МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра прикладної математики

Магістерська робота

АНАЛІЗ СОЦІАЛЬНИХ МЕРЕЖ І ПОШУК ІНФОРМАЦІЇ

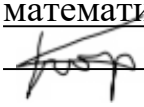
Виконав: студент групи ПМпМ-22с
спеціальності
113 - “Прикладна математика”
(шифр і назва спеціальності)

Скорупський О. О.
(підпис)

Керівник: доц. кафедри прикладної математики, канд. ф.-м. наук Ящук Ю. О.

(підпис)

Рецензент: ас. кафедри обчислювальної математики, канд. ф.-м. наук Борачок І. В.



(підпис)

ЛЬВІВ – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики_____
Кафедра прикладної математики_____
Спеціальність 113 - "Прикладна математика"_____
(шифр і назва)

«ЗАТВЕРДЖУЮ»
Завідувач кафедри Ящук Ю. О._____

"____"_____ 20___ року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

_____Скорупському Олександру Олеговичу_____
(прізвище, ім'я, по батькові)

1 .Тема роботи: Аналіз соціальних мереж і пошук інформації_____

керівник роботи Ящук Юрій Олександрович, доцент кафедри прикладної математики, кандидат фізико-математичних наук_____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від "22" вересня 2021 року №6

2. Строк подання студентом роботи 16 травня 2022 року_____

3. Вихідні дані до роботи: постановка задачі та її формалізація.

Дослідження методів побудови систем інформаційного пошуку.

Дослідження можливостей систем інформаційного пошуку.

Створення прототипу системи та застосування його для контентного аналізу соціальної мережі.

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)
   1) Вступ
   2) Базові поняття та побудова проєкту
   3) Завантаження та очищення даних
   4) Представлення та перетворення даних
   5) Створення індексу
   6) Контентний аналіз індексу
   7) Висновки
   8) Список використаних джерел

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
   1) Таблиці для візуальної демонстрації принципу роботи різних методів.
   2) Графічні зображення для візуальної демонстрації етапів розробки та принципу роботи різних методів.

3) Знімки екрану модулів програми, окремих частин коду та деяких результатів роботи програми.

6. Консультанти розділів роботи

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис, дата**	
		завдання видав	**завдання прийняв**
1-8	к. ф.-м. наук Музичук Ю.А.	жовтень 2021	квітень 2022

7. Дата видачі завдання: 02.09.2021

Календарний план

№ з/п	**Назва етапів магістерської роботи**	**Строк виконання етапів роботи**	**Примітка**
1.	Обговорення та формулювання завдань, об'єкту та мети дослідження. Аналіз актуальності проблеми.	вересень 2021	
2.	Дослідження літератури на тему розробки систем інформаційного пошуку. Огляд існуючих рішень.	жовтень 2021	
3.	Отримання облікових даних дослідника Twitter. Розробка модулів для завантаження даних.	листопад-грудень 2021	
4.	Фільтрація, токенізація, лемматизація, стеммінг та нормалізація даних.	січень 2022	
5.	Створення індексу.	лютий 2022	
6.	Апробація індексу. Вибір функції ранжування. Контентний аналіз. Побудова часових рядів.	березень-квітень 2022	
7.	Оформлення магістерської роботи.	травень 2022	

Студент _____ **Скорупський О. О.**
(підпис)

Керівник роботи _____ **Ящук Ю. О.**
(підпис)

ОЦІНОЧНИЙ ЛИСТ

магістерської роботи студента Скорупського Олександра Олеговича
прізвище, ім'я, по-батькові
групи ПМПм-22с факультету прикладної математики та інформатики

Відгук наукового керівника

Тематика

X	Комп'ютерне моделювання
X	Чисельні методи
	Оптимізація процесів
	Системне програмування
X	Бази даних
	Навчальні програми
	Веб-проектування
	Інше

Зміст

Максимальна кількість балів		
4	3	Складність, повнота розкриття дослідження
4	3	Наукова новизна, елементи творчості
4	4	Самостійність виконання, систематичність роботи
4	4	Якість та складність програм

Оформлення

4	4	Стиль, грамотність Ілюстративний матеріал Відповідність вимогам до роботи
Сума балів	18	

Коментарі

Студент Скорупський О.О. дослідив питання побудови систем інформаційного пошуку на прикладі відкритих даних із соціальної мережі Twitter. Використавши різні методи для обробки текстових даних, він побудував відповідний індекс у системі Lucene та дослідив його роботу, продемонструвавши результати алгоритмів ранжування даних. Робота виконана на належному рівні, була продемонстрована на науковій конференції та заслуговує на оцінку Відмінно.

Науковий керівник: к.ф.-м.н. доцент Ящук Ю.О. 

Оцінка рецензента _____б. Оцінка за захист _____б. СУМА БАЛІВ _____

Голова ЕК

РЕЦЕНЗІЯ

на магістерську роботу

студента Скорупського Олександра Олеговича
прізвище, ім'я, по-батькові

групи ПМпМ-22с факультету прикладної математики та інформатики

	Тематика
	Комп'ютерне моделювання
	Чисельні методи
x	Оптимізація процесів
	Системне програмування
	Бази даних
	Навчальні програми
	Веб-проектування
x	Інше

Максимальна кількість балів		Зміст
6	5	Складність
6	4	Наукова новизна
6	6	Повнота розкриття дослідження
6	5	Якість та складність програм
		Оформлення
2	2	Стиль, грамотність
2	2	Ілюстративний матеріал
2	2	Відповідність вимогам до роботи
Сума балів	26	

Коментарі

В роботі розглянуто основні методи створення систем інформаційного пошуку (ІП). Детально описано збирання і аналіз даних, алгоритм побудови системи та її реалізацію. Усі етапи формування системи ІП та використовувані моделі чітко проаналізовані, крім того деякі з них проілюстровані відповідними фрагментами коду. Варто зазначити, що більшість моделей вже були реалізовані, а програмна реалізація полягала у правильному використанні і конфігурації існуючих бібліотек та методів. Крім того для деяких етапів не описано програмну реалізацію, наприклад, для стемінгу та лемматизації. Проте, враховуючи детальний аналіз усіх моделей і схеми загального алгоритму побудови системи ІП, рекомендую до захисту оцінку "відмінно".

Рецензент **канд. фіз.-мат. наук Борачок І.В.**

Львівський національний університет імені Івана Франка

Подання
голові екзаменаційної комісії
щодо захисту магістерської роботи

Направляється

студент Скорупський Олександр Олегович

до захисту магістерської роботи

за спеціальністю_____113 - “Прикладна математика”_____

на тему _Аналіз соціальних мереж і пошук інформації_____

Магістерська робота і рецензія додаються.

