НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

До захисту допущено:

Завідувач кафедри

_____ Вадим МУХІН

«__30_»_____05_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою

“Інтелектуальні сервіс-орієнтовані розподілені обчислювання”

зі спеціальності 122 "Комп'ютерні науки"

на тему: «Кластеризація текстових документів на основі методу k-найближчих сусідів»

Виконав:
студент IV курсу, групи ДА-91
Мельник Антоній Михайлович _____

Керівник:
Професор, д.т.н.
Рогоза Валерій Станіславович _____

Консультант з нормконтролю:
доцент, к.т.н.
Кирюша Б.А. _____

Рецензент:
Доц., к.т.н.
Тимощук Н.М. _____

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

Студент _____

Київ – 2023

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 "Комп'ютерні науки"

Освітньо-професійна програма – "Інтелектуальні сервіс-орієнтовані розподілені обчислювання"

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Вадим МУХІН

«_30__»_____05_____2023 р.

ЗАВДАННЯ

на дипломну роботу студенту

Мельник Антонія Михайловича

1. Тема роботи «Кластеризація текстових документів на основі методу k-найближчих сусідів», керівник роботи Рогоза Валерій Станіславович, професор, д.т.н., затверджені наказом по університету від «_30__»_травня_____ 2023__ р. № 2065-с_____

2. Термін подання студентом роботи – 20 червня 2023 р.

3. Вихідні дані до роботи

Розробка алгоритму на основі методу k-найближчих сусідів.

Розробка додатка для кластеризації текстових документів на основі розробленого алгоритму

4. Зміст роботи

1. Вступ.
2. Порівняння методів кластеризації.

3. Поняття алгоритму кластеризації.
4. Вибір мови програмування та бібліотечних рішень.
5. Розробка алгоритму на обраній мові програмування.
6. Підбір текстових документів для тестування програми.
7. Виміри продуктивності алгоритму.
8. Функціонально-вартісний аналіз програмного забезпечення.
9. Висновки.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Презентація до захисту роботи.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В.		

7. Дата видачі завдання _____

Календарний план

з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	10.02.2023	
2	Проведення порівняльного аналізу кластеризації даних	15.04.2023	
3	Побудова алгоритму кластеризації текстових документів методом k-найближчих сусідів	30.04.2023	
4	Вибір мови програмування та аналіз бібліотек рішень	03.05.2023	
5	Написання та налагодження програми	12.05.2023	

6	Підбір тестових прикладів	15.05.2023	
7	Виконання експериментальних обчислень з метою підтвердження працездатності алгоритму	29.05.2023	
8	Оформлення дипломної роботи	15.06.2023	
9	Отримання допуску до захисту та подача документів в ДЕК	17.06.2023	

Студент Мельник А.М.

Керівник Рогоза В.С.

АНОТАЦІЯ

бакалаврської дипломної роботи Мельник Антонія Михайловича на тему «Кластеризація текстових документів на основі методу k-найближчих сусідів»

Метою дипломної роботи є дослідження методу k-найближчих сусідів, а також створення на його основі алгоритму кластеризації текстових документів, також слід провести аналіз інших методів кластеризації, а також можливості адаптувати метод k-найближчих сусідів для кластеризації даних, оскільки зазвичай він використовується для класифікації даних, а не кластеризації.

Впродовж дослідження даної теми, біло виявлено відсутність будь-яких напрацювань, щодо алгоритму кластеризації на основі методу k-найближчих сусідів. Тому вважаю дослідження в цій області, і розробку відповідного алгоритму цілком виправданими.

Також окрім створення відповідного алгоритму, тема моєї дипломної роботи включає досить важливе уточнення, а саме кластеризацію саме текстових документів, що також під собою включає ряд інших запитань, а саме препроцесінг тексту, для кращої кластеризації наших вхідних даних.

Не зважаючи на те що в області препроцесінгу тексту як для кластеризації так і класифікації текстових документів, вже є багато готових рішень, втім підбір правильної комбінації цих функцій є також досить важливим.

В результаті виконання нашої дипломної роботи ми розробимо новий алгоритм для кластеризації, а також розробимо додаток, що буде реалізовувати кластеризації текстових документів на основі розробленого алгоритму.

В кінці буде проведена оцінка працездатності цього алгоритму, на підібраному наборі текстових даних, яким буде також приділена увага. А також візуалізувавши наші тестові дані, ми оцінимо, чи коректно працює наш алгоритм.

Створений додаток та алгоритм можна буде використовувати в подальших дослідженнях в області кластеризації даних, а також не зважаючи на відносну трудоємкість цього алгоритму, використовувати в певних задачах в області кластеризації текстових даних.

Загальний обсяг роботи 98 с., 14 рис., 10 таблиць, 3 додатки, 17 джерела.

Ключові слова: кластеризація, k-найближчих сусідів, додаток для кластеризації, текстові документи.

Annotation

Melnyk Antonii Mykhailovich's bachelor's thesis on "Clustering of text documents based on the k-nearest neighbors method"

The purpose of the thesis is to study the method of k-nearest neighbors, as well as to create an algorithm of clustering text documents based on it, and also to analyze other methods of clustering, and the possibility of adapting the method for k-clustering data, as it is usually used to classify data rather than cluster.

Thro the study of this topic, the absence of any findings was revealed regarding the cluster algorithm based on the method of k-nearest neighbors. Therefore, I consider research in this area, and the development of the appropriate algorithm, quite justified.

Also, in addition to creating the corresponding algorithm, the topic of my thesis includes a very important clustering, namely the clustering of text documents, which also includes a number of other questions, i.e. text preprocessing, to better cluster our input data.

Despite the fact that in the field of text preprocessing for both clustering and classification of text documents, there are already many ready-made solutions, however, the selection of the right combination of these functions is also quite important.

As a result of our thesis, we will develop a new algorithm for clustering, as well as develop an application that will implement clusters of text documents based on the developed algorithm.

At the end, the performance of this algorithm will be assessed, on the selected set of text data, which will also be paid attention. And also by visualizing our test data, we evaluate whether our algorithm works correctly.

The created application and algorithm can be used in further research in the field of data clustering, as well as, despite the relative labor-intensive of this algoritm, to use in certain tasks in the area of text data clustering.

Total volume of work 98 s., 14 figures, 10 tables, 3 appendices, 17 sources.

Keywords: cauterization, k-nearest neighbors, cluster application, text documents.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

K-NN (k-Nearest Neighbors) – метод к-найближчих сусідів.

SOM (Self-Organizing Map) – нейронна мережа з некерованим навчанням, яка використовується для конструювання багатовимірного простору в простір з нижчою розмірністю (найчастіше, двовимірний).

Python – інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією.

scikit-learn – це безкоштовна програмна бібліотека машинного навчання для мови програмування Python, яка надає функціональність для створення та тренування різноманітних алгоритмів класифікації, регресії та кластеризації.

NLTK (Natural Language Toolkit) – популярна бібліотека для обробки природної мови на Python.

TF-IDF – (від англ. TF — term frequency, IDF — inverse document frequency) — статистичний показник, що використовується для оцінки важливості слів у контексті документа, що є частиною колекції документів чи корпусу.

NumPy (Numerical Python) - це бібліотека для мови програмування Python, яка надає підтримку для великих, багатовимірних масивів та математичних функцій, що дозволяє ефективно виконувати обчислення. Вона використовується для наукових обчислень, обробки даних, машинного навчання та багатьох інших сфер, де важлива швидкість та ефективність роботи з масивами даних.

Matplotlib - це бібліотека для мови програмування Python, яка використовується для візуалізації даних. Вона надає широкий спектр функцій і інструментів для створення різних видів графіків,

діаграм, графічних представлень та інших візуальних елементів. Matplotlib часто використовується в наукових дослідженнях, аналізі даних, статистиці, машинному навчанні та інших областях, де потрібна візуалізація даних.

XML (Extensible Markup Language) є розширюваним мовним засобом маркування. Він використовується для представлення та обміну даними у структурованому форматі.

BeautifulSoup є бібліотекою для парсингу HTML і XML документів у мові програмування Python.

PCA (Principal Component Analysis) - це метод зменшення розмірності даних шляхом перетворення їх у новий набір компонентів, які відображають найбільшу дисперсію в даних. Він використовується для спрощення аналізу даних та візуалізації, зберігаючи при цьому якомога більше інформації.

t-SNE (t-Distributed Stochastic Neighbor Embedding) - це алгоритм зменшення розмірності даних, який використовується для візуалізації складних наборів даних у низькорозмірному просторі.

UMAP (Uniform Manifold Approximation and Projection) - алгоритм зменшення розмірності даних для візуалізації та аналізу. Швидкий та ефективний у роботі з великими наборами даних, зберігає структуру та відстані між точками в проекції. Використовується в науці про дані та машинному навчанні.

ВСТУП

Метод k-найближчих сусідів (k-Nearest Neighbors, k-NN) - це простий метод, який використовується для алгоритмів класифікації та регресії, що використовується в машинному навчанні. У кластеризації, k-NN використовується для визначення приналежності нового прикладу до певного класу шляхом порівняння його з k найближчими сусідами у навчальному наборі даних. В регресії, використовується середнє (або медіана) значення k найближчих сусідів для передбачення значення вихідної змінної.

Історично, метод k-найближчих сусідів був розроблений в 1960-х роках і був одним з перших алгоритмів машинного навчання. Його простота та інтуїтивність зробили його популярним для початківців та основою для подальших розвинених методів. Крім класифікації та регресії, k-NN також може використовуватися для знаходження схожих об'єктів у рекомендаційних системах або для згладжування шуму в даних.

Однак, у методу k-найближчих сусідів є деякі обмеження. Він може бути чутливим до шуму та варіантності в даних. Також, велике значення k може призвести до згладжування міжкласових меж, тоді як мале значення k може бути вразливим до випадкових аномалій. Необхідно уважно вибирати значення k та підготовляти дані для досягнення кращих результатів з цим методом.

Тому впродовж роботи варто буде приділити достатньо уваги для підготовки даних, щоб наш алгоритм працював належним чином і був менш вразливим до випадкових аномалій.

Втім як було сказано вище, цей метод використовувався і використовується для завдань класифікації та регресії. Втім в області кластеризації даних цей метод не отримав достатнього розповсюдження. Тому і було прийнято рішення дослідити цю

область для алгоритму к-найближчих сусідів. Розробивши відповідний алгоритм, та додаток що буде кластеризувати текстові документи на основі розробленого алгоритму.

А також враховуючи що метод k-найближчих сусідів лежить в основі інших більш досконалих алгоритмів, маю сподівання, що розроблений алгоритм надалі зможе мати продовження розвитку в області кластеризації даних.

1. ПОРІВНЯННЯ МЕТОДІВ КЛАСТЕРИЗАЦІЇ

1.1. Опис методів кластеризації даних

Починаючи наше дослідження методу кластеризації за допомогою підходу k-найближчих сусідів, я неодмінно звернув увагу на альтернативні методи кластеризації, які були доступні. Необхідно відзначити, що існує безліч різних методів кластеризації, і вивчення всіх цих методів, порівнюючи кожен з ними з підходом k-найближчих сусідів, є практично неможливим завданням. З цього приводу було обрано п'ять методів, на які зосередимося у подальшому дослідженні.

Проте перед тим, як ми підемо у глибину цієї захоплюючої подорожі кластеризаційними методами, я бажаю провести огляд нашого методу k-найближчих сусідів, надаючи йому відповідне визначення та короткий опис його особливостей, навіть не зважаючи на те що цей метод зазвичай застосовується саме для класифікації, а не кластеризації:

А) Метод k-найближчих сусідів

Метод класифікації к-найближчих сусідів (k-NN) є одним з найпростіших та найпоширеніших методів класифікації. Його основна ідея полягає у тому, що кожен документ відноситься до того ж класу, що і його k найбільш близьких сусідів за заданою метрикою схожості. Приклад як відносять новий екземпляр до одного з класів зображено нижче (Рис. 1.1.1.). Кількість k може бути вибрана заздалегідь або визначена емпірично.

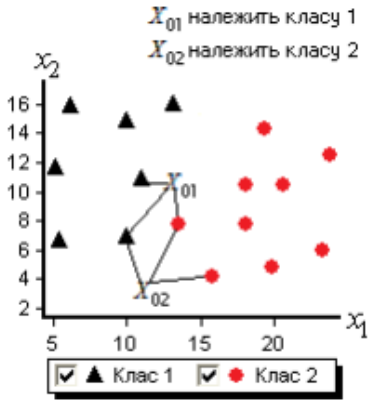

Рис. 1.1.1. - Класифікація нового екземпляра до одного з класів

Переваги методу к-найближчих сусідів полягають у простоті реалізації та відсутності необхідності в попередній підготовці даних. Крім того, цей метод може давати хороші результати для невеликих обсягів даних або для даних, що мають просту структуру.

Однак, метод к-найближчих сусідів має деякі недоліки. Наприклад, він може бути дуже чутливим до викидів (outliers) та шуму в даних, що може призводити до неадекватних результатів. Крім того, цей метод може бути невідповідним для великих обсягів даних, оскільки він може вимагати значних обчислювальних ресурсів для знаходження найближчих сусідів, а також великих обсягів пам’яті оскільки потребує зберігання одразу всієї колекції документів у пам’яті.

Також мені стало цікаво у яких випадках зазвичай використовують цей метод класифікації даних. І мені вдалось відшукати, що метод k-найближчих сусідів найчастіше використовується для даних з простою структурою, а отже він може дуже непогано підійти для кластеризації документів, оскільки метрики схожості, такі як косинусна міра схожості

(1.1.1), можуть бути використані для порівняння векторів термів, що відображають вміст документів.

$$\text{cosine simularity} = S_C(A, B) = \cos\theta = \frac{A*B}{\|A\|*\|B\|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i{}^2} * \sqrt{\sum_{i=1}^{n} B_i{}^2}} \#(1.1.1)$$

Де A, B - вектори параметрів двох об’єктів

Однак, при використанні методу к-найближчих сусідів для кластеризації текстових документів варто мати на увазі його обмеження. Зокрема, метод к-найближчих сусідів може бути недостатньо ефективним для великих колекцій документів, оскільки він може бути дуже чутливим до розміру та розподілу кластерів. Крім того, він може не давати задовільних результатів для текстів, що містять багато спільних слів або для текстів, що мають складну структуру.

Б) Агломеративна кластеризація

Агломеративна кластеризація - це метод кластеризації даних, що полягає у поєднанні дрібних кластерів у більші на основі схожості між ними. Значення схожості між двома кластерами зазвичай визначається за допомогою якоїсь метрики, такої як відстань між центроїдами кластерів або середньої відстані між елементами кластерів.

