МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **"ВИЯВЛЕННЯ ВИКОРИСТАНЬ ГРАФОВОЇ БАЗИ ДАНИХ GraphDB ЗА ДОПОМОГОЮ АВТОМАТИЧНОГО ДОБУВАННЯ ТЕРМІНІВ"**

Виконав студент 2 курсу, групи 8.1262
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та штучний інтелект
(назва освітньої програми)
Бармак В.С.
(ініціали та прізвище)
Керівник викладач кафедри комп'ютерних наук,
к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та штучний інтелект

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

_____ Шило Г.М.
(підпис)

" 23 " травня 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бармаку Віталію Степановичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Виявлення використань графової бази даних GraphDB за допомогою автоматичного добування термінів
керівник роботи Добровольський Геннадій Анатолійович, к.т.н.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від " 01 " травня 2023 року № 643-с

2. Строк подання студентом роботи 11.12.2023

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Постановка задачі, аналіз предметної області.
   2. Проектування.
   3. Реалізація та тестування.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 01.05.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.05.2023	
2.	Збір вихідних даних.	20.05.2023	
3.	Обробка методичних та теоретичних	26.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	20.09.2023	
5.	Розробка третього розділу.	01.10.2023	
6.	Оформлення та нормоконтроль	28.11.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	13.12.2023	

Студент _____ В.С. Бармак
(підпис) (ініціали та прізвище)

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

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота магістра “Виявлення використань графової бази даних GraphDB за допомогою автоматичного добування термінів”: 50 с., 5 табл., 10 джерел, 2 додатки.

GRAPHDB, FASTTEXT, МЕТОД К-СЕРЕДНІХ, CLUSTER, GOOGL SHCOLAR.

Об’єкт дослідження − процес автоматичного створення формальних моделей предметних областей.

Мета роботи: процес автоматичного виявлення зв’язків між концепціями.

Методи дослідження − моделювання, проектування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню дотичної до теми літератури та досліджень, а також огляду інструментів розробки.

У другому розділі детально розглянуто складові частини запропонованого методу.

Третій розділ присвячено реалізації та тестуванню запропонованого методу на прикладі корпусу статей, які згадують GraphDB − графову базу даних.

У роботі було запропоновано, досліджено та реалізовано метод виявлення зв’язків між автоматично вилученими методами, за допомогою моделі FastText та кластеризації методом К-середніх. Запропонований метод дозволяє знаходити пов’язані концепції, які є синонімами. Результати роботи методу можуть бути використані як частина процесу автоматичного складання та перевірки якості онтологій.

Новизна методу полягає у спільному застосуванні методів автоматичного вилучення термінів та подальшого використання мовної моделі FastText для пошуку зв’язків між термінами. У той час, як звичайне застосування мовної моделі полягає тільки в пошуку синонімів.

Отже, в результаті нашої роботи було створено зручний та ефективний веб-додаток алгоритму що дозволяє знаходити пов’язані концепції, які є синонімами.

SUMMARY

Master's Qualification Theses "Identifying uses of the GraphDB graph database using automatic term extraction": 50 pages, 5 tables, 10 sources, 2 supplements.

GRAPHDB, FASTTEXT, SCIENCE DIRECT, UML, K-MEANS METHOD.

The object of the research is a method of identifying relationships between automatically extracted methods using the Fasttext model and clustering using the K-means method.

The goal of the work is to develop a web application for implementing an algorithm that allows you to find related concepts that are synonyms.

Research methods – modeling, design, software, analytical.

The paper proposed, investigated and implemented a method for identifying relationships between automatically extracted methods using the Fasttext model and K-means clustering. The proposed method allows finding related concepts that are synonymous. The results of the method can be used as part of the process of automatic assembly and quality control of ontologies.

The novelty of the method consists in the joint application of methods of automatic extraction of terms and the further use of the Fasttext language model to search for connections between terms. While the usual application of the language model consists only in finding synonyms.

So, as a result of our work, a convenient and effective algorithm web application was created that allows you to find related concepts that are synonyms.

ЗМІСТ

ВСТУП

Формалізація знань та створення інформаційної моделі є першим та найважливішим етапом проектування будь-якої бази даних або бази знань. Але більшість наявних знань викладаються у вигляді текстових документів, зручних для сприйняття людиною і недостатньо формалізованих. Тому важливою є задача перетворення знань, викладених у відносно вільній текстовій формі, у формальні моделі, що можуть стати основою бази даних або бази знань.

Процес формалізації та моделювання обов'язково включає кроки виявлення концепцій і зв'язків між ними. Крок виявлення концепцій у текстах реалізується методами автоматичного виявлення термінів − достатньо детально дослідженими за останній час. В той же самий час методи автоматичного виявлення зв'язків між термінами розроблені набагато гірше.

У роботі запропоновано, досліджено та реалізовано метод виявлення зв'язків між автоматично вилученими методами, за допомогою моделі FastText та кластеризації методом k-середніх. Запропонований метод дозволяє знаходити пов'язані концепції, які не є синонімами. Результати роботи методу можуть бути використані як частина процесу автоматичного складання та перевірки якості онтологій.

Новизна методу полягає у спільному застосуванні методів автоматичного вилучення термінів та подальшого використання мовної моделі FastText для пошуку зв'язків між термінами. У той час, як звичайне застосування мовної моделі полягає тільки в пошуку синонімів.

Метою роботи: перевірити гіпотезу про можливість автоматичного виявлення зв'язків між термінами шляхом застосування мовних моделей, побудованих на основі дистрибутивної семантики.

Задачі:

1) Оглянути рішення та інструменти для реалізації системи;
2) Спроектувати та побудувати архітектуру системи;
3) Реалізувати метод виявлення зв'язків між термінами
4) Застосувати розроблений метод до пошуку термінів, зв'язаних із

"GraphDB".

1 ІСНУЮЧІ МЕТОДИ ВИЯВЛЕННЯ ПОВ'ЯЗАНИХ ПОНЯТЬ

1.1 Автоматичне створення онтологій

Сучасний стан справ окремих галузях комп'ютерних наук диктує необхідність залучення методів інженерії знань на вирішення широкого класу практичних завдань. Яскравим прикладом є ініціатива Semantic Web, основна мета якої – наділити величезні масиви даних, опублікованих у мережі Internet, більшою свідомістю, підвищити зручність роботи з цією інформацією. Одним із головних досягнень проекту Semantic Web стала розробка стандарту опису онтологій − OWL (Ontology Web Language), завдяки чому багато інженерів зі знань, програмістів та експертів отримали можливість використовувати загальні правила подання, зберігання та обробки онтологій.

Під онтологією в багатьох роботах розуміється формальний явний опис понять (часто званих класами) у предметній області, властивостей (іноді званих слотами) кожного поняття, що задають різні особливості та атрибути поняття, та обмеження на властивості (іноді званих фасетами) Онтологія визначає терміни, що використовуються для описи та подання галузі знань. Онтологія разом із набором індивідуальних випадків класів становить основу знань.

Сьогодні в теорії прийнято класифікувати онтологію за ступенем залежності від завдань або прикладної галузі, за мовою подання онтологічних знань та її виразним можливостям та іншим параметрам. Особливе місце займають онтології верхнього рівня (описують найбільш загальні концепти: простір, час, матерія, об'єкт, подія, дія і т. ін.) [3]. Існують онтології, орієнтовані предметну область [4-5]. Онтології, орієнтовані завдання, відбивають специфіку програми, що виконує конкретне завдання. Прикладні онтології описують концепти, які залежить як від онтології завдань, і від онтології предметної області.

Побудована онтологія предметної області може використовуватися для вдосконалення таких сфер діяльності як:

а) системи навчання. Для першого знайомства з предметною областю було б дуже корисно мати як “опорний сигнал” структуру цієї області, що легко сприймається. За допомогою онтології можна швидко знаходити посилання джерела інформації;

б) пошукові системи. Перехід від пошуку інформації за ключовими словами використання семантично значущих фрагментів текстів істотно полегшується, якщо використовується онтологія;

в) наукові дослідження. Велике значення має уніфікація термінології. Наявність онтології Про дозволить автоматизувати процес відстеження корисних даних та знань у потоці поточної інформації;

г) системний аналіз предметної галузі. Онтологія надає структуровану та частково формалізовану основу для проведення системного аналізу предметної галузі;

д) інтеграція даних та знань. При об’єднанні інформаційних баз онтологія допомагатиме встановлювати семантичну еквівалентність однакових фактів та понять, сформульованих у різних термінах.