Декан факультету: Дияк Іван Іванович _____
(підпис)

Довідка про успішність
_____Скорупського О. О._____

за період навчання на факультеті прикладної математики та інформатики
з 20___ року до 20___ року повністю виконав навчальний план
з таким розподілом оцінок за:
національною шкалою:
відмінно_____%, добре_____%, задовільно_____%;
шкалою ECTS:
A_____%; B_____%; C_____%; D_____%; E_____%

Секретар факультету _____ _____
(підпис) (прізвище та ініціали)

Висновок керівника магістерської роботи

Студент (ка) Скорупський О.О. виконав всі завдання, які були поставлені перед ним на цю роботу. Він дослідив питання побудови системи для пошуку та подальшого аналізу інформації, отриманої із популярної соціальної мережі Twitter. При цьому студент використав доступні методи для обробки тексту та подальше його збереження в індексі в системі Lucene. Побудований індекс досліджено використавши різні методи ранжування результатів. Робота виконана на належному рівні, була продемонстрована на науковій конференції та заслуговує на відмінну оцінку.

Керівник роботи _____
(підпис)

"15" травня 2022 року

Висновок кафедри про магістерську роботу

Магістерська робота розглянута.

Студент _____Скорупський О. О._____
(прізвище та ініціали)

допускається до захисту даної роботи в Екзаменаційній комісії.

В. о. завідувача кафедри _____ _____Ящук Ю. О._____
(підпис) (прізвище та ініціали)

"_____"_____ 2022 року

ЗМІСТ

1. Вступ

1.1. Об’єкт дослідження

Інформаційний пошук є процесом отримання ресурсів, які відповідають певному інформаційному запиту, із неструктурованої сукупності цих ресурсів. Автоматизовані системи інформаційного пошуку використовуються для зменшення так званого “інформаційного перевантаження”. Об’єктом дослідження цієї магістерської роботи є система інформаційного пошуку. З допомогою системи інформаційного пошуку збудованої, наприклад, на базі даних із соціальної мережі можна робити контентний аналіз цієї соціальної мережі. Найвідомішим прикладом систем інформаційного пошуку можна назвати пошукові системи в Інтернеті. Етапи розвитку системи ІП у поєднанні з методами контентного аналізу досліджуються в цій роботі та застосовуються до проблеми аналізу соціальних мереж.

1.2. Історичне підгрунтя

До 1970-х років було продемонстровано, що кілька різних методів пошуку добре працювали на невеликих корпусах текстів, таких як колекція Кренфілда, що скаладалась з кількох тисяч документів (Cranfield collection).[1] Великі пошукові системи, такі як система Lockheed Dialog, почали використовуватися ще на початку 1970-х років. У 1992 році Міністерство оборони США разом з Національним інститутом стандартів і технологій (NIST) виступили одними зі спонсорів Text Retrieval Conference (TREC).[2] Метою цього було об’єднання спільноти вчених, які працювали над методами інформаційного пошуку, надавши інфраструктуру, необхідну для оцінки методологій текстового пошуку у дуже великій колекції текстів. Це стало каталізатором дослідження методів, які масштабуються до величезних масивів текстових даних. Впровадження пошукових систем в Інтернеті ще більше посилило потребу у великомасштабних системах пошуку.

Зростання соціальних мереж призвело до формування нової галузі дослідження та нових даних для досліджень. За останні роки було створено багато алгоритмів для обробки природної мови та пошуку інформації.

1.3. Мета роботи

Метою магістерської роботи є створення впорядкованого і зрозумілого конвеєру для формування системи інформаційного пошуку та подальший контентний аналіз соціальної мережі. Для створення системи інформаційного пошуку необхідно мати набір текстових даних. Оскільки дані для систем інформаційного пошуку часто є неструктурованими та без визначених правил формування, цей набір потрібно підготувати та очистити для ефективного використання. В цій роботі буде досліджено процес розробки системи ІП, етапи підготовки вхідних даних та методи контентного аналізу. Дана система базується на відкритих даних взятих з потоку твітів соціальної мережі Twitter.

1.4. Методи дослідження

В магістерській роботі також будуть досліджені різні методи обробки текстових даних: токенізація, стемінг, лематизація, нормалізація, тощо. Завданням токенізації є відокремлення слів від розділових знаків, цифр, комплексів літер та цифр, інтернет-адрес, нікнеймів, знаків, тощо. Завдання в методів стемінгу, лематизації та нормалізації схоже, але відрізняється глибиною пропрацювання даних та підходами до вирішення проблеми. Наприклад, стемінг є процесом скорочення слова до його основи шляхом відкидання допоміжних частин, таких як закінчення чи суфікс. Лемматизація у свою чергу є складнішим методом, що базується на визначенні основи слова. А нормалізація тексту є ще більш комплексним підходом до обробки тексту при якому, наприклад, враховується контекст і частина мови конкретного слова. Індексація є одним з ключових принципів на яких базується система ІП. Процедура індексації дозволяє перевести великий набір текстових даних в легкодоступну і швидку для обробки модель зберігання даних. Також важливою характеристикою систем інформаційного пошуку є впорядкованість за важливістю окремих слів у контексті документа.

Важливість слова можна обчислювати, наприклад, з допомогою статистичної оцінки пропорційної кількості вживань цього слова у документі та обернено пропорційної кількості документів в яких слово вживається серед усієї колекції.[4]

1.5. Процес інформаційного пошуку

Процес інформаційного пошуку починається із запиту до системи. Запити є формальними заявами про інформаційні потреби, наприклад, рядки пошуку у веб-пошукових системах. Вони узгоджуються з інформацією з бази даних, однак під час інформаційного пошуку запит не є однозначним ідентифікатором конкретного об’єкту в колекції. На відміну від класичних SQL запитів до бази даних, під час інформаційного пошуку отримані результати можуть відповідати запиту повністю або лише частково, тому вони зазвичай впорядковуються за своїм ступенем відповідності.[5] Таке ранжування результатів є ключовою відмінністю інформаційного пошуку в порівнянні з пошуком у базі даних. Часто самі документи не зберігаються безпосередньо в системі інформаційного пошуку, а натомість представлені в системі метаданими.

Більшість систем інформаційного пошуку обчислюють числову оцінку того, як добре кожен об’єкт у інформаційній базі відповідає запиту, і ранжують об’єкти відповідно до цього значення. Після цього система видає об’єкти з найвищим рейтингом.