Особливістю агломеративної кластеризації є те, що на кожному кроці алгоритму об'єднують два найбільш схожих кластери, поки не залишиться тільки один кластер, який містить усі дані. Це дає змогу визначити оптимальну кількість

кластерів, використовуючи деревоподібну структуру кластерів, яка отримується під час агломеративної кластеризації.

Однією з переваг агломеративної кластеризації є те, що вона проста у реалізації й може бути використана з різними метриками схожості та розмірами даних. Проте вона має певні недоліки, такі як складність обчислень та висока чутливість до випадкових шумів в даних. Також, вона може давати неоптимальні результати для складних даних, які можуть мати несферичну форму кластерів або різний розмір і щільність.

В) Алгоритм k-середніх

Алгоритм k-середніх - це метод кластеризації, який розділяє набір даних на кілька кластерів шляхом знаходження k центрів кластерів (k - це заданий параметр), що мінімізують суму квадратів відстаней між кожним елементом даних і його найближчим центром кластера.

Основні кроки алгоритму k-середніх:

- Випадково ініціалізуємо k центрів кластерів.
- Для кожного елемента даних знаходимо найближчий центр кластера і додаємо його до цього кластера.
- Розраховуємо нові центри кластерів на основі середнього значення всіх елементів, які належать до кожного кластера.
- Повторюємо кроки 2 і 3 до тих пір, поки кластери не стабілізуються.

Особливості алгоритму k-середніх:

- Метод є дуже ефективним для кластеризації даних великих обсягів.

- Результати кластеризації можуть залежати від випадкової ініціалізації початкових центрів кластерів.
- Алгоритм може застрягати у локальних мінімумах, тому що він може знаходити лише найближчі центри кластерів, а не найкращі.
- Число кластерів k повинно бути відомим заздалегідь, що може бути проблемою у деяких випадках.

Г) Кластеризація на основі карти Кохонена

Кластеризація на основі карти Кохонена (Self-Organizing Map або SOM) - це метод машинного навчання без вчителя, який використовується для кластеризації даних і візуалізації багатовимірних даних у двовимірному просторі. Основна ідея полягає в тому, щоб зображати кожен зразок даних як точку у двовимірному просторі, так щоб близькі зразки даних були розташовані біля один одного, а віддалені зразки даних були розташовані далеко один від одного.

Карта Кохонена складається з групи вузлів (нейронів), розташованих на двовимірній сітці. Кожен вузол має вагові коефіцієнти, які описують його позицію на карті. На початку процесу навчання вагові коефіцієнти вузлів генеруються випадково або на основі попередньо заданих значень.

Процес навчання включає ітерації, під час яких карта Кохонена зміщується для того, щоб максимально відповідати вихідним даним. На кожній ітерації вибирається зразок даних, і кожен вузол карті Кохонена, чий вектор вагових коефіцієнтів найбільш близький до вектора зразка даних, рухається в напрямку цього зразка.

Коли процес навчання завершується, кожен зразок даних призначається до найближчого вузла карти Кохонена, і кластеризація даних може бути виконана на основі розташування вузлів.

Особливості методу кластеризації на основі карти Кохонена полягають у високій швидкості обчислень і вмінні відобразити великі обсяги даних.

1.2. Порівняння методів кластеризації даних

Порівняння різних методів кластеризації є дуже важливим етапом моєї дипломної роботи. Оскільки наразі не існує універсального й умовно ідеального методу кластеризації варто розглянути переваги та недоліки методів один перед одним, щоб дізнатись на які особливості та переваги методу к-найближчих сусідів варто робити ставку. Всі вище описані методи кластеризації були підібрані так, щоб вони максимально відрізнялись один від одного за принципом дії.

Далі думаю варто коротко виділити переваги кожного з методів, на мою думку найкраще для цього підійде табличка. Проте я не буду поглиблюватись і детально розбирати переваги кожного з методів у цьому щотижневому звіті оскільки моментів досить багато і вміщати все це в один звіт не бачу доцільним. До того ж хочеться зосередитись методі к-найближчих сусідів та його модифікаціях.

Перейдемо до порівняльної таблички (Таблиця 1)

Таблиця 1- Порівняння різних алгоритмів обробки даних

Алгоритм	Переваги	Недоліки	У яких областях використовується	Складність
Агломеративна кластеризація	Добре підходить для невеликої кількості кластерів; здатна працювати з різними метриками відстані; можливість візуалізації дерева кластерів	Чутливість до початкових умов; висока обчислювальна складність для великих наборів даних; проблеми з масштабуванням	Біоінформатика, соціальні науки, аналіз мовленнєвих даних	O(n^3) N - кількість даних
К-середніх	Швидкість виконання; простота реалізації; хороша ефективність при великій кількості даних; здатність роботи з різними метриками відстані	Чутливість до початкових умов; може вести до поганих результатів, якщо групи мають складну форму; відносно низька точність	Машинне навчання, аналіз ринку, кластеризація зображень	O(n * k * i) n - кількість даних, k - кількість кластерів, i - кількість ітерацій

K-найближчих сусідів	Простий у реалізації та розумінні; Ефективний для невеликих наборів даних; Добре працює з нерегулярними формами кластерів	Чутливий до шуму, не ефективний на великих наборах даних	Розпізнавання образів, голосовий та текстовий аналіз, біоінформатика	O(n^2) N - кількість даних
Кластеризація на основі карти Кохонена	Допомагає візуалізувати структуру даних, показує хороші результати на даних з високою розмірністю	Потребує підбору параметрів, не ефективний на великих наборах даних, кластери можуть перекриватись	Рекомендується для візуалізації даних та аналізу даних з високою розмірністю, наприклад, у текстовому майнінгу та обробці зображень.	O(N * M * K), де N - кількість об'єктів у вхідному наборі даних, M - кількість епох K - кількість клітинок

1.3. Підсумки порівняння

Вище було розглянуто декілька алгоритмів кластеризації даних, насправді впродовж тижня я розглянув дещо більшу кількість алгоритмів кластеризації даних, але для звіту обрав саме ці, оскільки мені вони здались найбільш різноманітними.

Розглянувши принципи роботи цих алгоритмів ми побачили наскільки алгоритми бувають різноманітними та цікавими за принципом роботи, від найбільш інтуїтивного методу, на мій погляд, к-середніх, до найбільш складного і заплутаного, кластеризації на основі карти Кохонена.

Але за цією різноманітністю варто не забути основну мету, навіщо ми розглядали всі ці алгоритми. Основним завданням для мене було виділити переваги саме алгоритму к-найближчих сусідів для кластеризації текстових документів.

Тому зараз я хочу коротко пояснити які переваги цього алгоритму я виділив для себе. По-перше, це його простота в розумінні та інтуїтивність. По-друге, якщо порівнювати з іншим простим в розумінні та в обчислювальній складності алгоритмом к-середніх, я для себе виділив особливість, що на відміну від к-середніх нам не потрібно вказувати кількість кластерів. Як на мене, ця особливість дуже збільшує область використання саме цього алгоритму. По-третє, ще обчислювальна складність, хоч він і не є оптимальним, і для його роботи нам потрібно при кожній ітерації доступ до всієї бази документів, втім він все ще досить простий. По-четверте, цей метод з певними модифікаціями вже широко використовується для класифікації текстових даних.

Підсумовуючи, к-найближчих сусідів гарний алгоритм для розуміння, що на мою думку в майбутньому дозволить внести певні власні модифікації до алгоритму, для покращення ефективності саме

для моїх задач, а розповсюдженість у використанні цього алгоритму у класифікації текстових даних означає що вже існує безліч модифікацій та покращень алгоритму для такого типу даних.

1.4. Висновки до розділу

У цьому звіті розглядається декілька алгоритмів кластеризації даних. Насправді було розглянуто кілька інших алгоритмів кластеризації даних, але обрано саме ці для доповіді, оскільки вони, здається, пропонують найбільше розмаїття.

Вивчивши принципи роботи цих алгоритмів, я зрозумів, наскільки вони різноманітні та цікаві у своїй роботі: від k-середніх, які, на мою думку, є найбільш інтуїтивно зрозумілим методом, до найскладнішої та найзаплутанішої кластеризації на основі карт Кохонена.

Але за цим розмаїттям не варто забувати про головну мету, заради якої розглядаються всі ці алгоритми. Моїм головним завданням було висвітлити переваги алгоритму k-найближчих сусідів для кластеризації текстових документів.

Зараз я хотів би коротко описати переваги, які я особисто виділив для цього алгоритму. По-перше, він простий для розуміння та інтуїтивно зрозумілий. По-друге, хоча обчислювальна складність не є оптимальною і вимагає доступу до всієї бази документів на кожній ітерації для його роботи, він все одно є дуже простим. По-четверте, цей метод, з деякими модифікаціями, вже широко використовується для класифікації саме текстових даних.

Таким чином, алгоритм k-найближчих сусідів є простим для розуміння алгоритмом, і я вважаю, що цей алгоритм можна модифікувати в майбутньому, щоб підвищити його ефективність для

моєї задачі. Крім того, той факт, що цей алгоритм широко використовується для кластеризації текстових даних, означає, що вже зроблено багато модифікацій і вдосконалень алгоритму для цього типу даних.

2. ПОБУДОВА АЛГОРИТМУ КЛАСТЕРИЗАЦІЇ

2.1. Параметри для кластеризації текстових документів

Існує безліч варіантів вибору параметрів за якими можна кластеризувати текстові документи. Втім в даному розділі я хочу звернути увагу на важливості вибору правильних параметрів для коректної кластеризації документів.

Розпочнемо з хорошого прикладу класифікації даних, який я знайшов як приклад для методу к-найближчих сусідів. З нього можна побачити наскільки важливо правильно підібрати параметри по яких будемо відносити об’єкт до того чи іншого класу. Хоча й суть моєї дипломної роботи полягає в тому, що потрібно розробити алгоритм для кластеризації, а не для класифікації, але підбір правильних параметрів важливий для всіх задач в області машинного навчання.

Суть цього прикладу полягає в тому, що у нас є список пасажирів лайнера з великою кількістю параметрів, нам треба визначити чи буде врятований кожен з пасажирів, чи ні. І експериментуючи яка кількість параметрів кожного пасажира є найкращою для хорошої кластеризації, виявилось, що достатньо всього 3 параметри, а саме вік, стать, вартість квитка. Врахування інших параметрів або давали мінімальний приріст точності класифікації на тренувальному наборі даних, або взагалі не давали ніякого приросту, як от, наприклад поле з ім’ям пасажира, або іншими паспортними даними.

Але все ж наша задача дещо відрізняється від описаного вище прикладу, оскільки ми розглядаємо текстові документи, у яких нема таких явних параметрів для кластеризації.

Почнемо розглядати приклади документів які ми будемо кластеризувати. Класичним прикладом можна назвати кластеризацію

повідомлень на електронній пошті. Тобто за допомогою методу кластеризації. В такому випадку я б для початку ввів би параметр довжини тексту, насиченість тексту розділовими знаками, а особливо знаками оклику, дужечками тощо. Далі варто розглянути алгоритм токенізації, де за один токен можна взяти одне слово. Втім повертаючись до того прикладу документів, яких я б хотів кластеризувати, то нам вищеописані методи не будуть дуже корисними. Адже я хочу кластеризувати документи за сенсовим навантаженням, а отже для нас корисним буде наступні варіанти.

Лічильник терміну в документах, де ми будемо групувати однокореневі слова і записувати їх частоту в документі. Також варто буде розглянути приріст при попередньому використанні алгоритмів стоп-лістингу і лематизації. Також в майбутньому буде розглянуто алгоритми вибірки ознака TF-IDF який і буде лежати в основі нашого алгоритму.

2.2. Вибір метрик для кластеризації

Насправді існує безліч метрик відстаней для кластеризації текстових документів. Все залежить від того які документи ми будемо кластеризувати. Також можна комбінувати ці метрики для досягнення кращого результату.

Я вважаю за потрібне остаточно визначитись з конкретними метриками або комбінаціями метрик вже на етапі реалізації алгоритму, щоб була можливість проекспериментувати з конкретними наборами даних.

В даному розділі я хотів би виділити декілька метрик, і коротко ознайомитись з принципами їх дії та коли їх краще використовувати. Поки я знайшов для себе 4 варіанти метрик про які буде йтись далі.

Коли використовується алгоритм k-NN для кластеризації текстових документів, можна використовувати різні метрики для визначення відстаней між документами. Ось декілька рекомендованих метрик:

- **Косинусна схожість** (2.2.1): це метрика, яка вимірює кут між векторами документів в просторі ознак. Вона часто використовується для порівняння текстових документів, оскільки зазвичай документи містять багато різних слів, і ця метрика дозволяє ігнорувати масштаби векторів та зосереджуватися на їхніх напрямках.

$$\text{cosine simularity} = S_C(A, B) = \cos\theta = \frac{A*B}{\|A\|*\|B\|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} * \sqrt{\sum_{i=1}^{n} {B_i}^2}} \#(2.2.1)$$

Де A, B - вектори параметрів двох об’єктів

- **Відстань Жаккара** (2.2.2): це метрика, яка вимірює схожість між множинами термів двох документів. Вона використовується для порівняння документів, що містять невелику кількість термів, таких як теги або ключові слова.

$$\text{sim}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \#(2.2.2)$$

Де X, Y - множини двох об’єктів

- **Евклідова відстань**(2.2.3): це метрика, яка вимірює відстань між векторами документів в n-вимірному просторі ознак. Вона використовується для порівняння документів, які містять невелику кількість ознак, які можуть бути представлені числовими значеннями.

$$\text{dist}(P, Q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \quad \#(2.2.3)$$

Де P, Q - вектори ознак двох об’єктів, і мають вигляд, $P = (p_1, p_2, \ldots, p_n) Q = (q_1, q_2, \ldots, q_n)$

- **Манхеттенська відстань**(2.2.4): це метрика, яка вимірює суму абсолютних різниць між ознаками двох документів. Вона також використовується для порівняння документів, які містять невелику кількість ознак, і може бути корисною, коли важливіше знаходити різниці між документами, а не їх схожість.

$$\text{dist}(P, Q) = \|p - q\| = \sum_{i=1}^{n} |p_i - q_i| \quad \#(2.2.4)$$

Де P, Q - вектори ознак двох об’єктів, і мають вигляд, $P = (p_1, p_2, \ldots, p_n) Q = (q_1, q_2, \ldots, q_n)$

Вибір метрики залежить від характеру даних та особливостей проблеми, яку намагаєтеся вирішити.

2.3. Алгоритм кластеризації на основі методу k-NN

Почнемо з того що згадаємо нашу основну проблему, як постає перед нами при створенню алгоритму кластеризації на основі методу к-найближчих сусідів.

Головною проблемою є те що цей метод зазвичай використовується саме для класифікації даних, тобто при класифікації у нас вже є дані, які розбиті на декілька класів, а ми вже потім даємо алгоритму нові дані, які він буде класифікувати. У нас же такої можливості нема. Нам потрібно якось почати кластеризувати дані з нуля.

