застосування до виявлення спільнот

Розглянемо спосіб застосування розробленого програмного продукту до виявлення спільнот на даних, описаних в статті [18].

Вхідними даними будуть:

n_words – кількість вузлів у графі;

n_topics – кількість спільнот;

p_max – максимальна кількість ітерацій (не обов’язковий параметр, за замовчуванням дорівнює 200);

A – матриця суміжності неорієнтованого графа соціальних зв'язків;

λ – параметр регуляризації;

H_0 – нормована матриця, заповнена рандомними приблизними значеннями (перше наближення матриці H).

Результатом виконання програми буде:

H – матриця, множник, якій потрібно знайти для вирішення задачі факторізації:

$$A=HH^T \tag{3.1}$$

де A – матриця близькості вузлів неорієнтованого графа соціальних зв'язків.

Для виявлення спільнот необхідно проаналізувати отриману матрицю H, в якій кожен рядок це один вузол графа, а кожен стовпець – це спільнота.

Для цього необхідно виконати такі кроки:

а) знайти суму в кожному стовпці матриці H, отримані показники будуть ненормованими імовірностями того, що навмання взятий вузол належить до спільноти a:

$$weight[a]=sum(H[a,b], b), \tag{3.2}$$

де a – відповідна спільнота;

b – відповідні вузли графу.

б) з обчислених показників вибрати N найбільших значень *weight[a]* таким чином, щоб їхня сума дорівнювала 95% від загальної суми всіх ваг;

в) нормувати кожен рядок матриці H, зважаючи на те, що рядок це один вузол графа, отриманий результат і буде ілюструвати ступінь приналежності вузла до спільноти.

Таким чином позиція з найбільшим результатом в рядку b; і буде номером спільноти, до якої належить вузол b.

Розглянемо застосування описаного алгоритму до виявлення спільнот на прикладі модельної задачі, граф до якої представлено на (рис. 3.1).

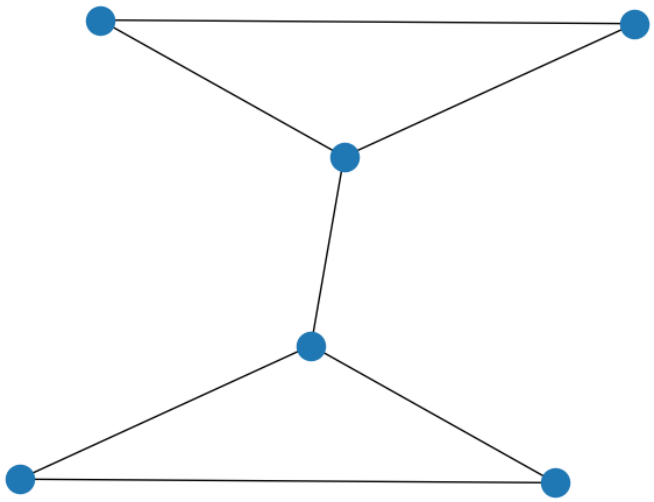

Рисунок 3.1 – Граф до модельної задачі

Вхідними параметрами будуть 2 спільноти та 6 вузлів. На основі графа складається матриця близькості між вузлами:

$$similarity[r, c] = 1 / (1 + length[r][c]), \quad (3.3)$$

де *length[r][c]* – відстань між вузлами *r* та *c*, найменша кількість ребр, які потрібно пройти від вузла *r* до вузла *c*.

Обчислена матриця близькості подається на вхід алгоритму факторизації.