Базовим етапом розробки онтології є визначення концептів предметної області, побудова тезауруса. Автоматизація цього етапу, як показує проведений аналіз, є складною і досі невирішеною в повному обсязі завданням. Пошук семантично близьких текстових документів, на основі яких може створюватися та деталізуватися онтологія предметної галузі, може вестися ефективно лише на основі базового списку термінів. При пошуку суттєвих труднощів виникають через проблеми синонімічності. У мові багато слів із загальним змістом. Хоча вони пишуться по-різному, але означають те саме. Прикладами синонімів є, наприклад, іменники “пластик” та “пластмаса”. Давно існують словники синонімів й у деяких пошукових системах

використовуються підвищення надійності пошуку. На жаль, синонімічність слів залежить від галузі знань, у яких вони застосовуються. Наприклад, висловлювання “надійний” і “ефективний” стосовно методів зберігання овочів можуть вважатися синонімами, а в області програмного забезпечення вони мають власний зміст.

Проведений аналіз показує, що на даний час для пошуку активно застосовуються семантичні пошукові механізми, однак у багатьох таких системах немає вбудованих списків термінів (тезаурій), які враховують синонімію понять предметної галузі. Для отримання найповніших результатів пошуку користувачам у пошукових запитах доводиться постійно перераховувати всі синоніми введеного терміну (наприклад, Росія – і Росія, і РФ).

За наявності тезаурусу термінів предметної області, користувачеві в пошуковому запиті достатньо лише один термін. Якщо в тезаурусі є список синонімів до введеного слова, то результати пошуку будуть присутні як документи, які містять слово, введене користувачем, так і документи, що містять слова-синоніми.

На жаль, через відсутність формалізованих словників термінів для конкретних предметних областей, автоматичне створення тезаурусу неможливе. Ручне складання тезаурус є досить трудомістким завданням, оскільки вимагає експертного аналізу значної кількості документів для виділення списку термінів предметної області, при цьому досить важко оцінити повноту отриманого списку. Для вирішення такого завдання необхідно використати автоматизоване створення списку термінів предметної галузі.

Аналіз існуючих підходів до пошуку синонімів показує, що ефективним з погляду автоматизації може бути використання корпусів електронних енциклопедичних текстів, згрупованих за категоріями, оскільки використання електронних словників синонімів обмежується мовними рамками або їхньою вузькою спеціалізацією, а також відсутністю словників для багатьох

предметних областей. Наявність категоризації статей і велика кількість самих статей у такому джерелі даних як, наприклад, Вікіпедія дозволяє отримати набір проблемно орієнтованих текстів практично на будь-яку тематику.

Метою даної є розробка алгоритмічного забезпечення для побудови списку семантично близьких слів на основі корпусу тематичних текстових документів.

Для досягнення цієї мети необхідно вирішити такі завдання:

а) провести аналіз основних етапів створення онтологій предметних областей, засобів та інструментаріїв їх розробки, можливостей систем семантичного пошуку як складової систем автоматизації проектування онтологій;

б) проаналізувати методи пошуку семантично близьких слів, які застосовуються при пошуку Інтернет-сторінок за запитами користувачів, вибрати базовий метод і розглянути метрики семантичної схожості документів, що застосовуються;

в) на основі базового методу розробити модифіковані алгоритми пошуку семантично близьких слів у корпусі текстових документів із гіперпосиланнями та категоріями;

г) розробити програмний комплекс для дослідження алгоритмів пошуку синонімів з використанням баз даних корпусів текстів;

д) вибрати показники чисельної оцінки семантичної близькості списку слів та оцінити працездатність запропонованих алгоритмів та вплив параметрів алгоритмів на результати пошуку.

1.2 Методи автоматизованої побудови онтологій

Для того, щоб вручну побудувати повну пов’язану онтологію з певної предметної області, необхідно витратити досить багато часу та ресурсів. Причина цього полягає в тому, що такі онтології повинні містити десятки

тисяч елементів, щоб бути придатними для вирішення широкого кола прикладних завдань, що виникають у цих предметних галузях (зокрема, аналізу наукових текстів заданої предметної галузі).

Ручна побудова онтології це довгий рутинний процес, який до того ж вимагає глибоких знань предметної області і розуміння принципів побудови онтологій. Зрозуміло, що повністю автоматизувати процес побудови онтологій неможливо – базові терміни та поняття мають бути запроваджені людиною-експертом. Проте процес подальшого побудови онтології можна організувати як навчання з урахуванням текстів заданої предметної області, впорядкованих за зростанням складності обробки. Ступінь складності обробки тексту може ґрунтуватися на різних критеріях, наприклад, за кількістю невідомих термінів, що зустрічаються в тексті, або за допомогою топологічного порядку дерева наукових праць, які посилаються один на одного.

Дослідженню проблем автоматизації процесів побудови онтологій присвячено багато робіт, серед яких [5-9]. У роботі [5] основні етапи процесу автоматизованої побудови онтології виділяються:

- визначення термів;
- визначення синонімів;
- визначення концептів;
- побудова таксономії концептів;
- визначення відносин;
- визначення правил.

Нижче наведені сучасні методи які пропонуються для вирішення кожного з цих завдань.

Виділення термів (як лінгвістичних реалізацій проблемно-залежних понять) є основою для вирішення наступних складніших завдань. Для автоматичного визначення термів має використовуватися лінгвістична обробка тексту для того, щоб визначити складені синтаксичні структури, які можна розглядати як самостійні терми. Іноземні дослідники обмежуються

побудовою зразків для пошуку синтаксичних структур та використанням статистичної обробки для включення лише “важливих” термів.

Визначення синонімів означає визначення семантичних варіантів терміну як у меж однієї мови, і між мовами (тобто, переклад терміна). Більшість робіт у цій галузі сфокусувалися на інтеграції із системою WordNet7, для отримання англійських синонімів, та EuroWordNet8, для двомовних та багатомовних перекладів синонімів та терміну. Однак, крім використання готових наборів синонімів, також працювали над алгоритмами визначення синонімів на основі кластеризації та пов’язаних методів. Окремим напрямом досліджень стало також використання статистичних оцінок для веб інформації.

Визначення концептів має забезпечувати:

- формулювання змісту поняття;
- набір прикладів поняття;
- визначення концептів;
- набір лінгвістичних реалізацій.

Найбільше досліджень щодо визначення концептів розглядають концепти як групи пов’язаних термінів. Альтернативними напрямками є дослідження концептів чи з погляду формальних і неформальних визначень. У цілому нині, виділення термів і синонімів вважатимуться підготовкою до визначення концептів загалом. Щодо автоматичного отримання таксономій із текстових даних, існує три основні парадигми. Перша полягає у використанні лексико-синтаксичних зразків. Друга полягає у використанні алгоритмів ієрархічної кластеризації. А третя походить з інформаційно-пошукового суспільства та базується на категоризації термінів як запропоновано.

Для визначення неірархічних відносин між концепціями використовується комбінування статистичного аналізу із лінгвістичним. Виділення правил виведення є найменш дослідженим питанням побудови онтологій. Деякі основні питання, пов’язані з автоматичним виділенням правил текстів, сформульовані у роботі.

Побудова онтології має логічно завершуватись її формальним представленням. Формальне подання онтології передбачає запис виявлених концептів та відносин однією з мов формального опису (OWL, KIF та ін.). Можливість автоматизації такого подання неодноразово підтверджена численними редакторами онтологій, що дозволяють генерувати формальні описи онтологій на основі їх таблично-графічних уявлень.