2. Базові поняття та будова проекту

2.1. Базові поняття та терміни

Система інформаційного пошуку - це система, яку люди можуть використовувати для отримання інформації. Основна мета пошукової системи полягає в тому, що вона приймаючи певні дані, повинна надавати інформацію. Ця мета означає, що пошукова система повинна розібратися в даних, які вона отримує, щоб надати щось, що може бути легко зчитано користувачами. Користувачі рідко потребують великої кількості даних на певну тему. Вони часто шукають конкретну інформацію, і їх би задовольнила лише одна відповідь, а не сотні чи тисячі результатів, які потрібно перевіряти.[7]

Інформаційний пошук (ІП) — це пошук матеріалів (зазвичай документів) неструктурованого характеру (зазвичай тексту), який задовольняє потребу в інформації з великих колекцій (зазвичай зберігаються на комп'ютерах). "Зазвичай текст", але це можуть бути і зображення, відео, дані, аудіо і тд. Зазвичай неструктурований (тобто без попередньо визначеної моделі), але, наприклад, XML, Voicexml, RDF, HTML є більш структуровані, ніж звичайний TXT або PDF. [8]

Насправді майже жодні дані не є неструктурованими. Наприклад, навіть слайди презенацій мають чітко визначені зони, такі як заголовок та текст.

Основні функції системи інформаційного пошуку зазвичай включають:

- індексування - ефективне завантаження та зберігання даних, що дозволяє швидко їх отримувати;
- обробка запитів - забезпечення функціональності пошуку, щоб кінцевий користувач міг виконувати пошук;
- ранжування - подання та сортування результатів відповідно до певних метрик для найкращого задоволення інформаційних потреб користувачів.

З допомогою систем ІП можна вирішувати не тільки завдання ІП, але й багато інших[10]:

- Кластеризація - маючи набір документів, потрібно згрупувати їх у кластери на основі їх вмісту.
- Класифікація - маючи набір тем, а також новий документ, потрібно вирішити до якої теми (тем) цей документ належить (наприклад, відсортовування спаму від звичайних листів).
- Аналіз ключових думок - аналіз та пошук ключових думок та настроїв в певному тексті чи колекції текстів (наприклад, аналіз соціальних мереж)

Повнотекстовий пошук: метод пошуку, який порівнює запит дослівно з кожним словом у тексті, не розрізняючи граматичну функцію (значення, позицію) різних слів.

2.2. Будова проекту та використані технології

Для імплементації проекту була обрана об’єктно орієнтована мова програмування Java. Це компільована мова програмування тому вона достатньо швидка, а також має досить розвинуту структуру вбудованих та сторонніх бібліотек, які недоступні для інших мов програмування.

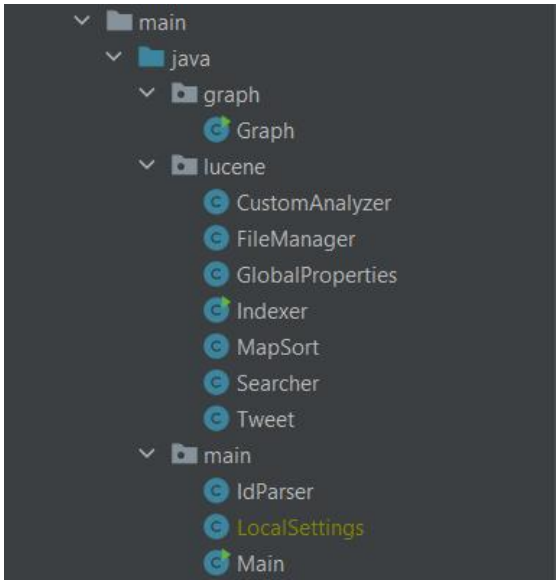


Рис. 2.2.1 Модулі проєкту

Середовище розробки - IntelliJ IDEA 2021. Програма для автоматизації роботи та автоматичного складання проєкту - Apache Maven.

Для розробки та автоматизації деяких описаних нижче методів та алгоритмів підготовки даних до індексації я використовував Apache Lucene. Apache Lucene - безкоштовна бібліотека з відкритим кодом для повнотекстового пошуку. Реалізована на Java, підтримується Apache Software Foundation і випускається під ліцензією Apache Software. Для роботи з файловою системою, деяких операцій з часом тощо були використані стандартні бібліотеки мови Java та бібліотеки Apache. А для завантаження даних через Twitter API було використано Twitter4j.

На даний момент проєкт містить три Java пакети: **main, graph** і **lucene** (Рис. 2.2.1)**.** Кожен модуль містить свою Main підпрограму для окремого запуску. Також проєкт містить директорію “data” у якій у піддиректоріях зберігаються проміжні дані та результати обробки різними алгоритмами для зручності тестування системи.

3. Завантаження та очищення даних

Для розробки системи інформаційного пошуку необхідні дані. У моєму випадку це текстова система пошуку, отже, для її створення потрібні текстові дані.

Існують різні методи отримання текстових даних з мережі - від скрапінгу форумів та агрегаторів новин до завантаження великих історичних архівів соціальних мереж через офіційні прикладні програмні інтерфейси (API). Twitter — соціальна мережа, яка є мережею мікроблогів, дає змогу користувачам надсилати короткі текстові повідомлення, використовуючи служби миттєвих повідомлень і сторонні програми-клієнти. Twitter є однією з найбільших існуючи соціальних мереж, а його концепція полягає в публікації користувачами коротких і в основному текстових (на відміну від Instagram, Facebook) повідомлень, які називаються твітами. Twitter часто стає площадкою для публікацій та обговорення політичних чи суспільно важливих думок, тому дані Twitter мають високу інформативність.

Першим кроком було створення облікового запису в Twitter. Я заповнив форми, описав проект і отримав доступ до стандартного облікового запису розробника для використання Twitter API. Після реєстрації та підтвердження проєкту в системі Twitter я отримав свої облікові дані, секретні ключі та токени (ACCESS_TOKEN, ACCESS_TOKEN_SECRET, API_KEY, API_SECRET_KEY та BEARER_TOKEN). Twitter пропонує безкоштовний REST API для потокової передачі даних у режимі реального часу та завантаження історичних даних частинами. Для користування цим API потрібно зробити та підтвердити в Twitter “обліковий запис розробника” та отримати облікові дані (секретні ключі і токени до Twitter API). [9]

Кожен модуль проєкту має власні завдання. Перший пакет, який опрацьовує вхідні дані це “main”. Він містить три класи:

- Main
- IdParser
- LocalSettings

LocalSettings містить лише змінні аутентифікації (токени доступу тощо). Я вирішив помістити їх в окремий файл, оскільки використовував систему контролю версій git для збереження проміжних версій моєї програми (щоб мати можливість відновити робочу версію у разі збою). Тому всі конфіденційні дані були перенесені в окремий клас Java, який був доданий до .gitignore.

IdParser містить деякі допоміжні статичні методи, що дозволяють працювати з файловою системою та проміжними даними. Також цей клас дозволяє регулювати кількість проміжних даних (взятих за рівномірним розподілом), які будуть передаватись на наступний етап конвеєру. Це необхідно для швидшого тестування алгоритмів використовуючи лише якусь частину від всіх даних.