Для наведеної задачі взято такі параметри:

```
parameters={"lambda":0.00001}
parameters['eta'] = 0.1
parameters['beta'] = 0.999
parameters['beta2'] = 1.001
parameters['maxError'] = 1e-10
parameters['maxIterations']=300000
```

Матриця близькості:

```
array([[1., 0.5, 0.5, 0.33333333, 0.25, 0.25 ],
       [0.5, 1., 0.5, 0.33333333, 0.25, 0.25 ],
       [0.5, 0.5, 1., 0.5, 0.33333333, 0.33333333],
       [0.33333333, 0.33333333, 0.5, 1., 0.5, 0.5 ],
       [0.25, 0.25, 0.33333333, 0.5, 1., 0.5 ],
       [0.25, 0.25, 0.33333333, 0.5, 0.5, 1. ]])
```

За результатами роботи обчислюється сума за стовбцями:

```
for c in res_grad.T:
  weight_sums.append(sum(c))
```

Отримані результати сортуються та нормуються:

```
sorted_weight_sums=sorted(weight_sums, reverse=True)
print(sorted_weight_sums)
norm = sum(weight_sums)
```

Далі обчислюється кумулятивно накопичувальна сума, як тільки вона стає більше порогового значення 0.5, подальше обчислення припиняється й

запам’ятовується мінімальне значення в стовбці. Всі стовбці, сума яких більше або дорівнює мінімальній, залишаються в результаті:

```
for i in range(len(sorted_weight_sums)):
  x = sum(sorted_weight_sums[:i+1]) /norm
  print(x)
  if x > 0.5:
    min_weight_sum = sorted_weight_sums[i]
    break
```

Для кожного рядка з усіх стовпчиків, що залишились обирається стовпчик з максимальним значенням, це і буде номер спільноти, до якої належить вузол:

```
binary_res = res_grad.copy()
for r in binary_res:
  cluster_id = numpy.argmax(r * mask)
  for i in range(len(r)):
    r[i] = 1 if i==cluster_id else 0
print(binary_res)
```

В результаті отримуємо:

```
[[0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1.]]
```

З результату видно, що вузли належать до двох спільнот та відповідає графу на (рис. 3.1).

ВИСНОВКИ

В даній кваліфікаційній роботі розглядались методи кластеризації, які самостійно визначають оптимальну кількість кластерів. Одним з них є симетрична розріджена невід'ємна матрична факторизація. Вона базується на методах наближення головних компонент та невід'ємной матричной факторизації.

Метою роботи була реалізація описаних формулами (2.1 – 2.3) в роботі [1] мультиплікативних ітерацій методами матричної алгебри, що дозволило прискорити процес кластеризації порівняно із наївним підходом.

Після розробки було проведено тестування швидкості роботи отриманої реалізації, в залежності від розміру вхідних даних. Порівняння проводилось з аналогічними показниками наївної реалізації та методом градієнтного спуску.

В ході роботи були виконані такі задачі:

а) ознайомлення із відомими методами кластеризації з метою виявлення різних підходів до визначення оптимальної кількості кластерів;
б) ознайомлення із методом головних компонент;
в) ознайомлення із методом симетричної розрідженої невід'ємної матричної факторизації;
г) реалізація мультиплікативних ітерацій методами матричної алгебри пакета numpy;
д) тестування швидкості роботи отриманої реалізації порівняно із наївним підходом.

Розглянувши результати тестування можна зробити висновок, що використання матричних операцій дозволило покращити наївну реалізацію, але потребує оптимізації. Поліпшити результати можна, якщо, наприклад, застосувати спеціальні математичні методи для роботи з великими матицями.

Також розроблений метод було успішно застосовано для задачі виявлення спільнот.

ПЕРЕЛІК ПОСИЛАНЬ

1. Добровольський Г. А. Модель, метод та інформаційна технологія відбору наукових публікацій у процесі підготовки бібліографічного покажчика : дис. доктора філософії : 05.13.06. Харків, 2021. 216 с.

2. Rokach, Lior, Oded Maimon. Clustering methods. Data mining and knowledge discovery handbook. Springer US, 2005. P. 321–352.

3. DeepDPM: Deep Clustering With an Unknown Number of Clusters URL : https://arxiv.org/pdf/2203.14309.pdf (дата звернення: 10.02.23).

4. Deng, L.; Yu, D. Deep Learning: Methods and Applications URL : http://research.microsoft.com/pubs/209355/DeepLearning-NowPublishing-Vol7-SIG-039.pdf (дата звернення: 10.02.23).