Отже, якщо розбити вище описану проблему на менші більш конкретні проблеми, що нам потрібно вирішити, то перше, це де нам

взяти К сусідів для того, щоб кластеризувати наступні дані. Тобто перша проблема полягає в тому як буде розпочинатись цей алгоритм. Першим що спадає на думку, це випадковим чином обрати декілька екземплярів максимально віддалених один від одного і кожному дати мітку власного кластера, далі обрати декілька максимально наближених документів до кожного кластеру, за методом k-means, і коли у нас буде кластеризовано достатньо даних, тоді вже починати алгоритм k-NN що буде працювати за схожим принципом, як і при класифікації даних.

Перейдемо до орієнтовного алгоритму кластеризації даних, що будемо реалізовувати потім. Варто зазначити, що цей алгоритм може бути модифікований відштовхуючись від ефективності та дієздатності на етапі реалізації на мові програмування. Паралельно з розробкою алгоритму я займаюсь реалізацією та експериментами на Python. Я розглянув декілька варіантів як можна реалізувати цей алгоритм, але поки найбільш дієздатним і ефективним на мою думку є наступний алгоритм.

Алгоритм кластеризації:

1. Вибрати параметр k - кількість найближчих сусідів.
2. Вибрати метрику відстані.
3. Перетворити кожен об'єкт у вектор ознак.
4. Створити порожній список кластерів.
5. Для кожного об'єкта дати йому випадковий кластер.
6. Для кожного об'єкта визначити k найближчих сусідів за обраною метрикою відстані.
7. Для кожного об'єкта порівняти його з його k найближчими сусідами та визначити найчастіший кластер серед цих сусідів.
8. Присвоїти кожному об'єкту кластер, який отримали в результаті попереднього кроку.

9. Якщо кластери змінились після останньої ітерації, повторити кроки 6-8.
10. Кластеризація завершена, повернути отримані кластери.

2.4. Висновки до розділу

На мою думку, розроблений алгоритм кластеризації даних на основі методу k-NN є ефективним і простим у реалізації. Він дозволяє класифікувати об'єкти на основі схожості з найближчими сусідами в просторі ознак.

Для використання даного алгоритму кластеризації даних на основі методу k-NN з текстовими документами, необхідно перед кластеризацією підготувати документи. Це може містити очищення тексту від зайвих символів, токенізацію, векторизацію, тобто перетворення документів у вектори числових ознак, що відображають семантичне значення документа.

Вибір метрики для визначення відстаней між документами залежить від конкретної задачі й типу даних. Для текстових даних можна використовувати косинусну схожість або евклідову відстань, Манхеттенську відстань, або відстань Жаккара між векторами ознак. Важливо також враховувати розмірність векторів та можливість нормалізації ознак, що дозволяє зменшити вплив деяких ознак на загальну відстань між документами.

На даний момент представлений алгоритм кластеризації даних на основі методу k-NN виглядає ефективним і простим у реалізації, але це все ще варто буде перевірити на етапі розробки. Також варто відзначити що дуже важливими моментами при розробці даного алгоритму для використання з текстовими даними необхідна попередня підготовка документів та вибір підходящої метрики для визначення відстаней між документами.

3. ВИБІР МОВИ ПРОГРАМУВАННЯ ТА БІБЛІОТЕЧНИХ РІШЕНЬ

3.1. Вибір мови програмування

Для реалізації алгоритму кластеризації на основі методу k-NN можна використовувати різні мови програмування, такі як Python, R, Java, C++, і т.д.

Але все-таки мій вибір впав саме на Python, далі коротко аргументую чому саме.

Python - одна з найпопулярніших мов програмування для машинного навчання та аналізу даних за допомогою кластеризації. Ось деякі переваги використання Python для кластеризації текстових документів на основі методу k-NN:

- Велика кількість ресурсів та документації: Python має велику спільноту програмістів і дослідників, які займаються машинним навчанням і аналізом даних, тому ви можете знайти безліч матеріалів і ресурсів, пов'язаних з кластеризацією текстових даних на основі методу k-NN.
- Бібліотеки машинного навчання та аналізу даних: Python має велику кількість бібліотек для роботи з машинним навчанням та аналізом даних, таких як scikit-learn, TensorFlow, PyTorch тощо. Scikit-learn, зокрема, містить багато реалізацій алгоритмів кластеризації, в тому числі k-NN.
- Зручний синтаксис: Python має досить простий і зрозумілий синтаксис, що дозволяє швидко і легко розробляти програми для обробки текстових даних.
- Розширюваність: Python дозволяє розширювати функціональність за допомогою різноманітних бібліотек та модулів. Це дозволяє програмістам швидко і легко розширювати функціональність своєї

програми, якщо потрібно додати нові функції або алгоритми.

Тому використання Python для розробки алгоритму кластеризації текстових документів на основі методу k-NN є розумним вибором з багатьох причин, про які я вже згадував. Зокрема, Python має велику кількість бібліотек і фреймворків, які забезпечують швидку і легку розробку проекту. Крім того, деякі бібліотеки машинного навчання та аналізу даних, такі як scikit-learn, надають реалізації алгоритмів кластеризації, включаючи k-NN, що дозволяє програмістам використовувати готові рішення замість того, щоб розробляти все з нуля.

Нарешті, Python має дуже зручний синтаксис, який спрощує розробку та підтримку коду. Це зробить процес розробки алгоритму простішим і зрозумілішим.

3.2. Вибір бібліотек для реалізації алгоритму

Для кластеризації текстових документів на основі методу k-NN я обрав використовувати бібліотеку scikit-learn, яка є однією з найпопулярніших бібліотек для машинного навчання та аналізу даних на Python.

Scikit-learn має багатий набір інструментів для кластеризації, включаючи реалізації алгоритмів k-середніх та k-NN. Scikit-learn надає інтерфейси для використання цих алгоритмів, що значно спрощує процес розробки та тестування моделей.

Scikit-learn також має багато інших корисних інструментів, які можна використовувати для обробки текстових даних, таких як попередня обробка даних, векторизація тексту тощо.

Крім того, scikit-learn має велику спільноту користувачів та документацію, яка дозволяє знайти багато корисних порад та прикладів

використання бібліотеки. Тому scikit-learn є чудовим вибором для кластеризації текстових документів на основі методу k-NN.

Також я знайшов бібліотеку NLTK (Natural Language Toolkit), яка є популярною бібліотекою для обробки природної мови на Python.

NLTK має широкий спектр інструментів, призначених для роботи з текстовими даними. Вона надає можливості для токенізації, лематизації, стемінгу та інших операцій над текстом, що дозволяє підготувати дані перед кластеризацією.

Крім того, NLTK включає в себе набір корпусів та лексичних ресурсів, які можна використовувати для розширення можливостей обробки тексту. Наприклад, можна отримати доступ до словничків, синтаксичних граматик, класифікаторів тощо.

NLTK також має підтримку для векторизації тексту, що є важливою операцією для подальшого використання методу k-NN у кластеризації текстових даних. Вона надає можливість створювати векторні представлення тексту на основі різних моделей, таких як Bag-of-Words, TF-IDF та інших.

Також не варто забувати та про NumPy, адже це потужна бібліотека для наукових обчислень у Python. Вона дозволяє ефективно працювати з числовими даними та масивами. У випадку побудови алгоритму кластеризації, NumPy допоможе виконувати швидкі та оптимізовані обчислення, такі як маніпуляції з матрицями та векторами, що є ключовими елементами цього процесу.

Matplotlib є потужною бібліотекою для візуалізації даних у Python. Вона надає засоби для створення різноманітних графіків та діаграм, що дозволяє наочно представити результати кластеризації. Завдяки Matplotlib можна легко створити графічні відображення

кластерів та їх взаємного розташування, що сприяє кращому розумінню структури даних та виявленню взаємозв'язків.

Використання цих бібліотек дозволяє зосередитися на розробці алгоритму кластеризації, не витрачаючи багато часу на написання оптимізованих математичних операцій чи складних графічних відображень. Завдяки їх широким можливостям та документації, використання NumPy та Matplotlib є зручним та ефективним способом реалізувати алгоритм кластеризації та візуалізувати його результати.

3.3. Висновки до розділу

Підсумовуючи ми бачимо, що використання мови Python є розумним вибором для поставленої задачі, а саме кластеризації текстових документів на основі методу k-найближчих сусідів.

Основні аргументи, які підтверджують цей вибір, включають:

1. Велика кількість ресурсів та документації: Python має широку спільноту програмістів і дослідників, які працюють у сфері машинного навчання та аналізу даних. Це означає, що є безліч матеріалів, документації та ресурсів, що полегшують розробку та розв'язання проблем, пов'язаних з кластеризацією текстових даних на основі методу k-NN.
2. Багато бібліотек машинного навчання та аналізу даних: Python має широкий вибір бібліотек, таких як scikit-learn, numPy, matplotlib, NLTK та інші, які надають реалізації алгоритмів кластеризації, включаючи k-NN. Ці бібліотеки забезпечують готові рішення інтегровані у Python, що спрощує розробку та реалізацію алгоритму кластеризації.
3. Зручний синтаксис: Python має простий і зрозумілий синтаксис, що полегшує розробку програм для обробки текстових даних. Це дозволяє

швидко та легко реалізовувати алгоритми кластеризації на основі методу k-NN.

4. Розширюваність: Python має можливості розширення функціональності за допомогою різних бібліотек та модулів. Це дає можливість розробникам швидко розширювати функціональність своєї програми, додавати нові функції або алгоритми за потреб.

4. РОЗРОБКА АЛГОРИТМУ НА ОБРАНІЙ МОВІ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ БІБЛІОТЕЧНИХ РІШЕНЬ

4.1. Вибір метрики для алгоритму

Я обрав для свого алгоритму я обрав косинусну метрику схожості. Косинусна схожість визначається як кут між двома векторами, які представляють два документи, і часто використовується для знаходження схожих документів у великих текстових колекціях. Косинусна схожість дорівнює косинусу кута між двома векторами, який визначається за допомогою їхніх координат у просторі. Таким чином, документи, які мають більше спільних слів, матимуть більшу косинусну схожість. Запишемо косинусну схожість у вигляді формули (2.3.1.1)

$$\text{cosine simularity} = S_C(A, B) = \cos\theta = \frac{A*B}{\|A\|*\|B\|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} * \sqrt{\sum_{i=1}^{n} {B_i}^2}} \#(2.3.1.1)$$

Де A, B - вектори параметрів двох об’єктів

Причиною вибору цієї метрики стало те, що вона може застосовуватись для текстів що мають великий обсяг, адже вона зосереджується на визначені кута між двома векторами, а не порівнює всі координати двох векторів.

Також дана метрика реалізована у вигляді зручної функції в бібліотеці scikit-learn. Яку ми реалізуємо наступним чином, задавши на вхід векторизовану матрицю з усіх текстів, на вихід отримаємо відсортовані значення косинусної схожості між всіма текстами:

```
  def build_nearest_matrix(self):
    """
    Будує матрицю найближчих сусідів
    :return: матриця найближчих сусідів
    """
    distances = cosine_similarity(self.matrix)
    self.nearNeigh = distances.argsort()[:, 1:self.neighborNum + 1]
    return self.nearNeigh
```

4.2. Обробка текстових даних перед застосуванням алгоритму

Дуже важливим аспектом кластеризації є попередня обробка даних. Одразу виділю декілька найважливіших пунктів, принаймні на мою думку, які вкрай необхідні при кластеризації текстових документів.

Пропоную наступні кроки попередньої обробки для деяких з них я орієнтовно написав код з використанням бібліотеки scikit-learn, що руде реалізовувати конкретні кроки:

- Токенізація: розбиття тексту на окремі слова (токени).
- Видалення стоп-слів: це слова, які не мають суттєвого значення для аналізу тексту, наприклад, частки, сполучники, займенники тощо.

```
def remove_stopwords(text_list):
    stop_words = ['my', 'big', 'is', 'am']
    vectorizer = CountVectorizer(stop_words=stop_words)
    vectorizer.fit_transform(text_list)
    return vectorizer.vocabulary_
```

- Видалення знаків пунктуації: це знаки, які не несуть інформацію про зміст тексту, такі як крапки, коми, знаки оклику та питання.

```
def remove_punctuation(text):
    table = str.maketrans('', '', string.punctuation)
    return [word.translate(table) for word in text.split(" ")]
```

- Лематизація: перетворення слова до його базової форми (леми), щоб зменшити кількість унікальних слів у тексті.

```
def lemmatize(text):
    lemmatizer = WordNetLemmatizer()
    li = text.split(' ')
    return [lemmatizer.lemmatize(word) for word in li]
```

- Векторизація: перетворення текстових даних на числовий вектор, який можна використовувати для застосування алгоритмів машинного навчання. Одним з популярних методів векторизації є TF-IDF.

```
def vectorize(text):
    stop_words = ['my', 'big', 'is', 'am']
    vectorizer = TfidfVectorizer(stop_words=stop_words)
    X = vectorizer.fit_transform(text_list)
    return X.toarray()
```

4.3. Вибір кількості кластерів

Вибір правильної кількості кластерів теж є дуже важливим етапом при кластеризації даних, адже якщо кількість кластерів буде занадто малою, то алгоритм не зможе виділити всі групи текстових файлів, що варто було б виділити. І навпаки, якщо кількість кластерів буде занадто великою, то наша кластеризація не буде нести достатнього сенсового навантаження, оскільки ми будемо виділяти ті файли які ми б не хотіли виділяти взагалі, адже у них буде мінімальна відмінність.

Проблема полягає в тому, що для методу k-NN не існує алгоритмів підбору оптимальної кількості кластерів, позаяк, наприклад для методу k-means існує метод ліктя, для вибору оптимального числа кластерів. Тому поки я не можу запропонувати альтернативи для вибору оптимальної кількості кластерів для мого методу. Але можна буде експериментувати, і обирати кількість кластерів відповідно до контексту вхідних текстових файлів, і методом підбору обирати найкращий випадок, коли кластеризація відбувається оптимальним чином.

4.4. Реалізація алгоритму кластеризації даних

Згадаємо наш алгоритм, що ми маємо реалізувати:

Алгоритм кластеризації:

1. Вибрати параметр k - кількість найближчих сусідів.
2. Вибрати метрику відстані.
3. Перетворити кожен об'єкт у вектор ознак.
4. Створити порожній список кластерів.
5. Для кожного об'єкта дати йому випадковий кластер.
6. Для кожного об'єкта визначити k найближчих сусідів за обраною метрикою відстані.
7. Для кожного об'єкта порівняти його з його k найближчими сусідами та визначити найчастіший кластер серед цих сусідів.
8. Присвоїти кожному об'єкту кластер, який отримали в результаті попереднього кроку.
9. Якщо кластери змінились після останньої ітерації, повторити кроки 6-8.