Main

Спочатку нам потрібно авторизувати та налаштувати нашу програму через вбудований клас бібліотеки Twitter4j ConfigurationBuilder. Ось приклад моєї конфігурації:

```
ConfigurationBuilder cfg = new ConfigurationBuilder();
cfg.setTweetModeExtended(true);
cfg.setJSONStoreEnabled(true);

cfg.setOAuthAccessToken(LocalSettings.access_token);
cfg.setOAuthAccessTokenSecret(LocalSettings.access_token_secret);

cfg.setOAuthConsumerKey(LocalSettings.api_key);
cfg.setOAuthConsumerSecret(LocalSettings.api_secret_key);

Twitter twitter = new TwitterFactory(cfg.build()).getInstance();
```

Рис. 3.1 Авторизація програми для доступу до Twitter API

За допомогою Java бібліотеки Twitter4j та використовуючи облікові дані Twitter можна підключитись до Twitter API і почати завантаження даних.

Після цього я викликаю статичні методи з класу IdParser, щоб отримати ідентифікатори List<String> IDs. Потім, передаючи всі ці ідентифікатори функції twitter.lookup(), я отримую ResponseList<Status>, де

кожен елемент типу Status є окремим твітом, який містить велику кількість даних. Маючи всі ці дані, я перевіряю, чи мова твіту англійська. А потім я зберігаю всі вибрані твіти у форматі JSON (JavaScript Object Notation) за допомогою TwitterObjectFactory.getRawJSON(status).

JSON — це текстовий формат для обміну даними між комп'ютерами. JSON базується на інтуїтивно зрозумілій формі передачі тексту і може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу.

Оскільки потрібно завантажити досить великий набір даних, важливо налаштувати певні параметри під час ініціалізації. Треба встановити значення True для параметрів “wait_on_rate_limit” і “wait_on_rate_limit_notify”. Існують обмеження швидкості завантаження даних із Twitter — можна робити лише обмежену кількість запитів на завантаження даних до API за певний проміжок часу. Налаштувавши значення цих параметрів значенням True, з’єднання з API не порушиться, коли обмеження будуть досягнуті. Замість цього програма просто зачекає поки тайм-аут закінчиться, і продовжить завантаження даних.

Кожен новий твіт для зручності подальшого використання зберігається в окремому рядку і містить до кількох сотень різних полів з даними про нього. Приклад формування твіта:

```
{
    "created_at":"full_date_of_the_tweet",
    "id":"some_id",
    "id_str":"some_string_id",
    "full_text":"RT @nickname: some text...",
    "truncated":false/true,
    "display_text_range":[0,140],
    "entities":{
        "hashtags":[],
        "symbols":[],
```

```
            "user_mentions":[],
            "urls":[]
            },
        "source": "some_link"
    …
}
```


Дані про твіти користувачів отримані від Twitter API мають багато надлишкових метаданих, які не несуть корисної інформації для створення системи інформаційного пошуку. Тому JSON файли потрібно спершу очистити залишивши лише ті поля, які потрібні для подальшого використання, а саме: ID користувача, ім’я користувача, мова твіту, час створення, кількість ретвітів та текст твіту. Мною було розроблено клас Tweet з відповідними полями, для того щоб мати змогу зчитувати очищені дані JSON об’єктів в повноцінні об’єкти мови Java.

```
public Tweet(Status status){
    id = Long.toString(status.getId());
    user = status.getUser().getScreenName();
    language = status.getLang();
    createdAt = status.getCreatedAt().getTime();
    retweetCount = status.getRetweetCount();
    text = status.getText();
}
```

Рис. 3.2. Конструктор розробленого класу Tweet

Tweet — це базовий клас, який використовується для представлення сутності твіту в індексі. Таким чином ними можна буде легко оперувати та використовувати для формування індексу в подальшому. Оскільки нам не потрібні всі дані, що містяться в об’єкті Status (об’єкти такого типу завантажуються через Twitter API), я зберігаю в індексі лише ідентифікатор твіту, ім’я користувача, мову, дату створення, кількість ретвітів і текст твіту. Твіт містить конструктор, який формує його з об’єкта Status. (Рис. 3.2.)

4. Представлення та перетворення даних

4.1. Модель “торба слів”

Отримавши дані з Twitter та перевівши їх у формат об’єктів класу Tweet потрібно з них будувати систему інформаційного пошуку. Найціннішою частиною твітів, яка потім і буде використовуватись для створення індексу є текст твіту. Його в подальшому можна розглядати як **документ з інформацією**. Маючи документи з інформацією у будь-якому форматі (txt, html, pdf) потрібно забезпечити єдине, структуроване представлення документа (наприклад, вектор слів, граф, тощо). Найпоширенішою моделлю представлення є “торба слів” (bag-of-words). Ця модель найчастіше використовується в обробці природних мов і інформаційному пошуку, але є і різновиди для комп’ютерного зору та згорткових нейромереж — bag-of-features. У моделі “торба слів” текст (наприклад, речення або документ) представляється у вигляді торби (мультимножини) його слів, не беручи до уваги граматику і навіть порядок слів, але зберігаючи множинність. Наприклад, є речення: “Я люблю каву, але я не люблю чай”. Перевівши його в “торбу слів” отримаємо:

я : 2	любити : 2	кава : 1	але : 1	не : 1	чай : 1

Табл. 4.1

Але завдання переведення документів в модель “торба слів” не є простим і має певні етапи.

4.2. Токенізація

Під токенізацією розуміють процес “сегментації тексту, що є послідовністю символів, таких як букви, пробіли, знаки пунктуації та цифри, на слова і фрази”[6]. Маючи послідовність символів і визначену одиницю документа, токенізація — це завдання подрібнення його на частини, що називаються **токени**, можливо, водночас відкидаючи певні символи, розділові знаки, цифри, комплекси літер та цифр, інтернет-адреси, нікнейми, інші знаки (%, +, -, /), тощо.

Результатом токенізації речення “Я люблю каву” будуть 3 токени:

1. Я
2. Люблю
3. Каву

Тобто токен є відокремленою частиною послідовності символів. Але, наприклад, просто розділивши речення пробілами та розділовими знаками ми отримаємо поганий результат. Причинами цьому є різні формати відображення дат (наприклад, 5.11.2021 або 5/11/2021 або 5 лист. 2021), чисел (123456789 або 123 456 789), певні особливості конкретних мов (артиклі поряд зі словами, скорочення, різні напрямки письма) і тд.