В процесі автоматичного створення онтологій наступним кроком після створення переліку концепцій є встановлення зв’язків між ними. Для встановлення зв’язків можна використати модель **фаст текст**, тому що вона навчається на n-грамах слів, а значить зберігає інформацію про спільне використання концепцій

1.3 Методи добування термінів (опис)

Потік текстової інформації постійно зростає, і різноманіття варіантів її представлення природною мовою змушує розширювати інструментарій для вилучення. Вилучення термінів (іноді також зване розпізнаванням термінів) є підзавданням вилучення інформації з текстів, цілого полягає в автоматичному вилучення релевантних слів і словосполучень із заданого корпусу текстів [1]. Незважаючи на різноманіття існуючих рішень, завдання вилучення термінів з текстів природною мовою продовжує залишатися актуальним і вимагає залучення нових фахівців. За її вирішенні можуть виникнути деякі проблеми.

- визначення межскладових термінів;
- розпізнавання лексичної одиниці як частини складеного терміна або як вільної лексичної одиниці;
- визначення лексичної одиниці як терміна в залежності від контексту та тематики тексту, у якому дана лексична одиниця використовується.

Існує безліч підходів до вирішення цих проблем. За останні 50 років автоматичне вилучення термінології з наукових текстів сформувалося як окрема галузь розробки алгоритмів та програмного забезпечення в рамках комп'ютерної лінгвістики, і в ній були отримані суттєві досягнення. Однак ці методи не дають абсолютних результатів і потребують подальшої розробки.

Одним із найбільш актуальних підходів до обробки текстів, та до вилучення термінології зокрема, можна вважати використання машинного навчання. У цьому підході ключовим питанням є вибір найбільш підходящої мовної моделі, яка після подальшого навчання зможе забезпечити кращу якість вилучення термінів із текстів наукових статей.

Огляд існуючих методів. Серед існуючих систем автоматичного вилучення термінів можна виділити низку загальних характеристик. Так, на вхід до системи зазвичай подається корпус текстів, на виході система формує списки термінів-кандидатів, які підлягають подальшій перевірці. Розглянемо різні методи та алгоритми для вилучення термінів з тексту, що застосовуються (окремо і в комбінаціях) на практиці.

Вилучення термінів із опорою на статистичні показники.

Імовірнісний метод вилучення термінів – це метод, заснований на даних про частотність словосполучення та про спільну зустрічальність словосполучень (можливе використання морфологічних шаблонів-фільтрів). Тобто такі системи працюють за рахунок виділення двох або більше лексичних одиниць, частота спільної зустрічальності яких більше деякого цього рівня.

Цей метод застосовується для отримання як термінології в цілому [2], так і тільки ключових слів для документа [3]. У систем, заснованих на такому підході, є слабке місце: при збільшенні довжини терміну падає частота його народження, в спеціалізованому корпусі обмеженого обсягу термін може зустрічатися один-два рази. Можна встановлювати низький поріг поєднання слів для включення його в контрольну групу, але тоді метод буде характеризуватись високою повнотою і, як наслідок, низькою точністю виділення термінів.

Вилучення термінів на основі правил. Системи, основу яких лежить цей метод, функціонують з допомогою підготовленого дослідниками докладного зведення правил. Залежно від правил їх можна поділити на три типи:

- системи, в яких за допомогою регулярних виразів та кінцевих автоматів задається виділення повторюваних структур термінологічних поєднань;
- системи, що використовують для автоматичної розмітки словники;
- системи, які використовують загальномовну інформацію (наприклад, про синтаксичну структуру речення, про часткову приналежність слів, що входять до неї), які виділяють базові мовні структури.

У статті [4] розглянуто метод, що спирається на інформацію про синтаксичну структуру пропозиції та часткову приналежність слів, що входять до нього. Інформація про синтаксичну структуру зберігається у вигляді шаблону речення (sentence pattern), а частеречна розмітка – у вигляді послідовності покажчиків на ту чи іншу частину мови (POS sequence of the sentence). Такий метод може використовувати термінологію, не вимагаючи корпусу розмічених даних (для холодного старту потрібен тільки словник шаблонів речень).

Дещо по-іншому застосовані методи на основі правил авторами статті [5], які представили двоетапний підхід до вилучення тематичної інформації, що відноситься до функціїціям деякого ПЗ, з посібника користувача. На першому етапі напівавтоматично витягується термінологія предметної області на основі лінгвістичних шаблонів, а потім застосовуються методи, що спираються на попередньо отриману термінологію предметної області та мовну інформацію про структуру речень

Метод на основі правил часто застосовується зі спиранням на онтології та словники. У статті [6] описано метод вилучення багатокомпонентних термінів, який спирається на великі лексичні ресурси у вигляді електронних

словників та перетворювачів із кінцевим числом станів для моделювання різних синтаксичних структур термінів. Така сама технологія використовується для лематизації витягнутих багатокомпонентних термінів. Вилучені та лематизовані багатослівні терміни фільтруються, щоб відхилити помилково запропоновані леми, а потім ранжуються шляхом введення показників, що поєднують лінгвістичну та статистичну інформацію (C-Value, T-Score, LLR та Keyness). Системи вилучення термінології з урахуванням правил дають хорошу точність, але здебільшого лише конкретної мови через особливості його граматики.

Вилучення термінології за допомогою методів машинного навчання.

Останнім часом широкого визнання набули методи, в основі яких лежить використання алгоритмів машинного навчання. Як правило, вони мають на увазі два етапи: вилучення ланцюжків слів, які потенційно можуть бути термінами, і подальше визначення терміна і уточнення його кордонів. У випадку тексти діляться на значні інтервали, які часто збігаються з окремими словами і звані токенами. Кожному токену відповідає вектор. З безліччю таких векторів далі працює модель.

Залежно від специфіки завдання при обчислення ваги моделі робиться наголос на передбачення або токена в контексті, або контексту для токена. Попередньо навчена на об’ємному корпусі текстів модель виділяє терміни-кандидати, які можуть бути верифіковані. Для визначення, чи є послідовність слів терміном, можуть бути використані різні ознаки: загальнолінгвістична інформація (часткова приналежність слів, головне слово фрази, кількість іменників у фразі та ін) [7], статистичні (довжина фрази, TF, IDF, TF- IDF або частота народження фрази в корпусі наукових текстів, як запропоновано в [8]) і гібридні ознаки [9].

Застосовуючи методи машинного навчання на вирішення завдання вилучення термінології з текстів, можна їх розширювати і вдосконалювати. У статті [10] запропоновано використовувати згладжування меж інтервалів при

векторизації. Векторизація в span-based моделі проводиться не для окремих токенів, а для інтервалів, для яких згодом обчислюється ймовірність відповідності сутності. Шляхом перерозподілу ймовірностей сутностей з анотованих інтервалів до сусідніх автори досягли кращих результатів при розв’язанні задачі вилучення сутностей.

У системі HAMLET [11] кожному кандидату пропонується обчислити ряд із 152 ознак і далі навчити класифікатор бінарного дерева рішень. Кандидати визначаються з урахуванням їх частини промови, але закономірності входження слова в термін визначаються автоматично з урахуванням даних навчання.

Традиційно багато систем екстракції термінів засновані на гібридному підході: спочатку застосовується лінгвістична фільтрація для визначення синтаксично правдоподібних термінів-кандидатів, потім кандидати оцінюються та класифікуються з використанням статистичних функцій, спеціальних метрик чи машинного навчання. Застосований підхід спирається на методи машинного навчання, доповнені вилученням термінології за допомогою словника та валідацією на основі правил.

Вилучені терміни можуть бути використані як елементи тематичного покажчика або ключових слів для автоматичного індексування документів. Також автоматичне вилучення термінів з наукових текстів спрощує процес створення словника термінів або складання онтології для певної предметної галузі, що особливо актуально для галузей, що швидко змінюються.