Кластеризація завершена, повернути отримані кластери.

Поки відкладемо експерименти з к-стю найближчих сусідів і к-стю кластерів, а зосередимось конкретно на алгоритмі.

Повний код програми можна знайти в додатках (Додаток А, Додаток Б, Додаток В).

А зараз ми коротко розглянемо основні функції що реалізують наш алгоритм:

Наступна функція реалізує всі попередні підготовки текстових документів, а саме лематизацію, видалення стоп слів та векторизацію за допомогою алгоритму Tf-Idf:

```
    def additional_text_prep_vectorize(self, text_list):
    """
    Функція виконує повну підготовку текстів до векторизації, а саме
    лематизацію, видалення стоп-слів та векторизує методом TF-IDF
    :param text_list: список текстів у вигляді словника зі вмістом
    під ключем "body"
    :return: Векторизований список текстів
    """
    nltk.download('wordnet')
    lemmatizer = WordNetLemmatizer()
    nltk.download('stopwords')
    stop_words = list(stopwords.words('english'))
    vectorizer = TfidfVectorizer(stop_words=stop_words)
    only_texts = []
    for text in text_list:
        lem_text = [lemmatizer.lemmatize(word) for word in
text['body'].split(" ")]
        final_text = ' '.join(lem_text)
        only_texts.append(final_text)
    # create vocab
    self.vocab = vectorizer.fit(only_texts)
    for text in text_list:
        text['body'] = vectorizer.transform([text['body']])
    return text_list
```

Наступна функція обчислює відстані між всіма текстами, за допомогою косинусної подібності, потім сортує за схожістю, бере кластери n найближчих текстів та присвоює в новий список майбутній кластер для кожного тексту:

```
    def find_k_nearest_neighbors(self):
    """
    Обчислює найближчих сусідів до кожного елемента
    і на основі цього повертає список нових кластерів
    :return: список нових кластерів
    """
    neighbor_indices = self.build_nearest_matrix()
    future_clusters = np.empty(self.textNum, dtype="int")
    for i in range(self.textNum):
        near_clusters = [self.clusterList[ind] for ind in
neighbor_indices[i]]
        mode_cluster = statistics.mode(near_clusters)
        future_clusters[i] = mode_cluster

    return future_clusters
```

І остання функція, яку варто розглянути, це власне цикл нашої кластеризації, де ми реалізуємо пункт 9 нашого алгоритму, описаного вище:

```
    def clusterize(self):
    """
    Цикл кластеризації
    :return: None
    """
    i = 0
    while i < self.iterNum:
        i += 1
        future_clust = self.find_k_nearest_neighbors()
        if np.array_equal(future_clust, self.clusterList):
            self.visualise(self.matrix)
            return
```

```
        self.clusterList = future_clust
        self.visualise(self.matrix)
```

4.5. Висновки до розділу

В ході розробки нашого алгоритму на мові Python використовуючи різні бібліотеки, в особливості statistics, numpy, matplotlib, scikit-learn, nltk, що дуже допомогли, представивши величезний набір функцій, за допомогою яких дуже легко було векторизувати тексти, обробити лексичну складову цих текстів, знайти відстані між текстами та інші операції з масивами та матрицями, ну і також із представленням результатів роботи.

В ході я помітив значні затрати часу на попередню обробку текстів, що займає майже стільки часу як і сам алгоритм. Але без попередньої обробки ми не зможемо досягти такої точності кластеризації.

5. ПІДБІР ТЕКСТОВИХ ДОКУМЕНТІВ ДЛЯ ТЕСТУВАННЯ ПРОГРАМИ

5.1. Етапи підготовки даних

Я б розбив підбір та підготовку даних, на ще декілька більш дрібних етапів. Адже підбір тестових даних для тестування алгоритму кластеризації на основі методу k-NN важливий етап, щоб переконатися в його ефективності та відповідності поставленим завданням. Основний принцип полягає в тому, щоб мати тестові дані, які максимально відображають реальний сценарій використання алгоритму.

Кроки які варто виконати при виборі тестових даних для тестування алгоритму:

- Визначити цільову аудиторію: вибрати групу людей або сферу, в якій планується застосовувати алгоритм кластеризації. Наприклад, це може бути кластеризація новинних статей, соціальних медіа дописів.
- Препроцесінг тексту: виконати Препроцесінг тексту для підготовки даних перед застосуванням алгоритму кластеризації. Це включає очищення тексту від непотрібних символів, токенізацію, лематизацію або стемінг, видалення стоп-слів і т.д.
- Векторизація тексту: перетворення текстових дані в числові вектори, які можна використовувати для кластеризації. Один із популярних підходів - використання методу TfIdf (Term frequency-inverse document frequency). Цей метод враховує частоту та важливість слів у текстах.
- Вибір кількості кластерів: Розгляньте цільову кількість кластерів, яку ви хочете отримати з алгоритму k-NN. Вибір кількості кластерів може бути суб'єктивним або може базуватися на певних вимогах

додатка. Наприклад, якщо планується кластеризувати новинні статті, можливо, буде логічним використовувати кількість кластерів, відповідну категоріям новин.

- Візуалізація результатів: використовувати візуалізацію, таку як діаграми розсіювання або дендрограми, для відображення кластерів та їх структури. Це допоможе краще розуміти результати кластеризації та зробити відповідні висновки.

У цілому, вибір тестових даних для кластеризації на основі методу k-NN залежить від конкретної задачі та цілей додатка. Важливо мати репрезентативні дані, що відображають реальну ситуацію, та використовувати адекватні метрики для оцінки результатів. Тестування алгоритму з різними наборами даних та параметрами допоможе підібрати оптимальні налаштування для конкретної задачі.

5.2. Визначення цільової аудиторії

Оскільки зараз нашою метою є підтвердження працездатності додатка, то нам підійдуть будь-які текстові дані, які можливо кластеризувати. Почавши свої пошуки джерел, де можна взяти текстові дані для перевірки алгоритму кластеризації, я знайшов наступний перелік джерел про які коротко тут розповім.

UCI Machine Learning Repository: Це великий репозиторій з відкритим доступом, який містить різні набори даних для машинного навчання. Ви можете знайти там набори даних, що підходять для кластеризації, наприклад, "Iris" або "Wine".

Kaggle: Kaggle - це платформа для спільного співробітництва та змагань з машинного навчання. Ви можете знайти велику кількість наборів даних, які були використані у різних конкурсах. Деякі з цих наборів даних можуть містити текстові дані, які підходять для

кластеризації.

OpenML: OpenML - це онлайн-платформа для відкритого доступу до наборів даних та алгоритмів машинного навчання. Ви можете знайти різноманітні набори даних з різних областей, які можна використовувати для кластеризації.

Text Corpora: Існують деякі текстові корпуси, які можна використовувати для кластеризації тексту. Наприклад, "20 Newsgroups" - набір новинних статей, або "Reuters-21578" - набір новин про фінанси та бізнес. Ці дані містять категоризовані тексти, які можна використовувати для тестування алгоритмів кластеризації тексту.

Втім мій вибір впав саме на набір статей, Reuters-21578. Я не опирався на те що це саме новини, просто в інших джерелах, дані були представлені змішаного типу, тобто текстові з числовими, що нам абсолютно не підходить, або взагалі завантажити потрібний набір даних було вкрай складно. Тому надалі ми будемо працювати з колекцією Reuters-21578.

5.3. Колекція Reuters-21578

Завантаживши дані Reuters-21578 я отримав архів, у якому було представлено безліч різних файлів, представлених далі на Рис 5.3.1

all-exchanges-str...	186	?	Текстовий докуме...	04.12.1996 20:27
all-orgs-strings.l...	316	?	Текстовий докуме...	04.12.1996 20:27
all-people-string...	2 474	?	Текстовий докуме...	04.12.1996 20:27
all-places-strings...	1 721	?	Текстовий докуме...	04.12.1996 20:27
all-topics-strings...	1 005	?	Текстовий докуме...	04.12.1996 20:27
cat-descriptions_...	28 194	?	Текстовий докуме...	04.12.1996 20:27
feldman-cia-wor...	273 802	?	Текстовий докуме...	10.12.1996 5:32
lewis.dtd	1 485	?	XML Document Typ...	23.01.1997 16:14
README.txt	36 388	?	Текстовий докуме...	26.09.1997 17:15
reut2-000.sgm	1 324 350	?	Файл SGM	04.12.1996 20:26
reut2-001.sgm	1 254 440	?	Файл SGM	04.12.1996 20:26
reut2-002.sgm	1 217 495	?	Файл SGM	04.12.1996 20:26
reut2-003.sgm	1 298 721	?	Файл SGM	04.12.1996 20:26
reut2-004.sgm	1 321 623	?	Файл SGM	04.12.1996 20:26
reut2-005.sgm	1 388 644	?	Файл SGM	04.12.1996 20:26
reut2-006.sgm	1 254 765	?	Файл SGM	04.12.1996 20:26
reut2-007.sgm	1 256 772	?	Файл SGM	04.12.1996 20:26
reut2-008.sgm	1 410 117	?	Файл SGM	04.12.1996 20:26
reut2-009.sgm	1 338 903	?	Файл SGM	04.12.1996 20:26
reut2-010.sgm	1 371 071	?	Файл SGM	04.12.1996 20:26
reut2-011.sgm	1 304 117	?	Файл SGM	04.12.1996 20:26
reut2-012.sgm	1 323 584	?	Файл SGM	04.12.1996 20:26
reut2-013.sgm	1 129 687	?	Файл SGM	04.12.1996 20:26
reut2-014.sgm	1 128 671	?	Файл SGM	04.12.1996 20:26
reut2-015.sgm	1 258 665	?	Файл SGM	04.12.1996 20:26
reut2-016.sgm	1 316 417	?	Файл SGM	04.12.1996 20:26
reut2-017.sgm	1 546 911	?	Файл SGM	04.12.1996 20:26
reut2-018.sgm	1 258 819	?	Файл SGM	04.12.1996 20:26
reut2-019.sgm	1 261 780	?	Файл SGM	04.12.1996 20:26
reut2-020.sgm	1 049 566	?	Файл SGM	04.12.1996 20:26
reut2-021.sgm	621 648	?	Файл SGM	04.12.1996 20:26

Рис 5.3.1 - Файли колекції Reuters-21578

Втім для мене цікавими є лише файли з розширенням .SGM. Вигляд вмісту кожного з файлів зображений на скріншоті (Рис 5.3.2). Файли Reuters-21578, які зберігаються у форматі .sgm (Standard Generalized Markup Language), є набором новинних статей, що використовуються для дослідження у сфері обробки природної мови та кластеризації тексту. Основна структура цих файлів виглядає так:

- Кожен файл .sgm представляє собою XML-документ.
- Кореневим елементом є <REUTERS>, який містить одну або більше новинну статтю.
- Кожна новинна стаття має свій унікальний ідентифікатор, який можна знайти в атрибуті "NEWID" елемента <REUTERS>.
- Вміст новинної статті знаходиться між тегами <TEXT>...</TEXT>. Він може містити заголовок, текст статті, метадані та іншу інформацію.

- Деякі статті можуть мати тег <TOPICS>, який містить категорії або теми, пов'язані з цією статтею.
- Деякі статті можуть мати тег <PLACES>, який містить інформацію про місцезнаходження, пов'язане з цією статтею.

Деякі статті можуть мати тег <D> або <DATE>, який містить дату публікації цієї статті.

У загальному, структура файлів Reuters-21578 дозволяє зберігати та організовувати велику кількість новинних статей з різними атрибутами та метаданими. Для роботи з цими файлами потрібно використовувати парсер XML, який дозволяє зчитувати та отримувати доступ до різних елементів та атрибутів документа.

```
<!DOCTYPE lewis SYSTEM "lewis.dtd">
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="5544" NEWID="1">
<DATE>26-FEB-1987 15:01:01.79</DATE>
<TOPICS><D>cocoa</D></TOPICS>
<PLACES><D>el-salvador</D><D>usa</D><D>uruguay</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;C T
&#22;&#22;&#1;f0704&#31;reute
u f BC-BAHIA-COCOA-REVIEW   02-26 0105</UNKNOWN>
<TEXT>&#2;
<TITLE>BAHIA COCOA REVIEW</TITLE>
<DATELINE>    SALVADOR, Feb 26 - </DATELINE><BODY>Showers continued throughout the week in
the Bahia cocoa zone, alleviating the drought since early
January and improving prospects for the coming temporao,
although normal humidity levels have not been restored,
Comissaria Smith said in its weekly review.
    The dry period means the temporao will be late this year.
    Arrivals for the week ended February 22 were 155,221 bags
```

```
    Total Bahia sales are currently estimated at 6.13 mln bags
against the 1986/87 crop and 1.06 mln bags against the 1987/88
crop.
    Final figures for the period to February 28 are expected to
be published by the Brazilian Cocoa Trade Commission after
carnival which ends midday on February 27.
 Reuter
&#3;</BODY></TEXT>
</REUTERS>
```

Рис 5.3.2 - Вміст .sgm файлів з колекції Reuters - 21578

5.4. Препроцесінг даних

Тепер перейдемо до розгляду функцій, які ми будемо використовувати для того, щоб дістати статті з .SGM файлів. Для цього я написав невеличкий клас SGMReader, що містить декілька функцій що зчитують файли та повертають як список словників з такими ключами як назва, дата, вміст. Коротко подивимось функцію, що власне і зчитує кожен з файлів.

Варто зазначити що у кожному файлі знаходиться близько 1000 статей. Тому я шукав всі вмісти, що знаходяться між тегами TEXT. Код функції:

```
    def parse_sgm_file(self, filename):
  with open(filename, 'r') as f:
      soup = BeautifulSoup(f, 'html.parser')
      texts = []
      for text in soup.find_all('text'):
          title = text.find('title').text if text.find('title') else None
          dateline = text.find('dateline').text if text.find('dateline')
else None
          b = text.find('body').text if text.find('body') else None
          body = re.sub('[^0-9a-zA-Z]+', ' ', str(b)).lower()
```

```
            textex = {'title': title, 'dateline': dateline, 'body': body}
            texts.append(textex)
        return texts
```

Як бачимо, я використав дуже зручні інструменти з бібліотеки BeatifulSoup. За допомогою якого ми легко можемо парсити навіть такі складні, на перший погляд, документи. Також я зробив функцію яка приймає цілу директорію файлів з таким розширенням і повертає один список:

```
    def parse_sgm_directory(self, directory_path):
texts = []
for filename in os.listdir(directory_path):
    if filename.endswith(".sgm"):
        file_path = os.path.join(directory_path, filename)
        text = self.parse_sgm_file(file_path)
        texts += text
return texts
```

В цілому ми легко можемо надавати на вхід алгоритму дані, які правильно розпарсені та структуровані для подальшої обробки.