4.3. Виключення стоп слів та фільтрація токенів

За допомогою списку стоп слів можна повністю виключити зі словника найпоширеніші слова. Зазвичай у них мало семантичного вмісту. Стоп словами є, наприклад, артиклі the, a(англ.), die, der, das(нім.), la, l(іт.) а також прийменники, частки і тд. Їх багато: ~30% всього тексту. Оскільки я при розробці системи інформаційного пошуку концентрувався на англійському сегменті твітів і формував систему інформаійного пошуку лише для англійської мови, то використовував вбудовану в Lucene колекцію англійський стоп слів.

```
@Override
protected TokenStreamComponents createComponents(String fieldName) {
    CharArraySet stopwordSet = new EnglishAnalyzer().getStopwordSet();
    UAX29URLEmailTokenizer src = new UAX29URLEmailTokenizer();
    TokenStream result = new LowerCaseFilter(src);
    result = new StopFilter(result, stopwordSet);
    result = new PorterStemFilter(result);
    result = new CapitalizationFilter(result);
    return new TokenStreamComponents(src, result);
}
```

Рис 4.3.1. Фільтри, які покращують токенізацію

Клас CustomAnalyzer розширює Analyzer. Я перевизначаю метод createComponents() базового класу аналізатора. CreateComponents() повертає

токени з вхідного рядка. Я використовую деякі фільтри, щоб покращити якість токенізації. (Рис. 4.3.1.)

4.4. Нормалізація, стемінг та лематизація

Наступним кроком, потрібно “нормалізувати” слова в тексті. Процес нормалізації однаково актуальний і для слів в запиті користувача до системи ІП. Завданням нормалізації є знайти відповідність між різними формами слова, формами відображення слова або, навіть, синонімами. Результат — нормалізована форма слова, для кожного слова. Ця нормалізована форма і буде окремим записом у системі ІП.

Нормалізація різних форм відображення слова:

Рис. 4.4.1. Приклад нормалізації

Також одними з основних методів нормалізації є стемінг та лематизація.

Стемінг — це процес скорочення слова до основи шляхом відкидання допоміжних частин, таких як закінчення чи суфікс. Результати стемінгу іноді дуже схожі на визначення кореня слова, але його алгоритми базуються на інших принципах. Тому слово після обробки алгоритмом стемінгу може відрізнятися від морфологічного кореня слова. Програми стеммінгу зазвичай називають стеммерами. Алгоритм стемінгу зводить слова “likes”, “liked”, “likely” та “liking” до кореневого слова “like”.(Рис. 4.4.2.)

Рис. 4.4.2. Приклад стеммінгу

Лемматизація – це процес об’єднання різних форм слів, щоб їх можна було проаналізувати як єдиний елемент. Лемматизація подібна до складання, але розглядає конкретний контекст до слів. Таким чином, вона пов’язує слова зі схожими значеннями з одним словом. Наприклад:

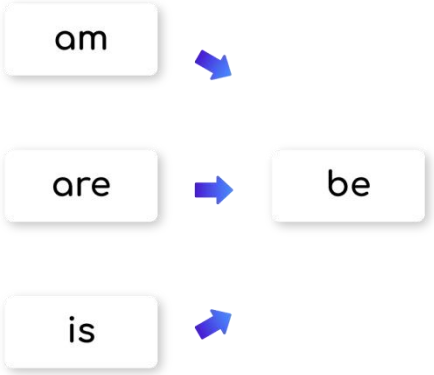


Рис. 4.4.3. Приклад лемматизації

Попередня обробка тексту включає як стеммінг, так і лемматизацію. Іноді до стемінгу та лематизації ставляться як до однакових методів. Насправді ж, лемматизація є складнішою ніж стемінг, оскільки лемматизація виконує морфологічний аналіз слів і розглядає контекст.

5. Створення індексу

5.1. Інвертований індекс

У своєму базовому варіанті створення індексу розглядається як створення матриці термінів документа (відношення “слово-документ”). **Матриця термінів документа** є матрицею інцидентності, яка описує частоту слів(термінів), які зустрічаються в колекції документів. Рядки відповідають документам у колекції, а стовпці — термінам. Іноді зустрічаються транспоновані матриці інцидентності, або відношення “слово-документ”, де документи є стовпцями, а терміни — рядками.

Такі матриці корисні в області обробки природної мови та обчислювального аналізу тексту. Хоча значення клітинок часто є просто булівським числом, яке показує присутність або відсутність конкретного слова, існують різні схеми для зважування значень, наприклад, нормалізація рядка і TF-IDF.

Але така форма для зберігання інформації не є найкращою. Розглянемо ситуацію при кількості документів N = 1 000 000. В кожному документі приблизно 1 000 слів. Нехай кількість різних слів буде 500 000. Тоді розмір матриці буде 500 000 * 1 000 000 = 500 000 000 000. Але серед цієї кількості нулів та одиниць буде не більше 1 000 000 000 одиниць (1 000 000 * 1 000). Отже, щонайбільше в матриці буде 1 одиниця на 500 нулів. Така матриця є дуже рідкою і займає значно більше місця в пам’яті ніж потрібно. Тому було розроблено нову структуру даних для більш ефективного використання пам’яті — інвертований індекс. Це ключова форма зберігання даних, що лежить в основі сучасного інформаційного пошуку.

Інвертований індекс — це структура даних, що зберігає відображення вмісту, такого як слова чи числа, до його розташування в документі або наборі документів. Простими словами, це структура даних, подібна до хеш-мапи, яка спрямовує вас від слова до документа. Існує два типи інвертованих індексів:

- Інвертований індекс на рівні запису містить список посилань на документи для кожного слова.
- Інвертований індекс на рівні слова додатково містить позиції кожного слова в документі.

Остання форма пропонує більше функціональних можливостей, але потребує більшої потужності обробки та простору для створення.

Процедура формування індексу починається з присвоєння всім документам серійного номеру (ID). Далі формується список всіх пар “термін - номер документу”. Цей список потрібно відсортувати і тоді з’єднати однакові терміни. Тоді для кожного слова зі словника слів створюється список (часто використовується структура linked list) з ID документів в яких це слово присутнє. Так формується **постінг-список.** Інвертований індекс, який складається зі словника слів і постінг-списку:

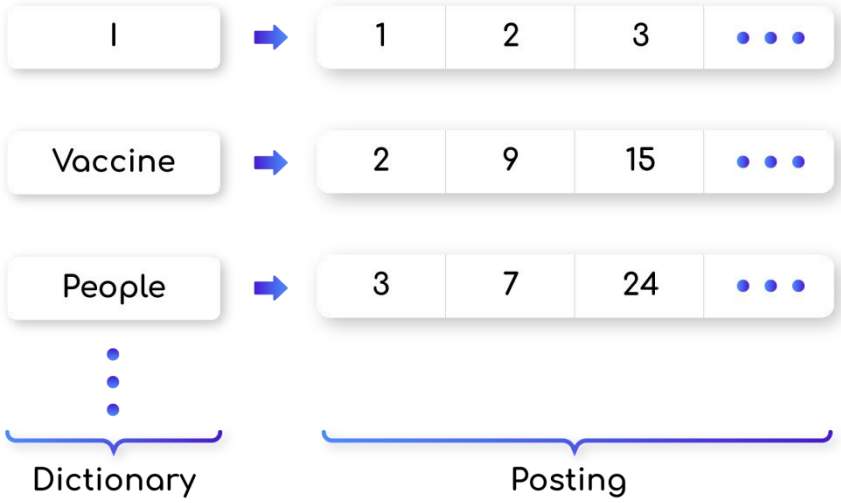


Рис. 5.1.1. Інвертований індекс

5.2. Розробка модулю для створення індексу

Клас **Indexer** містить деякі функції для створення індексу з архівних файлів, які містять об’єкти Status у форматі JSON.