2 ЗАСТОСУВАННЯ КЛАСТЕРИЗАЦІЇ АВТОМАТИЧНО ЗНАЙДЕНИХ ТЕРМІНІВ ДЛЯ ВИЯВЛЕННЯ ПОВ'ЯЗАНИХ ПОНЯТЬ

2.1 Навчання моделі fastText

Пропонований метод виявлення пов'язаних термінів складається з наступних етапів:

а) збір корпусу наукових статей на тему (GraphDB);

б) навчання моделі фасттекст на корпусі зібраних статей;

в) автоматичне вилучення термінів із корпусу зібраних статей;

г) кластеризація термінів за допомогою подання fastText методом К-середніх.

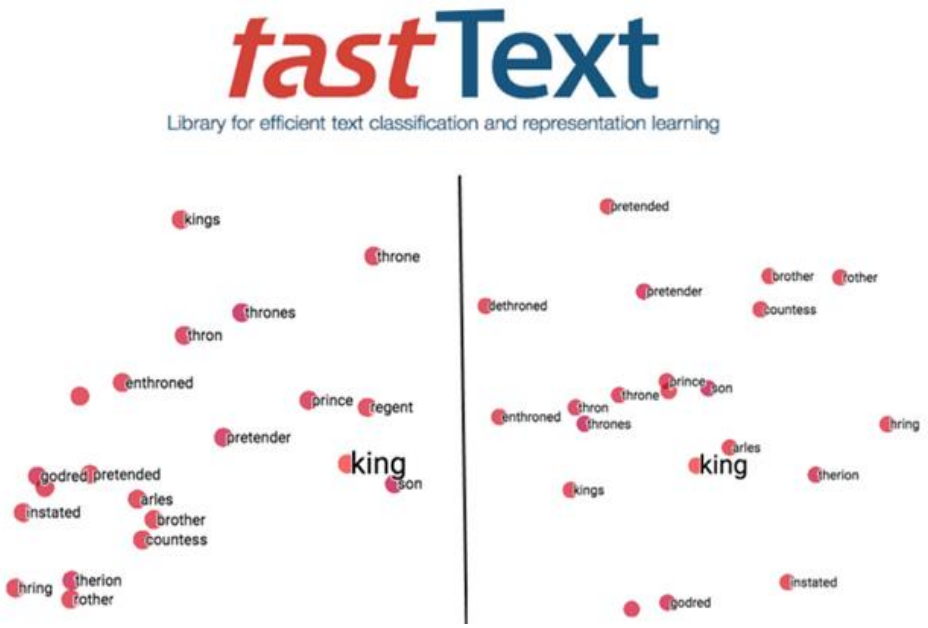

Рисунок 1 − Графічне представлення FastText

FastText (рис. 1) − це бібліотека, яка навчає розпізнавати слова та розбирати речення. Вона має векторні представлення заданих слів, підготовлені за допомогою класифікатора, тобто. алгоритм машинного навчання, який сегментує слова в інструмент навчання, підтримує кілька мов, включаючи англійську, німецьку, іспанську, французьку та чеську. Щоб краще обробляти структури даних із кількома категоріями, FastText використовує

ієрархічний класифікатор, який організовує категорії в деревоподібну структуру, а не в плоску структуру. FastText може ідентифікувати текст і розуміти такі вирази за виділенням груп. FastText було розроблено групою дослідників Facebook AI Research, до якої також увійшов творець Word2vec Томас Міколов (який у той час перейшов із Google у Facebook), використовуючи комбінацію моделей Skipgram і CBOW (Continuous Bag-of-Words) для отримання вектора представлення словами.

Отже, чим відрізняється FastText? Висока продуктивність порівняно з іншими пакетами та моделями. У моделі векторного шуму використовується Skip-gram з негативною дискретизацією. Негативна вибірка − це метод негативної вибірки для навчання векторної моделі, тобто, визначити ряд слів, які не є сусідами в контексті Для кожного позитивного шаблону (коли текстові слова розташовані близько одне до одного, наприклад, “м’який кіт”), ми створюємо більше негативів (“м’яке залізо”, “м’який радіосигнал”, “слабка втеча”) Chinom має загальний випадковий вибір від 3 до 20 слів. Цей випадковий вибір кількох екземплярів потребує менше комп’ютерного часу та дозволяє FastText працювати швидше.

Skip-gram ігнорує синтаксис, але деякі мови мають складний синтаксис, наприклад німецька. Тому тимчасову модель включили в основну. Структура короткого рядка складається з одного рядка (n-грами) n між 3 і 6 цифрами від початку рядка до кінця рядка, включаючи весь сам рядок, наприклад, і блокування слова з n = 3 —3. <n-gram lock < za, zam, amo, mok, ok> відображатиметься по порядку>. Так, наприклад, існує різниця між послідовністю <jam> у слові zam − і n-грамою zam від слова zamok . Це дозволяє вам працювати зі словами, які модель ніколи раніше не зустрічала.

Ділення на n-грами створює більший сигнал (тобто ви отримуєте більшу таблицю для інформації). Це може уповільнити навчання моделі з цими функціями. Хешування атрибутів (спеціальний процес, який може використовувати рядки символів однакової довжини для кодування об’єктів різного розміру) для визначення розміру атрибута. Токени отримують хеш-

індекс, який допомагає прискорити їх читання. Класифікація заснована на моделі лінійної класифікації, яка за архітектурою схожа на модель CBOW. Чим більше число класів, тим довше працює лінійна модель. Для оптимізації класифікатора використовується ієрархічний софтмакс на основі алгоритму кодування Хаффмана.

Дерево класів − це гілка, на вершині якої знаходиться більшість класів, а його дочірні класи − рідкісні класи. Наприклад, клас “organisms” матиме дочірні вузли таких класів, як “тварини” та “рослини”, від першого батьківського вузла до останнього дочірнього вузла, і ймовірність класу для кожного вузла встановлюється, тому дочірній вузол завжди має меншу ймовірність ніж батьківський вузол.

Нижче ми розглянемо, як можна використовувати бібліотеку в практичній роботі.

Як використовувати FastText в проектах NLP. Список усіх доступних мов на даний момент дорівнює 157. Тобто беремо готові лексичні вектори і використовуємо їх для представлення термінів з нашого корпусу.

Виявилося, що деяких слів не буде в попередньо вибраних векторах FastText, тому що навчання проходило в корпусі. Однак ми знаємо, що FastText вивчає слова та символи. Таким чином, модель здатна генерувати достатньо векторів навіть для слів, які раніше не зустрічалися.

За замовчуванням FastText використовує вектор слів із розмірністю 300, але за бажання розмір можна змінити за допомогою утиліт FastText. У деяких бібліотеках Python також є екземпляри FastText, які працюють незалежно від операційної системи. І, звичайно, модель працює в Google Collab.

GitHub підтримує встановлення бібліотеки та її використання в репозиторії проекту FastText. Нейронні мережі або класифікатори, такі як логічна регресія, можна використовувати для навчання шляхом перетворення підготовлених векторних представлень слів у бажані функції. Так чи інакше, використовуючи логістичну регресію, ви можете перевірити свій алгоритм автоматичної обробки тексту без збоїв у роботі.

Найпростіший спосіб використання FastText. Спробувати прорахувати векторне представлення FastText можна, використовуючи пакет gensim у Python. Там є модель FastText. Імпортується модель так:

from gensim.models import FastText

Перед тим, як подати моделі ваш корпус, не забудьте зробити передобробку тексту. Спочатку текст очищається від усіх символів, крім літер, наводиться в нижній регістр, робиться стемінг або лематизація. У модели текст подається токенізованим (розбитим на слова). Після цього можемо розпочинати будувати модель.

У моделі є гіперпараметри. Ось опис деяких із них:

- розмір векторні моделі. Якщо встановити 100 − кожне слово в корпусі буде представлено у вигляді 100-мірного вектора і тому подібне;
- розмір вікна. Цей параметр визначає, скільки сусідніх слів вважається частиною контексту. Якщо виставити 40, алгоритм візьме 40 слів спереду від слова і 40 слів ззаду від слова;
- найменша допустима кількість символів у слові, для якого створюватиметься векторне уявлення; так можна усунути частотні, але не дуже значущі слова типу спілок і прийменників;
- вибір архітектури векторної моделі: skip-gram або CBOW. Sg=1 переключить на модель skip-gram. За замовчуванням значення параметра 0 означає використання CBOW (метод представления).