5.5. Інші етапи обробки даних

Далі мають йти етапи: векторизація тексту, вибір кількості кластерів, вибір метрик оцінки результату, візуалізація даних.

Також для більш зручної роботи з функціями scikit-learn є необхідність перетворити дані у вигляді списку словників у три структури даних, два звичайних масиви. У одному з яких будуть зберігатись назви статей, що надалі можна буде використовувати як ID цих статей. У другому масиві зберігатимуться номери кластерів для

кожного текстів. Ну і третя структура, це буде двовимірний масив, де кожен рядок масиву ще набір координат для окремого тексту, а кожен стовпчик, це параметри TF-IDF для кожного терму зі словника.

Функція що перетворює одну структуру на іншу представлена далі, втім створює лише матрицю:

```
    def one_matrix_init(self, text_list):
n_col, n_row = text_list[0]['body'].toarray().shape
self.matrix = np.empty((len(text_list), n_row))
for i in range(len(text_list)):
    self.matrix[i] = text_list[i]['body'].toarray()
return self.matrix
```

Також я створив функцію, що одночасно змінює одну структуру на іншу, залишивши назви статей, і паралельно обираючи для кожної статті випадковий кластер:

```
    def one_matrix_cluster_init(self, text_list):
n_col, n_row = text_list[0]['body'].toarray().shape
self.textNum = len(text_list)
self.matrix = np.empty((len(text_list), n_row))
self.clusterList = np.empty(len(text_list), dtype="int")
self.nameList = np.empty(len(text_list), dtype="<U16")
for i in range(len(text_list)):
    random_cluster = i % self.clusterNum
    self.nameList[i] = text_list[i]['title']
    self.clusterList[i] = random_cluster
    self.matrix[i] = text_list[i]['body'].toarray()
return self.matrix
```

5.6. Висновки до розділу

В даному розділі було підібрано колекцію текстових документів, у нашому випадку це колекція Reuters-21578, що представляє собою

набір статей економічного характеру з 90-х років. Її було обрано, через те що це суто текстові дані без домішок інших типів даних, що є зручним набором, для того, щоб протестувати дієздатність алгоритму кластеризації саме текстових даних.

Втім варто одразу зазначити мінуси такого вибору, оскільки всі ці дані це новини одного напрямку, то виділити відмінності між ними буде набагато складніше, ніж між текстовими документами, що репрезентують різні напрямки.

Також ми розглянули як саме будо реалізовано парсер SGM файлів, що допомагає представити ці файли у зручному форматі для подальшого використання алгоритмом. І в цілому згадали всі етапи попередньої підготовки файлів перед використанням алгоритму.

6. ПІДТВЕРДЖЕННЯ ПРАЦЕЗДАТНОСТІ ПРОГРАМИ

6.1. Робота алгоритму

Розглянемо функцію main(), щоб потім розібрати, як буде працювати наш алгоритм:

```
    if __name__ == '__main__':
    # Створення об'єкта класу кластеризатора
    k = kNNclusterizer(neighborNum=2, clusterNum=2, iterNum=4)
    # Зчитування файлів
    textlist = k.readSGM(list=True, path=["testFiles/reut2-001.sgm",
"testFiles/reut2-000.sgm"])
    # Векторизація текстів
    textVectorised = k.additional_text_prep_vectorize(textlist)
    # Початкова ініціалізація кластерів
    matrix = k.one_matrix_cluster_init(textVectorised)
    # Візуалізація початкової ініціалізації
    vis = k.visualise(matrix)
    # Кластеризація
    k.clusterize()
    # Повернення кластеризованих текстів у вигляді словника
    result = k.returnClusters()
```

Для початку ми запустимо програму на тестових даних, які ми підібрали на минулому тижні переддипломної практики, а саме використаємо лише перші дві збірки статей, тобто в сумі 2000 статей.

Тобто для початку ми створюємо екземпляр класу кластеризатора, який буде кластеризувати опираючись на 5 найближчих сусідів. Кількість кластерів, якими будуть ініціалізуватись тексти на початку, за замовчуванням буде дорівнювати 4. Далі ми проводимо набір дій з підготовки тексту, таких як лематизація, прибирання стоп-слів і так далі, все це було описано у звіті за 3

тиждень практики. Ну і звісно відбувається векторизація. Потім ми трансформуємо структуру даних, де кожен текст розбивається у словник, на три різні масиви, перший масив імен, другий, матриця, де кожен рядок, це вектор одного тексту, і третє це масив кластерів. Де звернувшись по одному індексу, до трьох структур, ми отримаємо параметри одного тексту.

6.2. Візуалізація результатів

Візуалізація даних при кластеризації є важливим етапом, оскільки дозволяє зрозуміти структуру даних, виявити можливі закономірності та зв'язки між об'єктами. Вона допомагає виявити кластери, їх розподіл та взаємне розташування. При візуалізації текстових даних, які є векторами ознак, можна використовувати методи зменшення розмірності даних. Основна мета зменшення розмірності - збереження важливої інформації та спрощення подання даних без втрати суттєвих характеристик.

Деякі з популярних методів зменшення розмірності даних для візуалізації текстових даних включають:

Метод головних компонент (Principal Component Analysis, PCA): Використовується для зменшення розмірності шляхом проектування даних на нові ортогональні компоненти, які максимізують дисперсію даних.

t-SNE (t-Distributed Stochastic Neighbor Embedding): Цей метод зберігає відносності між точками, спробуючи зберегти подібність у високих та низьких розмірностях. Він ефективний для візуалізації складних зв'язків між об'єктами.

UMAP (Uniform Manifold Approximation and Projection): Цей метод також зберігає локальну та глобальну структуру даних. UMAP

зазвичай швидший за t-SNE, але все ж забезпечує хорошу візуалізацію даних.

Важливо відзначити, що вибір методу залежить від конкретної задачі та типу даних. Тому я вирішив спробувати реалізувати кілька методів та порівняти їх результати для кращого розуміння даних та кластерів. Візуалізація даних при кластеризації допомагає глибше зрозуміти структуру даних, з'ясувати, чи вдалося правильно виділити кластери, та спростити подання результатів аналізу.

Реалізувавши методи зменшення розмірності PCA та t-SNE я прийшов до висновку, що краще використовувати методі t-SNE, адже він краще відображає дані.

6.3. Результати роботи алгоритму

Запустивши алгоритм, ми перед початком кластеризації виводимо наші дані на графіку, де кожна точка відповідає за окремий текст, варто одразу відзначити ми використовуємо метод t-SNE для зменшення розмірності векторів. Дані з 2000 статей до кластеризації зображені на (Рис 6.3.1). Нагадаю що початково всім текстам присвоюється один з чотирьох кластерів випадковим чином.

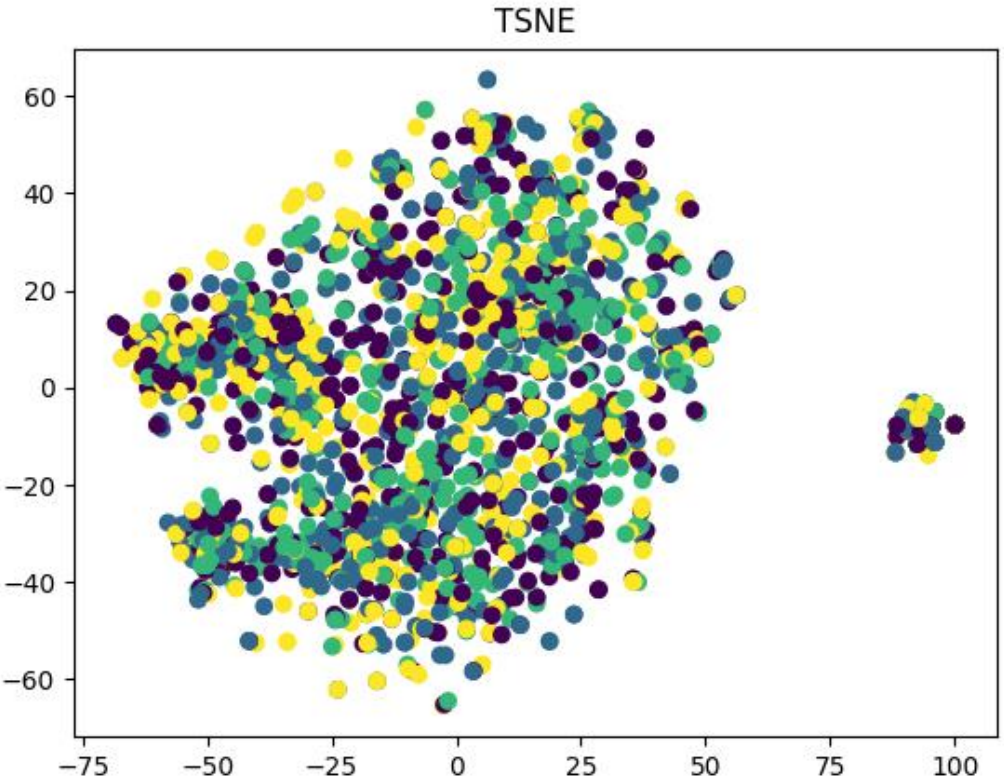

Рис 6.3.1 - Початкова ініціалізація кластерів

Перейдемо до другої ітерації нашого алгоритму, тобто присвоєння кожному тексту кластера, за методом найближчих сусідів, де кількість найближчих сусідів = 5. Знайшовши для кожного тексту свій кластер і присвоївши їх ми отримаємо наступний графік (Рис 6.3.2), де ми вже бачимо що алгоритм чітко виділив групу справа до одного кластеру.

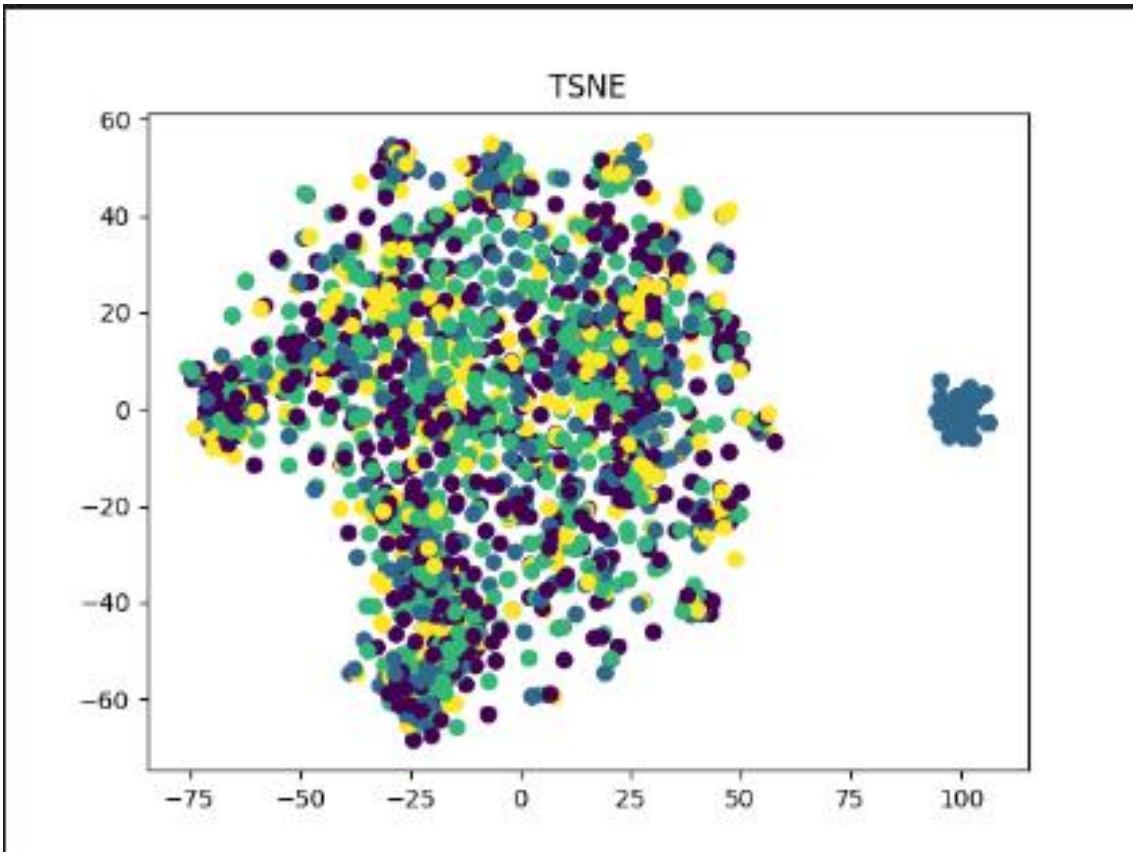

Рис 6.3.2 - Після першої ітерації алгоритму

Продовживши алгоритм, ми побачимо різке покращення кластеризації (Рис 6.3.3). Надалі ми продовживши алгоритм, ми не отримаємо приросту, а у нас просто трохи будуть зміщатись аномальні точки, які не хоче правильно кластеризувати алгоритм, хоча можливо тут проблема саме у візуалізації результатів.

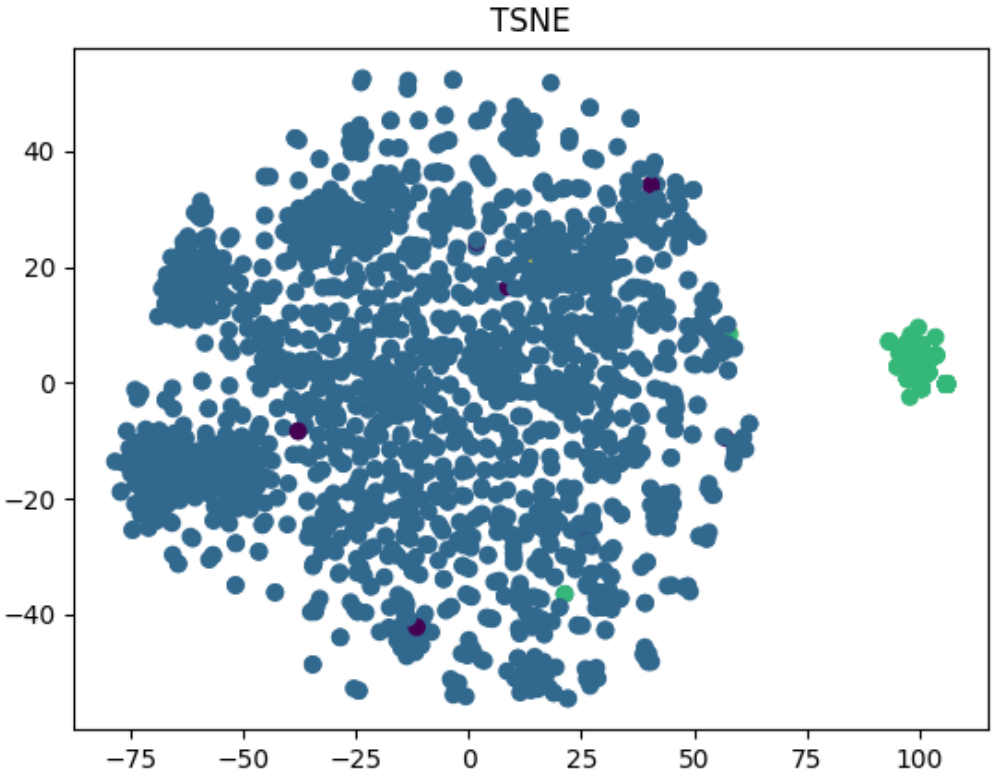

Рис 6.3.3 - Результат після 2-ї ітерації алгоритму

Але в цілому, як бачимо, наш алгоритм цілком дієздатний.