Метод **getTweetsFromJSON()** приймає на вхід шлях до файлу як вхідні дані і зчитує з нього всі об'єкти Status. Кожен об’єкт Status передається конструктору об’єктів твітів. Функція повертає List<Tweet>: усі твіти, що містяться в цьому файлі.

Метод **getAllTweet()** проходить через усі файли даних і використовує функцію getTweetsFromJSON(). Потім він збирає всі твіти в один список і повертає його.

```
private static void addDoc(IndexWriter w, Tweet t) throws IOException {
    Document doc = new Document();

    doc.add(new StringField( name: "id", t.id, Field.Store.YES));
    doc.add(new StringField( name: "user", t.user, Field.Store.YES));
    doc.add(new StringField( name: "language", t.language, Field.Store.YES));
    doc.add(new LongPoint( name: "createdAt", t.createdAt));
    doc.add(new LongPoint( name: "retweetCount", t.retweetCount));
    doc.add(new TextField( name: "text", t.text, Field.Store.YES));

    w.addDocument(doc);
}
```

Рис. 5.2.1. Створення документу Lucene

Метод **addDoc()** приймає вхідними параметрами IndexWriter і Tweet. Ця функція створює документи Lucene з полями Lucene для кожного твіту. (Рис. 5.2.1.)

Функція **makeIndex()** спочатку налаштовує Writer, Analyzer і Directory. Потім використовуючи метод getAllTweets() отримує всі твіти та формує документи Lucene для кожного з них. Останнім кроком є фіксація всіх документів (w.commit()) і формування індексу. (Рис. 5.2.2.)

```
private static void makeIndex() throws Exception {
    StandardAnalyzer analyzer = new StandardAnalyzer();
    Directory dir = new SimpleFSDirectory(FileSystems.getDefault().
            getPath( first: GlobalProperties.PATH + "index"));
    IndexWriterConfig config = new IndexWriterConfig(analyzer);
    IndexWriter w = new IndexWriter(dir, config);
    List<Tweet> listOfAllTweets =
            getAllTweets( path: GlobalProperties.PATH + "tweets/");
    for(Tweet t : listOfAllTweets){
        addDoc(w, t);
    }
    w.commit();
    w.close();
}
```

Рис. 5.2.2. Запис і збереження документів в індексі

6. Контентний аналіз індексу

6.1. Методи ранжування результатів

Запит - це рядок тексту, що описує інформацію, яку шукає користувач. Кожне слово запиту називається пошуковим терміном (термом) або ключовим словом.

Запит може являти собою один пошуковий термін, рядок термінів, фразу природною мовою або стилізований вираз з використанням спеціальних символів.

Використовуючи різні підходи, запит можна розширити, адаптувати і перетворити перед виконанням зіставлення з термінами, що зберігаються в інвертованих індексах. Потім, терміни запиту використовуються для пошуку відповідних документів.

Ці відповідні документи, також відомі як результати пошуку, сортуються відповідно до того, наскільки близько вони прогнозують відповідність вхідному запиту.

Коли користувач надсилає запит до пошукової системи, система повертає набір результатів. З великою колекцією документів набір результатів може бути дуже великий. Цінність для користувача залежить від порядку, в якому представлені результатів. Завдання сортування результатів називається також ранжуванням.

Три основні методи:

- Сортування результатів за датою;
- Ранжування результатів за схожістю між запитом і документом;
- Ранжування результатів за важливістю документів;

Функція ранжування має фундаментальний вплив на релевантність результатів пошуку, тому правильна робота з нею означає, що пошукова система матиме більш високу точність, і користувачі будуть отримувати найрелевантнішу та важливу інформацію насамперед.

На практиці функція ранжування часто є частиною моделі ІП. Така модель визначає, як пошукова система вирішує проблему надання

релевантних результатів щодо інформаційної потреби: від аналізу запиту до зіставлення, пошуку та ранжування результатів пошуку. Обґрунтування наявності функції ранжування в моделі полягає в тому, що важко вигадати функцію, яка дає точну оцінку, не знаючи, як система ІП обробляє запит.

Таким чином, спосіб, яким система ІП обчислює релевантність документа щодо запиту, впливає на дизайн цієї системи. Коли текст подається в систему, він аналізується і розбивається на шматки, які можна змінювати залежно від токенізаторів та фільтрів токенів. Цей ланцюг аналізу тексту генерує терміни, які потрапляють в інвертовані індекси, також відомі як постінг-списки. Варіант використання “пошук за ключовим словом” мотивував вибір постінг-списків для ефективного пошуку документів шляхом зіставлення термінів. Так само вибір порядку ранжування пар типу запит-документ може вплинути на вимоги до дизайну системи. Наприклад, функції ранжування може знадобитися доступ до додаткової інформації про індексовані дані, а не просто наявність або відсутність терму в постінг-списку.

6.2. TF-IDF

Основна мета функції ранжування – присвоїти оцінку парі “запит-документ”. Поширений спосіб вимірювання важливості документа щодо запиту заснований на обчисленні та отриманні статистики для слів запиту та документа. Такі пошукові моделі називаються статистичними моделями для пошуку інформації. Широко використовуваний набір таких пошукових моделей приймає рішення про ранжування певного документа, ґрунтуючись на тому, як часто відповідний термін з'являється в конкретному документі та в усьому наборі документів.

Часто статистичні моделі поєднують у собі частоту слова (**term frequency - TF**) та обернену частоту документа (**inverse document frequency - IDF**) щоб визначити міру релевантності документа з урахуванням запиту. Обгрунтування вибору цих метрик полягає в тому, що обчислення цих характеристик за термінами дає вам міру того, наскільки інформативним є

кожен з них. Тобто, кількість появ терміну запиту в документі, дає міру того, наскільки доречний цей документ для цього запитую. Це – term frequency.

$$\mathrm{tf}(t,d) = \frac{\text{count of t in d}}{\text{number of words in d}} \qquad (6.2.1)$$

де t - term (слово), d - document (набір слів).

Це значення дуже залежить від довжини документа. Для мого дослідження у Term Frequency я беру часове вікно (time window) як документ. Тобто, якщо вікно було 10 днів, кількість токенів t в документі d — це кількість токенів серед усіх твітів, які були опубліковані протягом цих 10 днів.

Але кількість різних токенів у різних часових вікнах різна. Аналогічно, загальна кількість слів у всіх твітах, опублікованих протягом певного періоду часу, відрізняється. І не можна сказати, що довший документ важливіший за коротший. Саме з цієї причини ми виконуємо нормалізацію частоти. Частоту ділимо на загальну кількість слів у документі.