Так як корпус даних не розбитий мітками, використаним методом “неконтрольоване навчання”, щоб в подальшому отримати вектора слов сусідніх із зразком.

а) При навчанні моделі використовувалася model="cbow" (“неконтрольоване навчання”).

Модель cbow (рис. 2) передсказує цільове слово відповідно до його контексту. Контекст представлений як набір слів, що містяться у вікні фіксованого розміру навколо цілого слова.

Давайте проиллюстрируем эту разницу на примере: учитывая пропозицію “Poets have been mysteriously silent on the subject of cheese” (“Поєти загадкова мовчали про сир”) і цільове слово “silent” (мовчати). Модель cbow бере все слово у вікні, наприклад “subject” або “mysteriously” (been, mysteriously, on, the), і використовує суму їх векторів для прогнозування ціли.

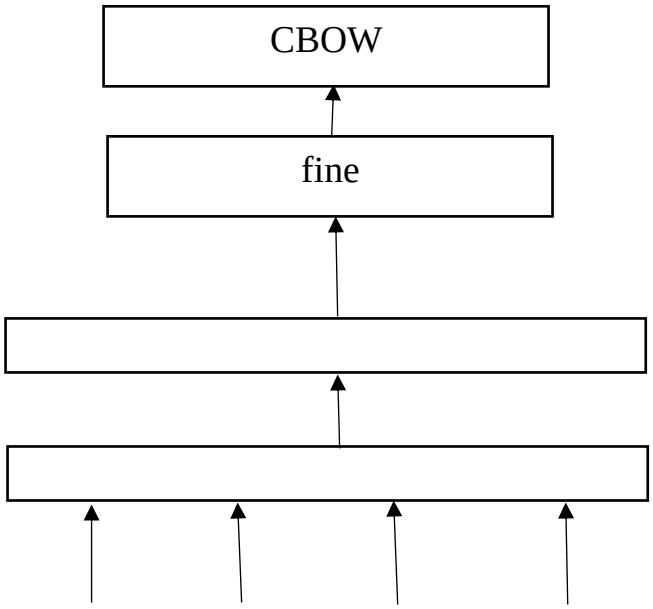

Рисунок 2 − Модель cbow

б) minCount − це мінімальна кількість входжень слів у програмі = 2;

в) У програмі також використовується біграма, wordNgrams=WORDGRAMS = 2. Використання біграмми може підвищити продуктивність моделей слов замість просто унігрaмм. Це особливо важливо для класифікації завдань, де важен порядок слів, таких як аналіз настроювань.

Позначимо “біграммою” конкатенацію двох послідовних токенів або слів. Також часто говоримо про n-грамми для позначення конкатенації будь-яких n-послідовних токенів. Наприклад, у пропозиції “Останній пончик ночі” уніграммами є “останній”, “пончик”, “із”, “це” і “нічь”. Біграмми: “Останній пончик”, “пончик”, “із” і “ніч”. Біграми особливо цікаві, тому що для більшості пропозицій ви можете відновити порядок слів, просто взявши набір n-грамм. Використовується наступна біграма, встановивши вихідну пропозицію:

“все зовні”, “я є”, “із жевательної гумки”, “зовні” і “є все”. Слово прийнято називать уніграммой;

д) minn − мінімальна довжина char ngram і у програмі = 3;

е) maxn − максимальна довжина char ngram і у програмі = 50;

ж) lr − швидкість навчання = 0.2;

з) dim = VECTOR_DIM (розмір векторів слів) = 100;

Розмір dim контролює розмір векторів: чим вони більше, тим більше інформації вони можуть зібрати, але для вивчення потрібно більше даних. Але якщо вони занадто великі, їх тренувати складніше і повільніше. За замовчуванням використовуємо 100 розмірів, але популярні будь-які значення в діапазоні 100–300;

и) epoch= EPOCHS кількість епох = 25;Метою моделі FastText є представлення слів у багатовимірному просторі.

2.2 Вилучення термінів

Беремо тексти та навчаємо модель.

Supmodel = fasttext.train_unsupervised(input=".\\data\\tmp.txt", model="cbow", minCount=2, wordNgrams=WORDGRAMS, minn=3, maxn= 50, lr= 0.2, dim= VECTOR_DIM, epoch= EPOCHS)

Отримаємо список найближчих за змістом термінів

wordList=model.get_nearest_neighbors(KEY_PHRASE,k=500)

Беремо список термінів і кластеризуємо їх використовуючи веторне для кожного терміну уявлення fastText (model.get_word_vector(**термін**)) і вивчаємо кластестери, що виходять, щоб перевірити гіпотезу що всередині одного кластера знаходяться пов’язані концепції.

2.3 Кластеризація слів на основі уявлення про fastText методом К-середніх

Кластеризація методом k-середніх (англ. k-means clustering) − популярний метод кластеризації, − сортує об’єкти в однорідні групи. Він був відкритий математиком Г’юго Стейнхаусом у 1950-х роках і майже напевно Стюартом Ллойдом у 1950-х роках. Він набув особливої популярності після публікації роботи МакКвіна (1967).

Мета методу полягає в тому щоб розділити n спостережень на k кластерів, так щоб кожне спостереження належало до кластера з найближчим до нього середнім значенням. Цей підхід повністю базується на мінімізації суми квадратів відстаней між кожним висловом і серединою його кластера, тобто ознакою.

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

де $d -$ метрика, $x_i - i -$ ий об’єкт даних, а $m_j(x_i) -$ центр кластера,
якому на $j -$ ій ітерації приписаний елемент x_i

У нас є колекція спостережень (об’єктів), кожна з яких має набір значень відповідно до кількості атрибутів. Відповідно до цих критеріїв об’єкт має багатовимірний простір. Дослідник визначає, які групи необхідно сформувати. k-спостережень вибираються випадковим чином, які на цьому кроці розглядаються як центри кластерів. Кожен зразок “віднесений” до однієї з n-груп – до тієї, яка має найменшу відстань. Новий центр кожного кластера зараховується як елемент, якість якого обчислюється як середнє арифметичне цієї кількості атрибутів елементів, що входять до цього кластера (повторюйте кроки 3-4), доки центр кластера не стане стабільним (тобто його дисперсія в середині була максимально збільшена. Кількість груп вибирається на основі плану дослідження. Якщо він недоступний, рекомендується спочатку створити 2 групи, потім 3, 4, 5 і порівняти отримані результати.

Принцип дії алгоритму полягає в пошуку таких центрів кластерів та наборів елементів кожного кластера при наявності деякої функції Ф(°), що виражає якість поточного розбиття множини на k кластерів, коли сумарне квадратичне відхилення елементів кластерів від центрів цих кластерів буде найменшим:

Принцип роботи алгоритму полягає в знаходженні таких центральних кластерів і кластерів елементів перед функцією Ф(°), яка задає характер поточних розбиттів множини на k кластерів, а всі в квадратних кластерах зміщення від центру цих кластерів буде дуже мало:

$$V = \sum_{i=1}^{k} \sum_{x_{j \in} S_i} (x_j - \mu_i)^2$$

де $k -$ число кластерів, $S_i -$ отримані кластери, $i = 1, 2, \dots,$
$k, \mu_i -$ центри мас векторів $x_j \in S_i$

Центри кластерів вибираються довільно під час першого запуску алгоритму, потім відстань до центру обчислюється ітераційно для кожного елемента в наборі, де кожен елемент пов'язаний із найближчим центром. Процес перерозподілу є типовим.

Алгоритм методу "Кластеризація за схемою к-середніх":

− вибрати k інформаційних точок як центри кластерів поки не завершиться процес зміни центрів кластерів;

− зіставити кожну інформаційну точку з кластером, відстань до центра якого мінімальна;

− переконатися, що в кожному кластері міститься хоча б одна точка. Для цього кожний порожній кластер потрібно доповнити довільною точкою, що розташована "далеко" від центра кластера;

− центр кожного кластера замінити середнім від елементів кластера;

− кінець.