6.4. Висновки до розділу

В цілому алгоритм має право на життя. Так співпало, що я не зміг перевірити на практиці, чи зберігається головна перевага методу k-найближчих сусідів для класифікації даних, у задачах з кластеризацією, а саме те що він гарно спрацьовує на даних і кластерах зі складною формою. Втім цей алгоритм досить простий в інтерпретації та реалізації.

Але якщо порівнювати з алгоритмом k-means, наприклад, то головним мінусом буде його висока обчислювальна складність особливо на великих наборах даних. А також як і у k-means він

чутливий до шуму і вкидів даних.

Також варто зазначити, що ми отримуємо перевагу, що кластеризація не буде залежати від вибору початкових центроїдів, як в алгоритмі k-means, оскільки ми ініціалізуємо дані випадковим чином, і завжди отримуємо майже однакову картину.

7. ВИМІРИ ПРОДУКТИВНОСТІ АЛГОРИТМУ

7.1. Час на попередню обробку даних

Одразу варто зазначити що виміри проводились на обладнанні з відносно низькою потужністю, тому в залежності від обчислювальної машини результати можуть відрізнятись.

Втім в даному розділі нас цікавить наскільки заповільниться робота алгоритму від збільшення кількості даних, адже як ми зазначали раніше метод k-найближчих сусідів не найкращим чином себе показує на великих об’ємах даних.

Тестування буде проводитись на все тих же даних, а саме на колекції Reuters. Під час тестування ми будемо окремо тестувати на даних з 1000 статей, 2000 статей, 5000 статей та 10000 статей.

Перейдемо до перевірки скільки часу займає попередня підготовка даних, а саме перетворення .sgm файлу на набір даних у зручному для нас вигляді, лематизація слів, прибирання стоп слів.

- 1000 статей: середній час на попередню обробку - 3.2 секунди
- 2000 статей: середній час на попередню обробку - 4.88 секунди
- 5000 статей: середній час на попередню обробку - 11.43 секунди
- 10000 статей: середній час на попередню обробку - 23.04 секунди

Подання у вигляді таблиці (Таблиця 2) та графіку (Рис. 7.1.1)

Таблиця 2 - залежність часу попередньої обробки даних від к-сті даних

	1000	2000	5000	10000
Попередня обробка	3,2	4,8	11,	23,

даних, с		8	43	04

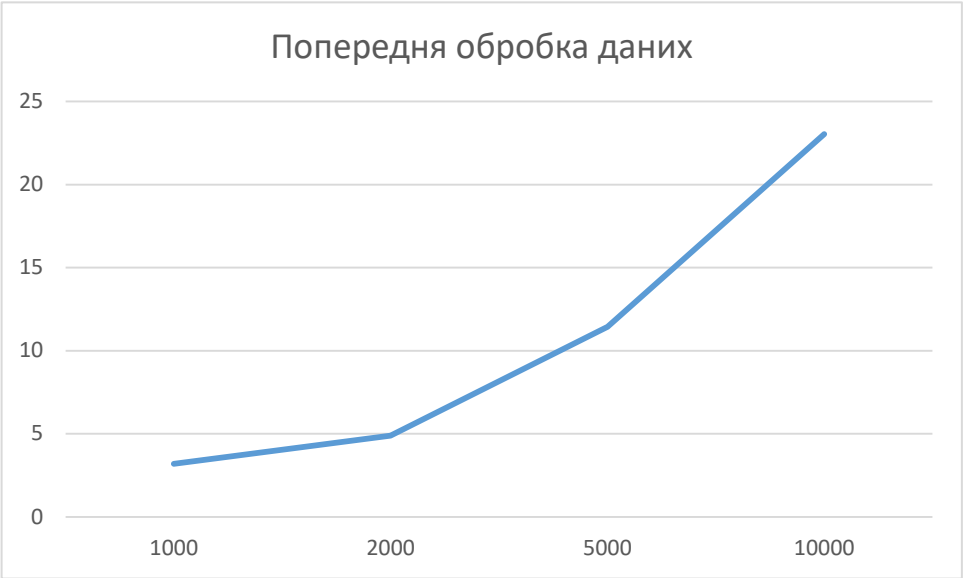

Рис. 7.1.1. - Графічне представлення залежності часу попередньої обробки даних від к-сті даних

7.2. Час на кластеризацію

Дані та їх кількість залишиться такою самою як і у попередньому підрозділі. Варто зробити примітку, що для вимірів ми відімкнули візуалізацію на кожній ітерації, оскільки це досить ресурсозатратний процес Тому можемо одразу перейти до замірів часу:

- 1000 статей: середній час кластеризації - 0.23 секунди
- 2000 статей: середній час кластеризації - 1.26 секунди
- 5000 статей: середній час кластеризації - 8.58 секунди
- 10000 статей: середній час кластеризації - 44.16 секунди

Подання у вигляді таблиці (Таблиця 3) та графіку (Рис. 7.2.1)

Таблиця 3 - залежність часу кластеризації даних від к-сті даних

	1000	2000	5000	10000
Кластеризація, с	0,23	1,26	8,58	44,16

Рис. 7.2.1. - Графічне представлення залежності часу кластеризації даних від к-сті даних

7.3. Результуючий час роботи алгоритму

Тепер підсумуємо дані з попередніх підпунктів, щоб визначити результуючий час виконання алгоритму і подамо це знову ж у вигляді таблиці (Таблиця 4) та графіку (Рис. 7.3.1.)

Таблиця 4 - Час роботи алгоритму на різній к-сті даних

	1000	2000	5000	10000
Попередня обробка даних, с	3,2	4,88	11,43	23,04

Кластеризація, с	0,23	1,26	8,58	44,16
Підсумковий час, с	3,43	6,14	20,01	67,2

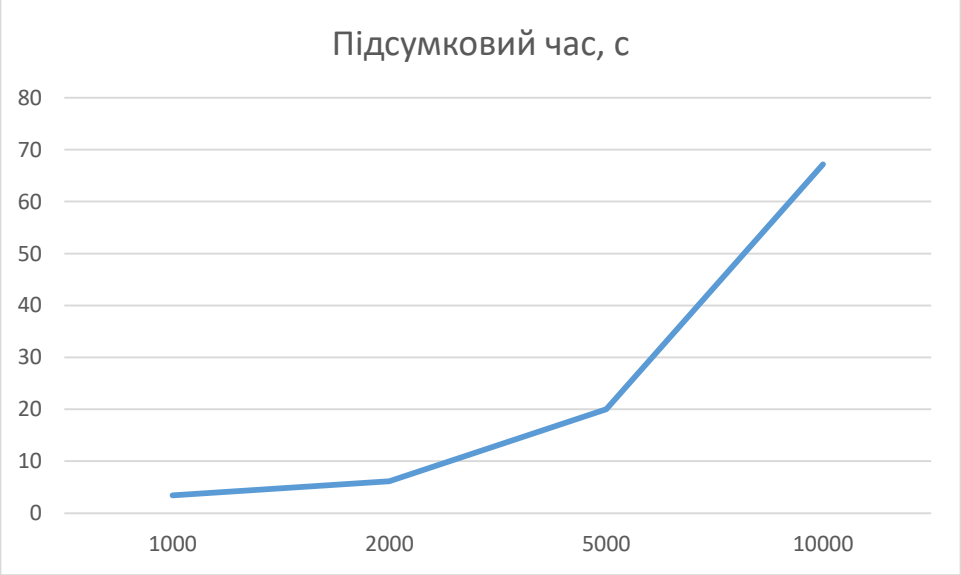

Рис. 7.3.1. - Графічне представлення залежності часу роботи алгоритму від к-сті даних

7.4. Висновки до розділу

Провівши замірювання часу роботи алгоритму у даному розділі, ми отримали цілком передбачувані результати.

Час попередньої підготовки даних збільшується в залежності від кількості даних майже лінійно, а от процес кластеризації в залежності від кількості даних більше нагадує експоненту. Власне це досить очікувано як для алгоритму побудованого на основі методу k-найближчих сусідів.

Можемо зробити висновок, що цей алгоритм краще застосовувати для невеликих об’ємів даних. В випадку нашого набору даних, це не більше 2000 статей.

8. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проведена оцінка основних функціональних та вартісних характеристик програмного продукту, а саме додатку для кластеризації текстових даних на основі алгоритм, в основу якого покладений метод k-найближчих сусідів.

Програмний продукт призначено для використання у навчальних цілях, та подальших досліджень алгоритмів кластеризації на основі розробленого продукту.

Нижче наведено аналіз різних варіантів реалізації модуля з метою вибору оптимальної стратегії створення програмного продукту, враховуючи при цьому як економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи та на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз – це технологія оцінки реальної вартості продукту або послуги незалежно від організаційної структури компанії. Прямі та побічні витрати розподіляються по продуктам та послугам у залежності від потрібних обсягів ресурсів на кожному етапі виробництва. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні найбільш оптимального розподілу ресурсів, що виділені на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Метод ФВА починається з визначення послідовності функцій, необхідних для виробництва продукту. Спочатку йдуть всі можливі функції, які розподіляються по двом групам: ті, що впливають на вартість продукту і

ті, що не впливають. Також на цьому етапі оптимізується сама послідовність скороченням кроків, що не впливають на витрати.

Для кожної функції визначається повний обсяг річних витрат та кількість робочих часів. На основі даних оцінок визначається кількісна характеристика джерел витрат. Після визначення джерел витрат проводиться кінцевий розрахунок витрат на виробництво продукту.

8.1 Постановка задачі техніко-економічного аналізу

Метод ФВА був застосований для проведення техніко-економічний аналізу розробки додатку для кластеризації текстових даних на основі алгоритм, в основу якого покладений метод k-найближчих сусідів. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для обробки та кластеризації текстових даних.

До продукту були визначені наступні технічні вимоги:

1) можливість виконання на персональних комп’ютерах із стандартною конфігурацією;
2) Коректна кластеризація текстових документів;
3) Читабельний код, який в подальшому можна буде легко редагувати;
4) Збереження зрозумілості коду що реалізує кластеризацію на основі методу к-найближчих сусідів, задля збереження основного принципу, даного методу, а саме інтуїтивності.

8.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який вирішує задачу кластеризації текстових даних на основі алгоритм, в основу якого покладений метод k-найближчих сусідів. Виходячи з конкретної мети, можна виділити наступні основні функції програмного продукту:

F_1 – вибір мови програмування;

F_2 – вибір бібліотек для машинного навчання;

F_3 – вибір середовища розробки.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

1) Python
2) R
3) Java
4) C++

Функція F_2:

1) numPy
2) Scikit-learn/numPy
3) Cluster/stats
4) Weka
5) MLPACK

Функція F_3:

1) PyCharm
2) Visual Studio Code
3) RCode

8.2.1 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи на рисунку 8.1. Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

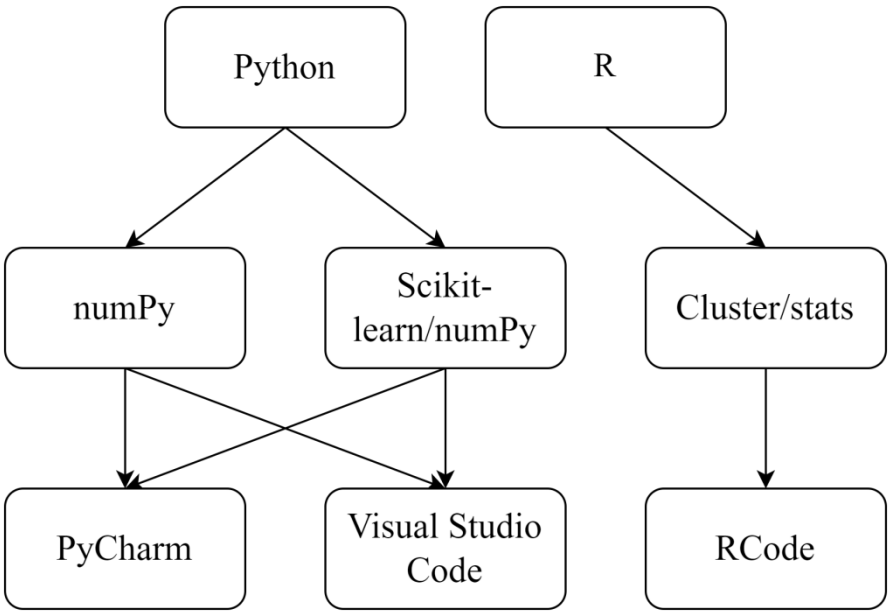

Рисунок 8.2.1 – Морфологічна карта варіантів реалізації функцій

На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (Таблиця 8.2.1).

Таблиця 8.2.1 – Позитивно-негативна матриця варіантів основних функцій

Функція	Варіант реалізації	Переваги	Недоліки
F_1	А	Легкий синтаксис, Швидка розробка, багато наукових бібліотек, для швидкої розробки	Повільний
	Б	Швидша ніж Python	Довший час розробки, менша розповсюдженість, відповідно менший набір інструментів

F_2	А	Написана на C++, дуже швидка, через те що менше різноманітних бібліотек потрібно поєднувати	Необхідність додаткової роботи, більшість функцій для кластеризації доведеться писати вручну, значно довший час розробки
	Б	Потребує значно менше коду оскільки існує багато готових рішень, швидка розробка	Дещо повільніший код в результаті
F_3	А	Зручний інтерфейс, існують інструменти для роботи з науковими бібліотеками.	Більша навантаженість інтерфейсу
	Б	Зосередженість на коді	Відсутність багатьох інструментів, менш зручний процес відлагодження програми
	В	Зручне IDE для R	-

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Оскільки реалізація програмного коду має бути максимально інтуїтивною задля подальших досліджень, варто обрати варіант А, тому що код на даній мові програмування буде більш зрозуміли для основної маси людей, і в цілому буде мати простішу структуру.