У найгіршому випадку, якщо слово не існує в документі, тоді це конкретне значення TF буде 0, а якщо всі слова в документі збігаються, то значення буде 1. Остаточне значення нормалізованого значення TF завжди буде в діапазоні [0..1].

З іншого боку, слова, які рідко з'являються в індексованих даних, можна вважати важливішими та інформативнішими, ніж більш поширені слова. Кількість документів в яких зустрічається термін запиту серед всіх проіндексованих документів називається частотою документа (**document frequency**). Частота документа вимірює важливість токена у всьому наборі документів. Єдина відмінність від TF полягає в тому, що DF – це кількість зустрічей терміна t у наборі документів N. Іншими словами, DF – це кількість документів, у яких присутнє це слово. Щоб зберегти це значення в діапазоні [0..1], ми нормалізуємо його, розділивши на загальну кількість документів.

IDF – обернена частота документа, яка вимірює інформативність слова t.

$$\mathrm{idf}(t) = \frac{N}{\mathrm{df}(t)} \qquad (6.2.2)$$

де t - слово, df - частота документа, N - кількість всіх документів в колекції.

Використання IDF зменшує вагу широковживаних слів. Коли ми обчислюємо IDF, значення буде дуже низьким для слів, які найбільш вживані. Тому що ці слова присутні майже у всіх документах, і IDF надасть цьому слову дуже низьке значення.

Більш підходящий спосіб виразити ранжування - це оцінити кожен документ, використовуючи логарифм TF (6.2.1), помножений на логарифм оберненої частоти документа IDF (6.2.2). Ця функція зважування називається TF-IDF:

$$w(t) = (1 + \log_{10} \frac{n_t}{\sum_k n_k}) * (\log_{10}(\frac{N}{N_t})) \qquad (6.2.3)$$

де w(t) – вага слова,

k – кількість різних слів в документі,

n_k – кількість k-тих слів в документі,

n_t – кількість зустрічань слова t в документі,

N – загальна кількість проіндексованих документів,

N_t – кількість документів зі словом t.

Я намагався використовувати різні концепції для визначення “документа” у формулі IDF, але не всі нормалізували дані правильно. Наприклад, я спочатку спробував взяти в якості документа в IDF те саме часове вікно, що й у TF, але результат був поганий, тому що такий документ був дуже великим, швидше за все містив усі токени і тому нормалізував значення IDF неякісно. Потім я спробував менші вікна (1/4 дня). Це покращило нормалізацію, але це все ще був занадто великий документ, і найчастіше вживані слова як рідковживані слова були знайдені у майже всіх документах.

Тоді я вирішив взяти за концепцію документа для IDF окремий твіт. Отже, DF у цьому випадку був просто відсотком усіх твітів, які використовували конкретне слово.

Знайшовши TF-IDF для всіх слів, я очистив дані від деяких непотрібних, але частих токенів для аналізу. Вони не були з набору стоп-слів і тому пройшли фільтри під час токенізації. Наприклад, “Rt” був найбільш використовуваним маркером, але це лише маркер, який позначає, що даний твіт був ретвітнутий. Це слово створюється соціальною мережею Twitter автоматично і тому не несе жодного інформаційного навантаження про те, які слова люди використовують частіше.

Далі я відсортував дані за значенням TF-IDF і зберіг їх у кількох різних форматах файлів, щоб я міг прочитати їх із файлів і використовувати для наступних алгоритмів. Ось приклад частини файлу, яка показує “найважливіші” слова та їхні TF-IDF:

```
Vaccin 0.1343155502250887
Covid 0.034106663149738925
I 0.023420318934934483
Peopl 0.02263603579706883
You 0.020452085486584008
Have 0.020440116888412723
Get 0.01842219689705301
We 0.018245060349230402
19 0.01647848283781966
Receiv 0.013912816336098123
Ha 0.012942256286596423
Dai 0.012002877715527771
Dose 0.011789152886820918
From 0.011634784683136085
My 0.01110778177859726
Sai 0.01043726777994017
Just 0.010111267352538852
So 0.009529465389966863
About 0.009478864059066158
New 0.009095589372108068
All 0.009071634629183383
Coronaviru 0.008953515529311477
Who 0.008830494412925069
Out 0.008549422673229496
More 0.00851930870232568
Now 0.008436835741434537
First 0.008432044792849598
Us 0.00836132186361833
Up 0.008260785111432741
```

Рис. 6.2.1. “Найважливіші” слова та їхні значення TF-IDF

6.3. Часові ряди

Часовий ряд — це послідовність точок даних, які зазвичай вимірюються в послідовні моменти часу, розташованих через однакові проміжки часу.[11]

Часовий ряд токену — це часовий ряд, який відстежує кількість повторень одного токену за певний час.

Я використовував 24-годинні(добові) інтервали, щоб побудувати часовий ряд для кожного токена. Далі, мною було розглянуто часове вікно розміром 10 днів, починаючи з 04.01.2021 по 14.01.2021. Для кожного токена я рахував кількість згадок цього слова щодня. Ці дані були повернуті списком з 10 значень для кожного токена, а потім були використані для створення рядка SAX для кожного токена.

6.4. Розробка модулю для контентного аналізу та допоміжних функцій

Клас **Searcher** містить такі функції:

1) **makeTokens()** - приймає як вхідний параметр об’єкт типу String (наприклад, дані з індексованого поля “text” твіту) та аналізатор. Цей статичний метод виконує токенізацію тексту і в результаті повертає List<String> токенів. Ця функція використовується для побудови TF-IDF.

```
public static HashMap<String, Double> TF(int time_shift) throws Exception {
    HashMap<String, Integer> frequencies = new HashMap<~>();
    Directory dir = new SimpleFSDirectory(FileSystems.getDefault().
            getPath( first: GlobalProperties.PATH + "index"));

    Query q = LongPoint.newRangeQuery( field: "createdAt",
            lowerValue: GlobalProperties.START +  GlobalProperties.DAY * time_shift,
            upperValue: GlobalProperties.START + GlobalProperties.DAY *
                    (GlobalProperties.TIME_WINDOW_SIZE + time_shift));

    IndexReader reader = DirectoryReader.open(dir);
    IndexSearcher searcher = new IndexSearcher(reader);

    TopDocs docs = searcher.search(q, GlobalProperties.INF);
    ScoreDoc[] hits = docs.scoreDocs;
```

Рис. 6.4.1. Частина функції TF()