5 William L. Hamilton Inductive Representation Learning on Large Graphs URL : https://cs.stanford.edu/people/jure/pubs/graphsage-nips17.pdf (дата звернення: 10.02.23).

6 Thomas N. Kipf Semi-Supervised Classification with Graph Convolutional URL : Networks https://arxiv.org/pdf/1609.02907.pdf (дата звернення: 10.02.23).

7 Xiaodan Liang Semantic Object Parsing with Graph LSTM URL : https://arxiv.org/pdf/1603.07063.pdf (дата звернення: 10.02.23).

8 Alvaro Sanchez-Gonzalez Graph Networks as Learnable Physics Engines for Inference and Control URL : https://arxiv.org/pdf/1806.01242.pdf (дата звернення: 10.02.23).

9 Peter W. Battaglia Interaction Networks for Learning about Objects, Relations and Physics URL : https://arxiv.org/pdf/1612.00222.pdf (дата звернення: 10.02.23).

10 Alex Fout Protein Interface Prediction using Graph Convolutional Networks URL : https://proceedings.neurips.cc/paper_files/paper/2017/file/f507783927f2ec2737ba40a fbd17efb5-Paper.pdf (дата звернення: 10.02.23).

11 Takuo Hamaguchi Knowledge Transfer for Out-of-Knowledge-Base Entities: A Graph Neural Network Approach URL : https://www.ijcai.org/Proceedings/2017/0250.pdf (дата звернення: 10.02.23).

12 Hanjun Dai Learning Combinatorial Optimization Algorithms over Graphs URL : https://arxiv.org/pdf/1704.01665.pdf (дата звернення: 10.02.23).

13. Kohonen, Teuvo Intro to SOM. URL : http://www.cis.hut.fi/projects/somtoolbox/theory/somalgorithm.shtml (дата звернення: 10.02.23).

14 Mathilde Caron, Piotr Bojanowski, Julien Mairal, and Armand Joulin. Unsupervised pre-training of image features on non-curated data. URL : https://arxiv.org/abs/1905.01278 (дата звернення: 10.02.23).

15 Christos Avgerinos, Vassilios Solachidis, Nicholas Vretos,and Petros Daras. Non-parametric clustering using deep neural networks. URL : https://ieeexplore.ieee.org/document/9171232 (дата звернення: 10.02.23).

16. N. Gillis Introduction to nonnegative matrix factorization. URL : arxiv.org/abs/1703.00663 (дата звернення: 10.02.23).

17. M. Udell, C. Horn, R. Zadeh, S. Boyd, et al. Generalized low rank models. Foundations and Trends in Machine Learning, 2016. vol. 9, no. 1, P. 1–118.

18 Wang, Y., Bu, Z., Yang, H., Li, H.-J., Cao, J. An effective and scalable overlapping community detection approach: Integrating social identity model and game theory. URL : https://www.researchgate.net/publication/344171415_An_effective_and_scalable_overlapping_community_detection_approach_Integrating_social_identity_model_and_game_theory (дата звернення: 10.02.23).

19 Z. Bu, H. Li, C. Zhang, J. Cao, A. Li, Y. Shi, Graph k-means based on leader identification, dynamic game, and opinion dynamics URL : https://ieeexplore.ieee.org/document/8662615 (дата звернення: 10.02.23).

20 V.D. Blondel, J. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks URL : http://refhub.elsevier.com/S0096-3003(20)30556-7/sbref0013 (дата звернення: 10.02.23).

21. Hoyer, P. O. Non-negative sparse coding. Proceedings of the 2002 12th IEEE Workshop, IEEE, 2002. P. 557–565

22 Python URL : https://www.python.org/doc/ (дата звернення: 10.02.23).

23. Quan Nguyen Mastering Concurrency in Python: Create faster programs using concurrency, asynchronous, multithreading, and parallel programming URL : http://onreader.mdl.ru/MasteringConcurrencyInPython/content/index.html (дата звернення: 30.10.22).

24 NumPy URL : https://numpy.org/ (дата звернення: 10.02.23).