Основною перевагою методу k-середніх є його простота та швидкість. Метод k-середніх кращий для кластеризації великої кількості спостережень,

ніж метод ієрархічного кластерного аналізу (де дендограми перевантажені та втрачають видимість).

Недоліком простого методу є те, що він не враховує кореляційний стан елементів групи, тому в метод та його нечіткі аналоги (методи нечітких k-середніх) були внесені різні модифікації, які на першому кроці алгоритму , елемент множини може належати до кількох груп.

Незважаючи на очевидні переваги методу, він має серйозні недоліки:

− результат класифікації сильно залежить від початкових позицій кластерних центрів

− алгоритм чутливий до викидів, які можуть викривлювати середнє

− кількість кластерів має бути заздалегідь визначена дослідником

Метод k-середніх дуже простий і прозорий, тому успішно використовується в різних галузях − комерційна класифікація, географія, астрономія, сільське господарство і так далі.

Результатом роботи методу є список слів. Їх аналіз дозволяє перевірити гіпотезу про те, що в кластері існують терміни, що описують пов’язані поняття.

3 ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ

3.1 Метод бібліографічного виявлення та відбору

Для інформації використовувалася пошукова система Google Academy. Пошук у Google Shcolar: База даних GraphDB. Відібрано перші 100 незалежних статей. Весь матеріал був об'єднаний в один рукопис розміром 4,4 мегабайта.

Розділ 3 формулює IT-методологію для ідентифікації та відбору бібліографій на основі розробленої моделі та дозволяє створювати бібліографічні описи рекомендацій і наукових результатів, а також необхідного розміру. Опис технології включає її цілі, бізнес-середовище, контекст, структури даних і цілі, процедури пошуку та вибору, а також компоненти. Діаграма реалізації технології UML показана на рисунку 3.

Описано розроблену методику визначення та відбору літератури, яка складається з таких взаємопов’язаних частин.

а) удосконалений спосіб відбору первинних видань, який відрізняється від паралелізму модифікованими рукописами;

б) Удосконалений підхід до імовірнісної тематичної моделі текстових документів, яка відрізняється від близьких аналогів використанням головних компонент для визначення тематичної квантифікації.

в) Покращений метод керування ітераціями “сніжної кулі”, який відповідно відрізняється застосуванням імовірнісної тематичної моделі до текстових документів та наявністю критерію фіксованої точки;

Рисунок 3 – Діаграма діяльності інформаційної технології бібліографічного виявлення та відбору

г) Покращений метод аналізу мережі цитування, який відрізняється вдосконаленим методом перетворення препринтів між циклами мережі цитування

д) Найпростіший спосіб отримати повний текст документів – пошук у вільно доступній літературі;

е) Покращений метод виявлення пасивного сліду, який є унікальним завдяки використанню кінцевого автомата Ахо-Хоракіка для виявлення перехідних подій протягом тривалого періоду часу.

ж) Метод визначення великої кількості слів.

Кожен із перерахованих пунктів є реалізацією відповідної частини розробленої інформаційної технології та описує питання, необхідні для практичного застосування на основі прикладів, розроблених раніше.

Вони визначають, як ефективність інформаційних технологій значною мірою перетворюється на менш важливі публікації.

3.2 Навчання моделі FastText

Кластеризація − це процес поділу великої кількості об’єктів на менші групи, які називаються кластерами. Кластеризація, як математичний алгоритм, має велике застосування в багатьох сферах: вона походить від біологічних наук, таких як біологія та фізіологія, і закінчується маркетингом та пошуковою оптимізацією в соціальних мережах

Існує багато алгоритмів кластеризації, але метод k-середніх буде розглянуто нижче, оскільки він коротший і простий для розуміння.

Кластеризація методом k−середніх:

Вихідним завданням буде розподіл довільної кількості n−вимірних точок по k кластерів.

а) випадковим чином створюються k точок, надалі називатимемо їх центрами кластерів;

б) для кожної точки ставиться відповідно до найближчого до неї центр кластера;

в) обчислюються середні арифметичні точки, що належать до певного кластера. Саме це значення стають новими центрами кластерів;

г) кроки б) і в) повторюються до тих пір, поки перерахунок центрів кластерів приноситиме плоди. Як тільки вираховані центри кластерів збігатимуться з попередніми, алгоритм буде закінчено.

Приступимо до реалізації алгоритму:

Вихідні дані алгоритму:

n − кількість рядків;

k − кількість кластерів;

dim − Розмірність точок (простору).

Вихідні дані алгоритму:

cluster − двомірний масив розмірністю dim * k, що містить k точок − центри кластерів;

cluster_content − масив, що містить у собі k масивів − масивів точок, що належать відповідному кластеру.

```
def clusterization ( array, k ):
        n = len (array)
        dim = len (array [ 0 ])

        c luster = [[ 0 for i in range (dim)] for q in range (k)]
        cluster_content = [[] for i in range (k)]

        for i in range (dim):
                for q in range (k):
                        cluster[q][i] = random.randint ( 0 , max_cluster_value )

        cluster_content = data_distribution (array, cluster)
```

Змінні завдання. Первинні центри кластерів створені за допомогою бібліотеки random, max_claster_value − константа, що задає зразкові межі вихідної множини;

За допомогою функції data_ditribution () здійснено первинне розподілення точок за кластерами. Розглянемо цю функцію Детальніше :

```
def data_distribution ( array, cluster ):
    cluster_content = [[] for i in range (k)]

    for i in range (n):
        min_distance = float ( ' inf ' )
        situable_cluster = − 1
        for j in range (k):
            distance = 0
            for q in range (dim):
                distance += (array[i][q]−cluster[j][q])** 2

            distance = distance**( 1 / 2 )
            if distance < min_distance :
                min_distance =distance
                situable_cluster = j

        cluster_content [ situable_cluster ].append(array[i])

    return cluster_content
```

Для кожного рядка розраховується відстань до кожного центру кластерів. Тут застосовується стандартний алгоритм:

а) за початкову найкоротшу відстань (min_distance) береться незрівнянно велике зі значеннями точок число;

б) потім відбувається обчислення відстані центру кожного кластера

$$p = \sqrt{\sum_{i=1}^{n} (x_n - x_n^{\text{ценироїд.}})^2}$$

– обчислення відстані між точкою та центром у n−мірному просторі.

в) якщо обчислена відстань менша за мінімальну, то мінімальна відстань прирівнюється до обчисленого і точка прив'язується до цього кластера (situable_cluster);

г) після обробки точки, масив cluster_content у вибраний кластер (situable_cluster) кластер вкладається значення точки.

Функція повертає масив cluster_content .

Надалі, як і належить, дана послідовність дій звертається до циклу:

```
privious_cluster = copy.deepcopy (cluster)
    while 1 :
        cluster = cluster_update (cluster, cluster_content , dim)
        cluster_content = data_distribution (array, cluster)
        if cluster == privious_cluster :
            break
        privious_cluster = copy.deepcopy (cluster)
```

Даний цикл цілісним чином описує крок 4 з опису алгоритму k−середніх (див. вище).

Після розподілу точок центрами кластерів відбувається перерозподіл вже центрів кластерів по прив’язаних до них точках. Розглянемо функцію cluster_content () докладніше :

```
def cluster_update ( cluster, cluster_content , dim ):
    k = len (cluster)
    for i in range (k): # за і кластерами
        for q in range (dim): # за параметрами q
            updated_parameter = 0
            for j in range ( len ( cluster_content [i])):
                updated_parameter += cluster_content [i][j][q]
            if len ( cluster_content [i]) != 0 :
```

```
                    updated_parameter = updated_parameter / len (
cluster_content [i])
                        cluster[i][q] = updated_parameter
    return cluster
```

Для кожного кластера, для кожного з n вимірів обчислюється нове значення за допомогою нехитрого середнього арифметичного:

− складаються всі значення; сума ділиться на кількість точок у кластері;

− кластер набуває оновленого значення.