Функція F_2:

Оскільки, програмний продукт реалізується мовою Python, як було визначено вище, то будемо обирати з варіантів А та Б. Повертаючись до мети створити максимально зрозумілий додаток в плані коду, ми оберемо варіант Б

Функція F_3:

Вибір середовища розробки також є важливим, оскільки у варіанті А, для нас існує безліч вбудованих інструментів, будемо використовувати саме це рішення.

Таким чином, будемо розглядати такі варіанти реалізації програмного продукту:

– F_1(А) – F_2(Б) – F_3(А)

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

8.3 Обґрунтування системи параметрів програмного продукту

8.3.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб схарактеризувати програмний продукт, будемо використовувати наступні параметри:

– X1 – швидкодія операцій мови програмування в залежності від обраної серверної технології;
– X2 – об’єм пам’яті для збереження даних персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми;
– X3 – час, який витрачається на виконання коду;
– X4 – потенційний об’єм програмного коду, який необхідно створити безпосередньо розробнику.

8.3.2 Кількісна оцінка параметрів

Гірші, середні та кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту як показано у Таблиці 8.3.2.

Таблиця 8.3.2 – Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	1000	10000	50000
Об’єм пам’яті	X2	Мб	2000	100	10
Час попередньої обробки даних	X3	с	20	5	1
Потенційний об’єм програмного коду	X4	кількість рядків коду	1000	600	300

За даними таблиця 8.3.2 будуються графічні характеристики параметрів (див. рис. 8.3.2–8.3.5).

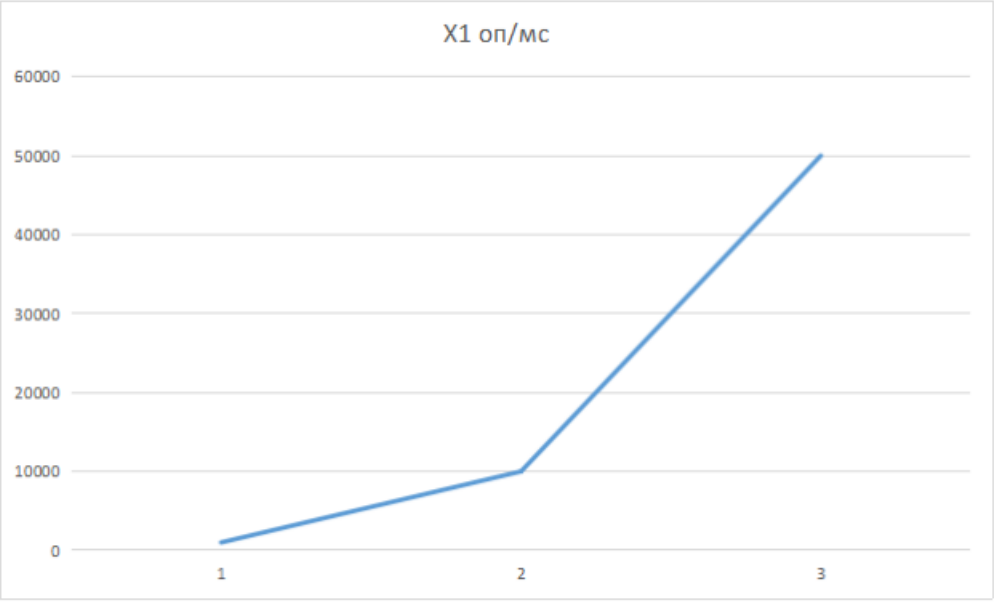

Рисунок 8.3.2 – Швидкодія мови програмування

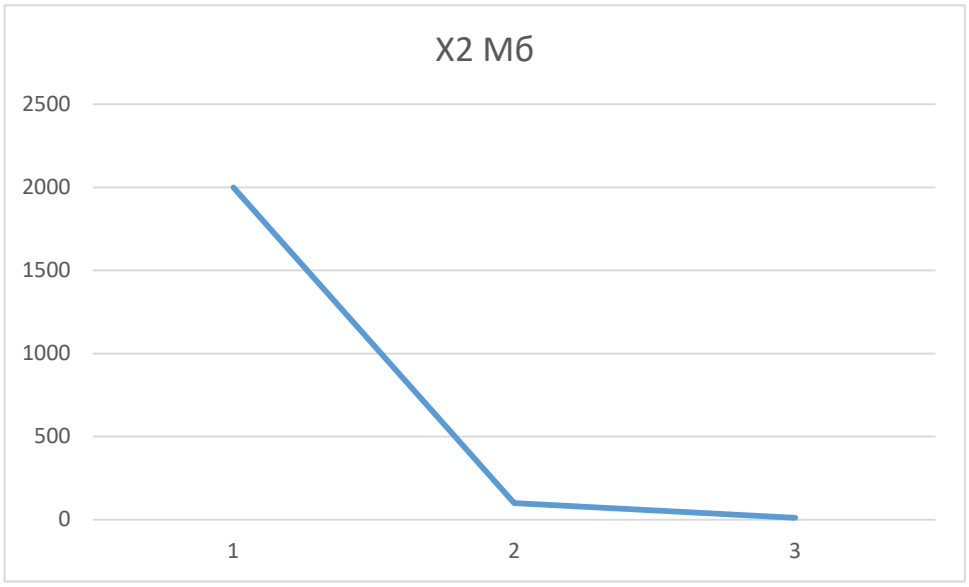

Рисунок 8.3.3 – Об’єм пам’яті для збереження даних

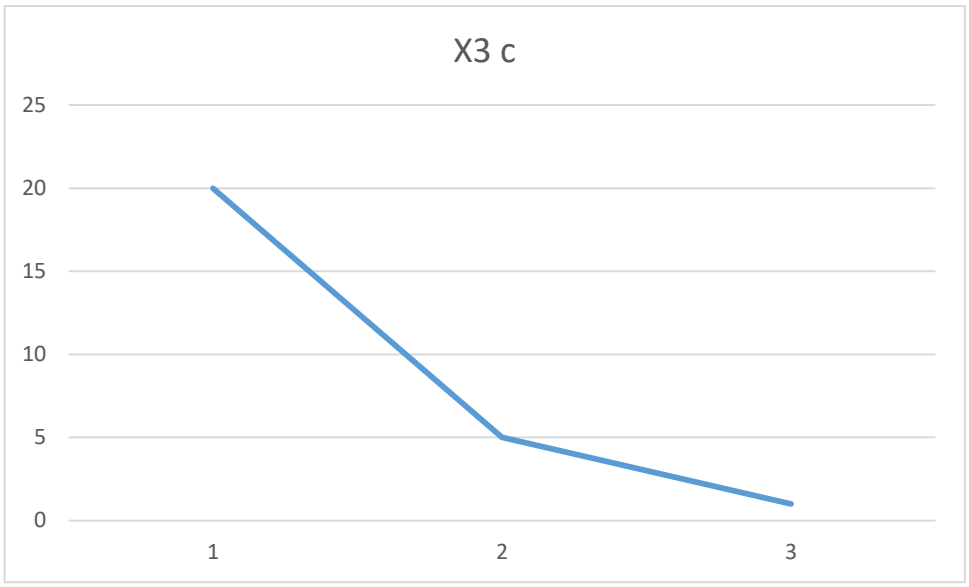

Рисунок 8.3.4 – Час виконання коду

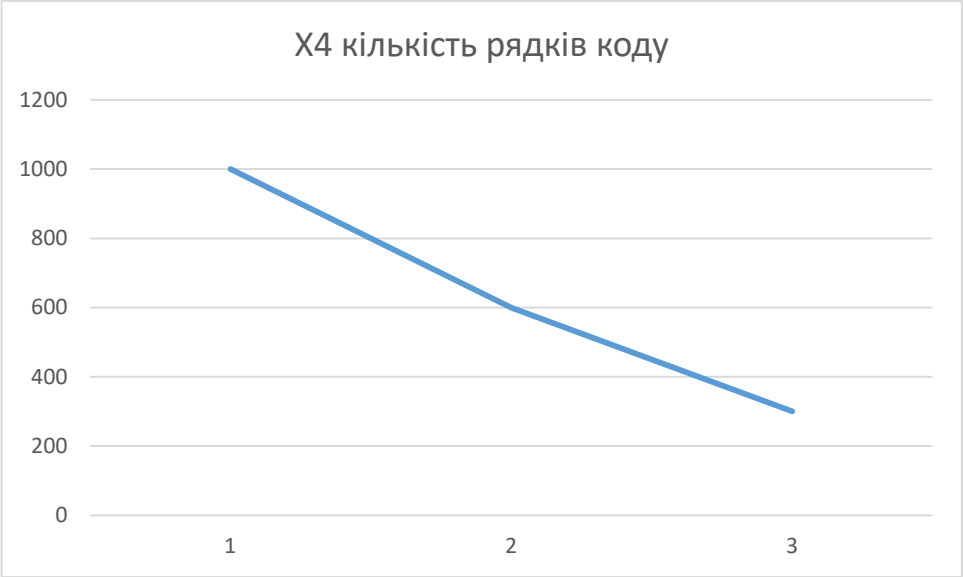

Рисунок 8.3.5 – Потенційний об’єм програмного коду

8.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який має найбільш зручний інтерфейс та зрозумілу взаємодію з користувачем.

Значущість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 5 людей. Визначення коефіцієнтів значущість передбачає:

1) визначення рівня значущості параметра шляхом присвоєння різних рангів;
2) перевірку придатності експертних оцінок для подальшого використання;
3) визначення оцінки попарного пріоритету параметрів;
4) обробку результатів та визначення коефіцієнта значущості.

Результати експертного ранжування наведені у таблиці 8.4.1.

Таблиця 8.4.1 – Результати ранжування показників

Позначення параметра	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
	1	2	3	4	5	6	7			
X1	2	3	2	4	1	1	3	12	-5,5	30,25
X2	1	1	3	1	2	2	2	12	-5,5	30,25
X3	3	2	1	2	4	3	1	18	0,5	0,25
X4	4	4	4	3	3	4	4	28	10,5	110,25
	10	10	10	10	10	10	10	70	0	171

Для перевірки ступеня достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \#(8.4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів T:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \ \#(8.4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \ \#(8.4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 171 \ \#(8.4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 171}{7^2(4^3 - 4)} = 0{,}697 > W_k = 0{,}67 \ \#(8.4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 8.4.2.

Таблиця 8.4.2 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	<	>	<	<	>	>	1,5
X1 і X3	<	>	>	>	<	<	>	>	1,5
X1 і X4	<	<	<	>	<	<	<	<	0,5
X2 і X3	<	<	>	<	<	<	>	<	0,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	<	<	<	<	>	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases} . \#(8.4.6)$$

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

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \#(8.4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \#(8.4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \#(8.4.9)$$

$$b_i' = \sum_{i=1}^{N} a_{ij} b_j \#(8.4.10)$$

Як видно з таблиці 8.4.3, різниця значень коефіцієнтів вагомості не перевищує 5%, тому більшої кількості ітерацій не потрібно.

Таблиця 8.4.3 – Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Різниця значень коефіцієнтів
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	

									вагомості
X1	1,0	1,5	1,5	0,5	,5	,28	6,25	,28	2%
X2	0,5	1,0	0,5	0,5	,5	,16	,25	,16	0%
X3	0,5	1,5	1,0	0,5	,5	,22	2,25	,21	5%
X4	1,5	1,5	1,5	1,0	,5	,34	1,25	,36	5%
Всього					6		9		

8.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об'єм пам'яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (Таблиця 8.5.1):

$$K_K(j) = \sum_{i=1}^{n} K_{i,j} B_{i,j}, \quad \#(8.5.1)$$

де n – кількість параметрів;

K_i – коефіцієнт вагомості i–го параметра;

$B_i^в$ – оцінка i–го параметра в балах.

Таблиця 8.5.1 – 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
F1	А	X1	1000	10	0,28	2,8
F2	А	X2	500	5	0,16	0.8
	Б	X2	280	10	0,16	1,6
F3	А	X3	5	5	0,21	1,05
F4	А	X4	1000	10	0,36	3,6

За даними з таблиці 8.5.1 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}] \quad \#(8.5.2),$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 2{,}8 + 0{,}8 + 1{,}05 + 3{,}6 = 8{,}25$$

$$K_{K2} = 2{,}8 + 1{,}6 + 1.05 + 3{,}6 = 9{,}05.$$

З обчислень вище можна побачити, що другий варіант F1(А)-F2(Б)-F3(А)-F4(А) є кращим у якого коефіцієнт технічного рівня має найбільше значення.

8.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

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

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \#(8.6.1)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

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

K_M – 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 степеня новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =40 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1.

Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.95. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 40 \cdot 1.8 \cdot 0.95 = 68.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =32 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$$Т_2 = 32 \cdot 0.9 \cdot 0.8 = 23.04 \text{ людино-днів.}$$

Оскільки трудомісткість всіх варіантів є однаковою, її можна об’єднати в одну групу:

$$Т0 = (68.4 + 23) \cdot 8 = 731{,}5 \text{ людино-години.}$$

В розробці беруть участь два програмісти з окладом 80000 грн, один аналітик в області кластеризації даних з окладом 50500. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.}, \#(8.6.2)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_{ч} = \frac{80000 + 80000 + 50500}{3 \cdot 21 \cdot 8} = 417{,}65 \text{ грн.} \#(8.6.3)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_{ч} \cdot Т_{i} \cdot К_{Д}, \#(8.6.4)$$

де $С_{ч}$– величина погодинної оплати праці програміста;

$Т_{i}$ – трудомісткість відповідного завдання;

$К_{Д}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$С_{ЗП} = 417.65 \cdot 731.5 \cdot 1.2 = 366\,613{,}17 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 366\,613{,}17 \cdot 0.22 = 80\,654{,}9 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ($С_{М}$)

Так як одна ЕОМ обслуговує одного програміста з окладом 80000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_{Г} = 12 \cdot М \cdot К_{З} = 12 \cdot 80000 \cdot 0{,}2 = 192\,000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_{Г} \cdot (1 + К_{З}) = 192000 \cdot (1 + 0.2) = 230\,400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 201\,600 \cdot 0{,}22 = 50\,688 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 60000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 60000 = 17\ 250 \text{ грн.},$$

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:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 60000 \cdot 0.05 = 3450 \text{ грн.},$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 10) \cdot 8 \cdot 0.85 =$$

$$= 1625{,}2 \text{ годин},$$

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:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1625.2 \cdot 0,4 \cdot 0.85 \cdot 4.79 = 2646,8 \text{ грн.},$$

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:

$$С_Н = Ц_{ПР} \cdot 0.67 = 60000 \cdot 0,67 = 40\ 200 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \#(8.6.5)$$

$$С_{ЕКС} = 230\ 400 + 50\ 688 + 17\ 250 + 3450 + 2646,8 + 40\ 200 = 344\ 634,8 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 344\ 634,8 / 1625,2 = 212,05 \text{ грн/год.}$$

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:

$$С_М = С_{М-Г} \cdot Т, \#(8.6.6)$$

$$С_М = 212,05 \cdot 731,5 = 115\ 114,575 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67, \#(8.6.7)$$

$$С_Н = 230\ 400 \cdot 0,67 = 154\ 368 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н, \#(8.6.8)$$

$$С_{ПП} = 230\ 400 + 50\ 688 + 115\ 114,575 + 154\ 368 = 550\ 570,575 \text{ грн.}$$

8.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj} / С_{Фj}, \#(8.7.1)$$

$$К_{ТЕР1} = 8,25/ 550\ 570,575= 1,49 \cdot 10^{-5},$$

$$К_{ТЕР2} = 9.05/ 550\ 570,575= 1,64 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 1,13 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації

програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,64 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мову програмування Python;
- використання бібліотек numPy/SciKit-learn
- інтерфейс користувача PyCharm;

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

8.8 Висновки до розділу

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту. Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

Проведений аналіз показав, що варіант створення плагіну за допомогою Python та використовуючи numPy/SciKit-learn є вигідним з економічної точки зору.