На цьому місці алгоритм закінчує свою роботу. Повний алгоритм виглядає так:

```
def clusterization ( array, k ):
    n = len (array)
    dim = len (array [ 0 ])

    cluster = [[ 0 for i in range (dim)] for q in range (k)]
    cluster_content = [[] for i in range (k)]

    for i in range (dim):
        for q in range (k):
            cluster[q][i] = random.randint ( 0 , max_cluster_value )

    cluster_content = data_distribution (array, cluster)

    privious_cluster = copy.deepcopy (cluster)
    while 1 :
        clu ster = cluster_update (cluster, cluster_content , dim)
        cluster_content = data_distribution (array, cluster)
        if cluster == privious_cluster :
```

break

privious_cluster = copy.deepcopy (cluster)

Таким чином, ми виконали необхідні та достатні умови для аналізу та реалізації кластеризації методом k−середніх.

3.3 Вилучення термінів

Програма вилучення термінів на вході отримує тексти статей, її результатом є список термінів упорядкований за релевантністю, наприклад, як представлено в таблиці 1.

Таблиця 1 – GraphDB Ontotext

№	Терміни упорядковані за релевантністю
1	**2**
1.	GraphDB
2.	AMME
3.	Models
4.	modeling language
5.	model elements
6.	graph databases
8.	modeling tool
9.	Modeling Method Engineering
10.	Resource Description Framework
11.	ADOxx
12.	Karagiannis
13.	showcase modeling language

Продовження таблиці 1

№	Терміни упорядковані за релевантністю
14..	RDF graphs
15.	Information Systems Engineering
16.	ADOxx metamodeling platform
17.	OWL inference patterns
18.	Ontology
19.	candidate business partners
20.	conceptional schema
21.	diagrammatic contents
22.	interoperability mechanism
23.	modeling environment
24.	ontology engineering methodologies
25.	knowledge
26.	Ontology-Based Process Modeling
27.	modeling method engineering
28.	knowledge management systems
29.	requirements-driven modeling methods
30.	case AMME acts
31.	data management technology
32.	Professional Judicial Knowledge
33.	modeling method
34.	business process management
35.	Springer Nature SciGraph
36.	handling many-to-many relationships
37.	production process phases
38.	modeling language notation
39.	showcase cross-model reasoning

3.4 Результати кластеризації

Застосовувався метод К-середніх, який отримував на вхід згенеровані моделі FastText розмірністю 100 (прописати параметри кластеризації)

На вхід отримував уявлення витягнутих термінів, що обчислюються за допомогою моделі FastText. Результатом кластеризації є кластери слів, наприклад кластер, що включає термін GraphDB та інший кластер найближчий до GraphDB (табл. 2).

Таблиця 2 – FastText

№	Кластер 0	Кластер 1
1	**2**	**3**
1.	RDF4J	graphx
2.	SPARQL	nosql
3.	trillion	macmillan
4.	gstore a graph	multigraphs
5.	PUK Parliament’s	hypergraphdb
6.	guting	infinitegraph
8.	BigOWLIM	graphql
9.	jdbcrdf4j	jsp
10.	Ontotext GraphDB	graphlog
11.	allegrograph	verlag
12.		2opt3gb

Вручну проаналізувано список перших 10 термінів, їх значення описано в в табл. 3 та табл. 4.

Таблиця 3 – Аналіз Кластерів 0

№	Терміни кластера 0:	Пояснення
1	**2**	**3**
1.	RDF4J	RDF4J − бібліотека, клієнт SPARQL
2.	SPARQL	SPARQL − мова запитів до GraphDB
3.	trillion	trillion − trillion + тестова БД
4.	gstore a graph	gstore a graph – механізм запитів на основі Sparql, один з аналогів
5.	PUK Parliament's	PUK Parliament’s Служба даних парламенту (PUK) на базі GraphDB
6.	guting	Guting − один із авторів та засновників
7.	BigOWLIM	BigOWLIM − стара назва GraphDB
8.	jdbcrdf4j	JDBC RDF4J − протокол SPARQL
9.	Ontotext GraphDB	Ontotext GraphDB − високоефективна та надійна графічна база даних із підтримкою RDF і SPARQL, сумісна зі стандартами W3C, яка може допомогти вам створити великі графи знань.
10.	allegrograph	AllegroGraph − горизонтально розподілена база даних графів і документів

Таблиця 4 – Аналіз Кластерів 1

№	Терміни кластера 1:	Пояснення
1	**2**	**3**
1.	graphx	*GraphX* − призначений для розподіленої обробки графів. Може працювати як у середовищі кластера Hadoop під управлінням YARN, так і без компонентів ядра Hadoop, підтримує кілька розподілених систем зберігання − HDFS, OpenStack Swift, Amazon S3
2.	nosql	NoSQL – це нереляційна база даних, схожа на GraphDB .
3.	macmillan	Видавництво, що видавало підручники з GraphDB
4.	multigraphs	Структура даних
5.	hypergraphdb	HyperGraphDB − база даних на основі графів, система розроблена спеціально для проектів, що використовують можливості штучного інтелекту та семантичного вебу.
6.	infinitegraph	InfiniteGraph − це крос-платформна, масштабована, включаюча хмарні технології база даних, спроектована для підтримки високої пропускну здатність. Використовується для пошуку онтологій (прихованих відносин)
8.	graphql	GraphQL – мова запитів даних та мова маніпулювання даними з відкритим вихідним кодом для побудови веб-орієнтованих програмних інтерфейсів. GraphQL був розроблений як внутрішній проект компанії Facebook у 2012 році, яка у свою чергу використовує GraphDB
9.	jsp	JSP (JavaServer Pages) − сервлети можуть використовуватися для звернення до GraphDB
10.	graphlog	1.Програма для вивчення рукописного тексту; 2. API для Python до графових баз даних
11.	verlag	Springer-verlag (з нім. “Видавництво “Шпрінгер”) − міжнародна видавнича компанія, яка видавала документацію з GraphDB
12.	2opt3gb	Сервера на яких запускається GraphDB

ВИСНОВКИ

У роботі було запропоновано, досліджено та реалізовано метод виявлення зв’язків між автоматично вилученими методами, за допомогою моделі FastText та кластеризації методом К-середніх. Запропонований метод дозволяє знаходити пов’язані концепції, які є синонімами. Результати роботи методу можуть бути використані як частина процесу автоматичного складання та перевірки якості онтологій.

Новизна методу полягає у спільному застосуванні методів автоматичного вилучення термінів та подальшого використання мовної моделі FastText для пошуку зв’язків між термінами. У той час, як звичайне застосування мовної моделі полягає тільки в пошуку синонімів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бахрушин В.Є. “Методи аналізу даних”. Запоріжжя, Класичний приватний університет, 2011, с. 117.
2. Dharma E.M., Ford L.G., H.L. Hendric Spits W., Soewito B. "The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification". Computer Science Department, BINUS Graduate Program – Doctor of Computer Science, Bina. Nusantara University, Jakarta 11480, No 2Indonesia. Journal of Theoretical and Applied Information Technology. 31st January 2022. Vol.100. pp. 349−359.
3. Bojanowski P., Grave E., Joulin A., and Mikolov T., “Enriching Word Vectors with Subword Information,” in Transactions of the Association for Computational Linguistics, Dec. 2017, vol. 5, pp. 135–146, doi: 10.1162/tacl_a_00051.
4 Bhattacharjee, J. FastText Quick Start Guide: Get started with Facebook's library for text representation and classification. Book. Packt Publishing Ltd, 2018. P. 194.
5 Pennington J., Socher R., and Manning C., “Glove: Global Vectors for Word Representation,” in Proceedings of the 2014. Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, vol. 31, no. 6, pp. 1532–1543, doi: 10.3115/v1/D14-1162.
6 Festus E., Ayetiran, Bačovský D, Lupták D., Štefánik M., Sojka P. “One Size Does Not Fit All: Finding the Optimal Subword Sizes for FastText Models across Languages”. In Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021), , Held Online. INCOMA Ltd. pp. 1068–1074.
7 Hartmann M., Kementchedjhieva Y., and Sogaard A. 2018. “Why is unsupervised alignment of English embeddings from different algorithms so hard?” In Proceedings of the 2018 Conference on Empirical Methods in Natural

Language Processing, Brussels, Belgium. Association for Computational Linguistics. pp. 582–586.

8. Тоцька О.Л. Автоматизація методу К-середніх кластерного аналізу за допомогою програмного пакета StatSoft Statistica 6.0. Інтеграція країн з перехідною економікою у світовий економічний простір : стан і перспективи : матеріали міжнар. наук. студ.-асп. конф., Львів, 13–14 трав. 2005 р. Львів. : Вид-во ЛНУ ім. І. Франка, 2005. С. 372–373.
9. Lande D.V., Snarskii A.A., Yagunova E.V., and Pronoza E., “The Use of Horizontal Visibility Graphs to Identify the Words that Define the Informational Structure of a Text”, In: Proceedings of the 12th Mexican International Conference on Artificial Intelligence, 2013, pp. 209−215.
10. Qorina E. S., Zamhari A., Hasan H., Hulliyah K., and Saepudin D. “Comparative Analysis of the Performance of the Fasttext and Word2vec Methods on the Semantic Similarity Query of Sirah Nabawiyah Information Retrieval System: A systematic literature review,” in 2020 8th International Conference on Cyber and IT Service Management (CITSM), Oct. 2020, pp. 1–4, doi: 10.1109/CITSM50537.2020.9268827.

ДОДАТОК А

Код програми

```
import string
import fasttext
import numpy as np
from numpy.ma import copy
import os.path

from klusterization import K_clusterization, Find_word_index, sort_claster

KEY_PHRASE  ="GraphDB"

MODEL_NAME           = ".\\data\\article.model"

#TEXT_CORPUS_NAME      = ".\\data\\corpus1.txt"

TEXT_CORPUS_NAME      = ".\\data\\dataset.txt"

VECTOR_DIM = 100 #розмір вектора слова
CLUSTER_NUM =10  # кілікість кластерів
EPOCHS = 25
WORDGRAMS = 2
NEIGHDORS = 500 #схожіх слів


def clean_file(file_name,out_file_name):
    string.punctuation = string.punctuation + '“' + '”' + '-' + '’' + '‘' + '—'
    string.punctuation = string.punctuation.replace('.', '')
    # Loads the data and preprocesses data and stores corpus in raw_text
    raw_text = open(file_name, encoding='utf8').read().lower()

    file_nl_removed = ""
    for line in raw_text:
        line_nl_removed = line.replace("\n", " ")
        # removes newlines
        file_nl_removed += line_nl_removed

    file_p = ""
    for char in file_nl_removed:
```

```
            if char not in string.punctuation:
             file_p +=char
            else:
             file_p += " "
        file_nl_removed=""
          # removes all special characters
        string.punctuation = string.punctuation + '.'
        file_q = ""
        for char in file_p:
            if char not in string.punctuation:
                file_q += char
            else:
                file_q += " "

        out_text = open(out_file_name, encoding='utf8',mode = "w")
        out_text.write(file_q)
        out_text.close()

    def train_models(text_name):
        clean_file(text_name, ".\\data\\tmp.txt")
        supmodel = fasttext.train_unsupervised(input=".\\data\\tmp.txt",
model="cbow", minCount=2, wordNgrams=WORDGRAMS, minn=3, maxn= 50,
lr= 0.2, dim= VECTOR_DIM, epoch= EPOCHS)
        os.remove(".\\data\\tmp.txt")
        supmodel.save_model(MODEL_NAME)


    def save_model(model,savepath):
        model.save_model (savepath)

    if not os.path.isfile(MODEL_NAME):
          train_models(TEXT_CORPUS_NAME)

    model=fasttext.load_model(MODEL_NAME);

    print("")
    #model.predict(KEY_PHRASE)

    # получим список ближайших по смыслу соседей
    wordList=model.get_nearest_neighbors(KEY_PHRASE,k=NEIGHDORS)

    # формуемо масив векторов ціх слів
    vectors=np.zeros((len(wordList),VECTOR_DIM))
    i = 0
    for w in wordList:
```

```
    print(w[1]," ",w[0])
    vector = model.get_word_vector(w[1])
    j=0
    for v in vector:
        vectors[i][j] =v
        j=j+1
    i=i+1

#формуемо кластери
clasters = K_clusterization(vectors, CLUSTER_NUM)
# сортіруемо кластери
clasters = sort_claster(model.get_word_vector(KEY_PHRASE), clasters)

print("\nword clasters")

i=0
for c in clasters:
  print("  claster: ",i," distance =",c[1])
  for vect in c[0]:
    index = Find_word_index(vect,vectors)
    if index >=0 :
     print(wordList[index][1])
  print("\n")
  i=i+1
```

ДОДАТОК Б

Кластерний аналіз

```
import math
from random import random, randrange

import numpy as np
from numpy.ma import copy

# кластерний аналіз
def cluster_equ(claster1, claster2):
    res = True
    i = 0
    for v in claster1:
        j = 0
        for c in v:
            if c != claster2[i][j]:
                res = False
                break
            j = j + 1
        i = i + 1
    return res

def K_clusterization(array, k):
    n = array.shape[0]
    dim = array.shape[1]

    def data_distribution(array, cluster):
        cluster_content = [[] for i in range(k)]

        for i in range(n):
            min_distance = float('inf')
            situable_cluster = -1
            for j in range(k):
                distance = 0
                for q in range(dim):
                    distance += (array[i][q] - cluster[j][q]) ** 2

                distance = distance ** (1 / 2)
                if distance < min_distance:
```

```
                min_distance = distance
                situable_cluster = j

            cluster_content[situable_cluster].append(array[i])
        return cluster_content
    def cluster_update(cluster, cluster_content, dim):
        k = len(cluster)
        for i in range(k):  # по i кластерам
            for q in range(dim):  # по q параметрам
                updated_parameter = 0
                for j in range(len(cluster_content[i])):
                    updated_parameter += cluster_content[i][j][q]
                if len(cluster_content[i]) != 0:
                    updated_parameter = updated_parameter / len(cluster_content[i])
                cluster[i][q] = updated_parameter
        return cluster

    cluster = [[0 for i in range(dim)] for q in range(k)]
    cluster_content = [[] for i in range(k)]
    max_cluster_value = 1.0
    for i in range(dim):
        for q in range(k):
         cluster[q][i] = randrange(0, max_cluster_value)

    cluster_content = data_distribution(array, cluster)

    privious_cluster = copy(cluster)
    while 1:
        cluster = cluster_update(cluster, cluster_content, dim)
        cluster_content = data_distribution(array, cluster)
        if  cluster_equ(cluster, privious_cluster) :
         break
        privious_cluster = copy(cluster)
    return cluster_content
########################################

#сортируем кластеры по расстоянию от слова(вектора) vector
def sort_claster(vector, clusters):

  def cluster_center(cluster):
   n = 0
   center = np.zeros(vector_size)
   for vector in cluster:
     center = center + vector
     n = n+1
```

```
  if n>0: center = center*(1.0/n)
  return center

 def distance(v1, v2):
  dist = 0.0
  i = 0
  for x in v1:
   dist += (x - v2[i]) ** 2
   i += 1
  return math.sqrt(dist)

 def find_nearest_cluster(v): # int
   i = 0
   index = -1
   min_dist = float('inf')
   for cluster in clusters:
      center = cluster_center(cluster)
      dist = distance(v,center)
      if dist < min_dist:
        index = i
        min_dist = dist
      i += 1
   return index

 def f_distance_sorting(cluster_tuple):
     return  cluster_tuple[1]

 vector_size = vector.size
 cluster_content = [(c,  distance(vector, cluster_center(c))) for c in clusters]

 cluster_content.sort(key=f_distance_sorting)
#  return [c[0] for c in cluster_content]
 return cluster_content
def vector_equ(v1,v2):
   i=0
   for v in v1:
      if v != v2[i] :
         return False
      else:
       i=i+1
   return True

def Find_word_index(vector, vectors):
   i = 0
   for v in vectors:
```

```
    if vector_equ(vector,v):
        return i
    else:
        i=i+1
return -1
```