ВИСНОВКИ

В даній дипломні роботі було розглянуто проблему створення алгоритму для кластеризації текстових документів на основі методу k-найближчих сусідів.

Було проведено аналіз і порівняння з іншими, наявними алгоритмами кластеризації. А також проведено аналіз наявних пропозицій щодо використання даного методу для кластеризації даних. Втім не було виявлено, пропозицій як використовувати даний метод для кластеризації даних.

Тому було вирішено створити власний алгоритм, та перевірити його працездатність на текстових даних.

Створивши алгоритм і розробивши відповідну програму, що реалізує даний алгоритм, я не зміг виявити ніяких переваг над іншими більш простими алгоритмами, що вже існують, наприклад k-середніх. Хоча теоретично наш алгоритм має виділяти кластери більш складних форм, ніж алгоритм k-середніх.

Хоч і він справляється досить добре на невеликих даних, втім його повільна робота на трішки більших даних є величезним мінусом даного алгоритму

Втім як ми знаємо методі k-найближчих сусідів лежить в основі багатьох інших алгоритмів, наприклад, в області класифікації даних.

Тому алгоритм розроблений згідно з дипломним завданням з певними модифікаціями може мати продовження в області кластеризації даних, з певними модифікаціями.

ПЕРЕЛІК ПОСИЛАНЬ

1. Учасники проектів Вікімедіа. Кластеризація методом к–середніх – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Кластеризація_методом_к–середніх (дата звернення: 07.05.2023).

2. Учасники проектів Вікімедіа. Метод k-найближчих сусідів – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Метод_k-найближчих_сусідів (дата звернення: 07.05.2023).

3. Учасники проектів Вікімедіа. Однозв'язна кластеризація – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Однозв'язна_кластеризація (дата звернення: 07.05.2023).

4. Cosine similarity – text similarity metric – study machine learning. *Study Machine Learning – Study Machine Learning*. URL: https://studymachinelearning.com/cosine-similarity-text-similarity-metric/ (дата звернення: 14.05.2023).

5. Harrison O. Machine learning basics with the k-nearest neighbors algorithm. *towardsdatascience.com*. URL: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761 (дата звернення: 14.05.2023).

6. Lianne & Justin. Just into data. data cleaning in python: the ultimate guide (2020). *towardsdatascience.com*. URL: https://towardsdatascience.com/data-cleaning-in-python-the-ultimate-guide-2020-c63b88bf0a0d (дата звернення: 16.05.2023).

7. lingvolab. Кластеризация текстовых документов по семантическим признакам (часть первая: описание алгоритма). *Хабр*. URL: https://habr.com/ru/articles/324540/ (дата звернення: 18.05.2023).

8. *Medium*. URL: https://towardsdatascience.com/text-clustering-da5b8d83f866 (дата звернення: 18.05.2023).

9. NLTK :: nltk package. *NLTK :: Natural Language Toolkit*. URL: https://www.nltk.org/api/nltk.html (дата звернення: 18.05.2023).

10. NumPy documentation. *NumPy*. URL: https://numpy.org/doc/ (дата звернення: 14.05.2023).

11. Ralhan A. Self organizing maps. *medium.com*. URL: https://medium.com/@abhinavr8/self-organizing-maps-ff5853a118d4 (дата звернення: 28.04.2023).

12. Reuters Corpora @ NIST. *Text REtrieval Conference (TREC) Home Page*. URL: https://trec.nist.gov/data/reuters/reuters.html (дата звернення: 28.04.2023).

13. Scatter plots with a legend – Matplotlib 3.7.1 documentation. *Matplotlib – Visualization with Python*. URL: https://matplotlib.org/stable/gallery/lines_bars_and_markers/scatter_with_legend.html (дата звернення: 13.05.2023).

14. Scikit-learn: machine learning in Python. *scikit-learn: machine learning in Python – scikit-learn 0.16.1 documentation*. URL: https://scikit-learn.org/stable/documentation.html (дата звернення: 13.05.2023).

15. Sklearn.decomposition.PCA. *scikit-learn*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (дата звернення: 13.05.2023).

16. Sklearn.manifold.TSNE. *scikit-learn*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html (дата звернення: 13.05.2023).

17. UCI machine learning repository. URL: https://archive.ics.uci.edu/ml/index.php (дата звернення: 21.05.2023).

ДОДАТОК А

Зчитувач .SGM файлів

```
    import os
import re
from bs4 import BeautifulSoup


class SGMReader:
   def parse_sgm_file(self, filename):
       """
       Парсинг одного файлу .sgm
       :param filename: шлях до файлу
       :return: список текстів з файлу
       """
       with open(filename, 'r') as f:
           soup = BeautifulSoup(f, 'html.parser')
           texts = []
           for text in soup.find_all('text'):
               title = text.find('title').text if
text.find('title') else None
               dateline = text.find('dateline').text if
text.find('dateline') else None
               b = text.find('body').text if text.find('body')
else None
               body = re.sub('[^0-9a-zA-Z]+', ' ',
str(b)).lower()
               textex = {'title': title, 'dateline': dateline,
'body': body}
               texts.append(textex)
           return texts

   def parse_sgm_directory(self, directory_path):
       """
```

```
        Парсинг всіх файлів в дерикторії з розширенням .sgm
        :param directory_path: шлях до папки
        :return: список текстів з файлів
        """
        texts = []
        for filename in os.listdir(directory_path):
            if filename.endswith(".sgm"):
                file_path = os.path.join(directory_path, filename)
                text = self.parse_sgm_file(file_path)
                texts += text
        return texts

    def parse_sgm_file_list(self, file_list):
        """
        Парсинг файлів зі списку що мають розширення .sgm
        :param file_list: список шляхів до файлів
        :return: список текстів з файлів
        """
        texts = []
        for filename in file_list:
            file_path = os.path.join(filename)
            text = self.parse_sgm_file(file_path)
            texts += text
        return texts
```

ДОДАТОК Б

Клас кластеризатора

```
    import statistics
import numpy as np
import matplotlib.pyplot as plt
from klusterizerAdditionalFunc import SGMReader
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import KMeans
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer


class kNNclusterizer:
    def __init__(self, clusterNum=2, neighborNum=5, iterNum=2):
        self.clusterNum = clusterNum
        self.neighborNum = neighborNum
        self.iterNum = iterNum
        self.vocab = None
        self.clusterList = None
        self.nameList = None
        self.matrix = None
        self.nearNeigh = None
        self.textNum = 0

    @staticmethod
    def readSGM(path, folder=False, file=False, list=False):
        """
        Зчитування файлів з розширенням .sgm
        :param path: шлях до потрібних файлів/папки
```

```
        :param folder: True - якщо шлях веде до папки
        :param file: True - якщо шлях веде до файлу
        :param list: True - якщо заданий список шляхів до файлів
файлів
        :return: список текстів з файлів
        """
        sgm = SGMReader()
        if folder:
            text_list = sgm.parse_sgm_directory(path)
        elif file:
            text_list = sgm.parse_sgm_file(path)
        elif list:
            text_list = sgm.parse_sgm_file_list(path)
        return text_list

    def init_cluster(self, text_list):
        """
        Перетворює список текстів в більш структурований вигляд,
        а саме список словників де для кожного тексту є поля з
назвою,
        списком параметрів або термів із основної частини тексту
а також присвоює
        випадковий кластер для кожного тексту
        :param text_list: список текстів
        :return: список словників
        """
        n_col, n_row = text_list[0]['body'].toarray().shape
        dt = np.dtype([('name', 'U10'), ('coordinates',
np.ndarray, (n_col, n_row)), ('cluster', 'i4')])
        cluster_arr = np.empty(shape=[len(text_list), 1],
dtype=dt)
        for i in range(len(text_list)):
            random_cluster = np.random.randint(self.clusterNum)
            cluster_arr['name'][i] = text_list[i]['title']
```

```
            cluster_arr['coordinates'][i] =
text_list[i]['body'].toarray()
            cluster_arr['cluster'][i] = random_cluster
            # cluster_arr[i] = (text_list[i]['title'],
text_list[i]['body'], random_cluster)
        return cluster_arr

    def one_matrix_cluster_init(self, text_list):
        """
        Створює три структури даних, дві з яких списоки з назвами
і
        номером кластеру, а також матрицю з мараметрами або
термами з тексту

        :param text_list: список текстів
        :return: матриця термів/па
        """
        n_col, n_row = text_list[0]['body'].toarray().shape
        self.textNum = len(text_list)
        self.matrix = np.empty((len(text_list), n_row))
        self.clusterList = np.empty(len(text_list), dtype="int")
        self.nameList = np.empty(len(text_list), dtype="<U16")
        for i in range(len(text_list)):
            random_cluster = i % self.clusterNum
            # random_cluster = np.random.randint(self.clusterNum)
            self.nameList[i] = text_list[i]['title']
            self.clusterList[i] = random_cluster
            self.matrix[i] = text_list[i]['body'].toarray()
        return self.matrix

    def one_matrix_init(self, text_list):
        n_col, n_row = text_list[0]['body'].toarray().shape
        self.matrix = np.empty((len(text_list), n_row))
        for i in range(len(text_list)):
```

```
            self.matrix[i] = text_list[i]['body'].toarray()
        return self.matrix

    @staticmethod
    def body_vectorize(text_list):
        """
        Векторизує список текстів
        :param text_list: список текстів
        :return: веторизований список текстів
        """
        vectorizer = TfidfVectorizer()
        for text in text_list:
            text['body'] =
vectorizer.fit_transform([text['body']])
        return text_list

    def additional_text_prep_vectorize(self, text_list):
        """
        Функція виконує повну підготовку текстів до векторизації,
а саме
        лематизацію, видалення стопслів та векторизує методом TF-
IDF
        :param text_list: список текстів у вигляді словника зі
вмістом
        під ключем "body"
        :return: Векторизований список текстів
        """
        nltk.download('wordnet')
        lemmatizer = WordNetLemmatizer()
        nltk.download('stopwords')
        stop_words = list(stopwords.words('english'))
        vectorizer = TfidfVectorizer(stop_words=stop_words)
        only_texts = []
        for text in text_list:
```

```
            lem_text = [lemmatizer.lemmatize(word) for word in
text['body'].split(" ")]
            final_text = ' '.join(lem_text)
            only_texts.append(final_text)
        # create vocab
        self.vocab = vectorizer.fit(only_texts)
        for text in text_list:
            text['body'] = vectorizer.transform([text['body']])
        return text_list

    def visualise(self, matrix, reduction="TSNE"):
        """
        Відображає документи і їх взаємне розміщення у вигляді
точок на площині
        для кращого розуміння як працює алгоритм
        :param matrix: матриця параметрів текстових документів
        :param reduction: метод зменшення розмірності матриці для
візуалізації
        """
        if reduction == "TSNE":
            x = TSNE(n_components=2, perplexity=30,
learning_rate=200)
            plt.title("TSNE")
        else:
            x = PCA(n_components=2)
            plt.title("PCA")

        X_reduced = x.fit_transform(matrix)
        plt.scatter(X_reduced[:, 0], X_reduced[:, 1],
c=self.clusterList)
        plt.show()

    def build_nearest_matrix(self):
        """
```

```
        Будує матрицю найближчих сусідів
        :return: матриця найближчих сусідів
        """
        distances = cosine_similarity(self.matrix)
        self.nearNeigh = distances.argsort()[:,
1:self.neighborNum + 1]
        return self.nearNeigh

    def find_k_nearest_neighbors(self):
        """
        Обчислює найближчих сусідів до кожного елемента
        і на основі цього повертає список нових кластерів
        :return: список нових кластерів
        """
        neighbor_indices = self.build_nearest_matrix()
        future_clusters = np.empty(self.textNum, dtype="int")
        for i in range(self.textNum):
            near_clusters = [self.clusterList[ind] for ind in
neighbor_indices[i]]
            mode_cluster = statistics.mode(near_clusters)
            future_clusters[i] = mode_cluster

        return future_clusters

    def clusterize(self):
        """
        Цикл кластеризації
        :return: None
        """
        i = 0
        while i < self.iterNum:
            i += 1
            future_clust = self.find_k_nearest_neighbors()
            if np.array_equal(future_clust, self.clusterList):
```

```
                self.visualise(self.matrix)
                return
            self.clusterList = future_clust
            self.visualise(self.matrix)

    def returnClusters(self):
        """
        Формує словник, де ключ це номер кластеру, а
        значення список з назвами документів, що відносяться
        до кластеру
        :return: словник кластерів
        """
        l = {}
        for i in range(len(self.clusterList)):
            if self.clusterList[i] in l.keys():
                l[self.clusterList[i]].append(self.nameList[i])
            else:
                l[self.clusterList[i]] = [self.nameList[i]]
        return l
```

ДОДАТОК В

Функція main()

```
    from kNNclusterizer import kNNclusterizer
import nltk

if __name__ == '__main__':
    # Створення об'єкта класу кластеризатора
    k = kNNclusterizer(neighborNum=2, clusterNum=2, iterNum=4)
    # Зчитування файлів
    textlist = k.readSGM(list=True, path=["testFiles/reut2-
000.sgm", "testFiles/reut2-001.sgm"])
    # Векторизація текстів
    textVectorised = k.additional_text_prep_vectorize(textlist)
    # Початкова ініціалізація кластерів
    matrix = k.one_matrix_cluster_init(textVectorised)
    # Візуалізація початкової ініціалізації
    vis = k.visualise(matrix)
    # Кластеризація
    k.clusterize()
    # Повернення кластеризованих текстів у вигляді словника
    result = k.returnClusters()
```