2) **TF()** приймає як вхідний параметр зсув у часі. Це дозволяє нам послідовно застосовувати його до всіх часових проміжків (вікон). Функція підраховує і повертає значення TF для всіх токенів, які зустрічаються в інтервалі часу. (Рис. 6.4.1.)
3) **IDF()** приймає токен як вхідні дані. Надсилає запит до індексу для цього токена та знаходить кількість зустрічань (кількість твітів, у яких цей токен згадується за весь час – 40 днів, з січня по лютий 2021 року). Функція повертає значення idf для даного токену.
4) **getTimeSeries()** приймає токен і зміщення часу як вхідні дані. Функція повертає масив із 10 елементів, які відповідають появі певного токена в кожен з 10 днів часового проміжку.
5) **round ()** — додаткова функція, яка дозволяє заокруглювати значення типу Double.

```
public static ArrayList<Integer> getTimeSeries(String token, int time_shift) throws IOException {
    Directory dir = new SimpleFSDirectory(FileSystems.getDefault().getPath( first: GlobalProperties.PATH + "index"));
    IndexReader reader = DirectoryReader.open(dir);
    IndexSearcher searcher = new IndexSearcher(reader);
    TotalHitCountCollector collector = new TotalHitCountCollector();
    ArrayList<Integer> occurrences = new ArrayList<~>();

    for (int i = 0; i < 10; i++){
        Query q = new TermQuery(new Term( fld: "text", token));
        Query q2 = LongPoint.newRangeQuery( field: "createdAt",
                lowerValue: GlobalProperties.START +  GlobalProperties.DAY * (i + time_shift),
                upperValue: GlobalProperties.START + GlobalProperties.DAY * (i + 1 + time_shift));
        BooleanQuery.Builder bqb = new BooleanQuery.Builder();
        bqb.add(q, BooleanClause.Occur.MUST);
        bqb.add(q2, BooleanClause.Occur.MUST);
        TopDocs hits = searcher.search(bqb.build(), Math.max(1, collector.getTotalHits()));
        int num_hits = (int) hits.totalHits.value;
        occurrences.add(num_hits);
    }

    return occurrences;
}
```

Рис. 6.4.2. Метод getTimeSeries()

6) **convertIntegers()** дозволяє перетворити часовий ряд у масив подвійних значень, які потім використовуються в реалізації алгоритму SAX (див. Розділ 6.5.)
7) Метод **makeQuery()** використовується для виконання потрібних запитів до індексу.

Функція **buildTimeSeries()** з класу Indexer приймає вхідним параметром HashMap токенів і повертає HashMap<String, ArrayList<Integer>>, де ArrayList<Integer> — це масив довжиною 10, де кожен елемент позначає скільки разів певний токен зустрічається щодня у визначеному часовому вікні. Також в цьому модулі містяться деякі допоміжні класи.

Клас **FileManager** розроблений для роботи з файлами та запису різних структур даних в власному форматі. Клас містить методи createFile(), readHashMapFromFile(), writeToFileSDMap() та writeToFileSSMap().

Клас **GlobalProperties** містить глобальні змінні для швидкої та зрозумілої взаємодії з ними. Наприклад 86400000 - це кількість мілісекунд в 1 добі. Ці змінні необхідні для операцій з часовими вікнами. (Рис. 6.4.3.)

```
public static final long DAY = 86400000L;
public static final long START = 1609718400000L;
public static final long END = 1613174400000L;
public static final int INF = Integer.MAX_VALUE;
public static final int TOP_VALUES_SIZE = 5000;
public static final int TIME_WINDOW_SIZE = 10;
```

Рис. 6.4.3. Глобальні змінні необхідні для операцій з часовими проміжками.

Клас MapSort містить деякі корисні методи і компаратори для сортування різних структур HashMap.

6.5. Symbolic Aggregate Approximation

SAX (Symbolic Aggregate Approximation) — це символічне представлення часових рядів, яке дозволяє зменшувати розмірність та записувати часовий ряд використовуючи нижню межу Евклідової відстані.[12]

SAX перетворює часовий ряд X довжини n в рядок довільної довжини m, де m < n зазвичай, використовуючи алфавіт A розміру |A| >= 2.

Алгоритм складається з двох кроків:

1) Перетворює вихідний часовий ряд у представлення **PAA** (Piecewise Aggregate Approximation).[13]

2) Перетворює дані PAA в рядок символів.

Символ вираховується із середнього значення конкретного сегмента. SAX ігнорує деяку інформацію в сегменті, а саме тенденцію(тренд) зміни значення в сегменті. Це може спричинити дещо неправильну класифікацію в деяких випадках(коли потрібна дуже висока точність), оскільки подання SAX не може розрізнити різні часові ряди з подібними середніми значеннями, але різними трендами. Для мого дослідження не потрібно знати детальні зміни тренду тому алгоритм SAX добре справився зі завданням.

```
public static String makeSAX(double[] ts) throws SAXException {
    int alphabetSize = 3;
    double nThreshold = 0.01;
    NormalAlphabet na = new NormalAlphabet();
    SAXProcessor sp = new SAXProcessor();
    SAXRecords res = sp.ts2saxByChunking(ts, ts.length,
            na.getCuts(alphabetSize), nThreshold);
    String sax = res.getSAXString( separatorToken: "");
    return sax;
}
```

Рис. 6.5.1. Метод для представлення часових рядів у формі SAX

Метод **makeSAX()** класу Searcher приймає Double масив як вхідні дані. (Рис. 6.5.1.) Повертає рядок SAX, сформований із часового ряду певного токену. Алфавіт для побудови рядка SAX містить три символи: a, b, c.

```
Fold acacaaaaaa
Keir bccaaaabab
Elig bbcbbbbbbb
Rat cccaabaacb
Kanyou95 bbbbbbbbbc
Diff abccaacabb
Folk bbbbbcaacb
Raw aaabbbbcaa
Freedom acabaabbcb
Hurt cbbabaacab
Decent bbbbccaaaa
Western acacaaabbc
Grow acbbcbccaa
Asanilta bbbbbbbbbc
Sure bccabaabba
Probabl caaacaaaaa
Ramp ccbbaacaba
Michael abbbccacba
Uneven aaaaaccaaa
Vein acaaaacaaa
Weekend aabccbacaa
Small bbaabaaccc
Univers bbbbbbbbbb
```

Рис. 6.5.2. Результат представлення часових рядів деяких токенів у формі SAX

Для імплементації методу переведення часових рядів у форму SAX я використовував алфавіт розміру |A|=3, тобто він складався з букв a b і c. Щоб продовжити використання вже підготовлених даних, результати SAX були збережені в різних форматах файлів для зручності використання на інших кроках аналізу в майбутньому. Приклад представлення часового ряду для деяких випадково вибраних токенів подано на рис. 6.5.2.

Можна помітити що для деяких токенів рядок символів SAX є майже однорідним, а для деяких він часто міняється. Це означає, що деякі слова на вибраному часовому проміжку зустрічались приблизно з однаковою частотою кожного дня (випадок коли літери не міняються часто, наприклад, bbbbbbbbbb), а деякі, навпаки, в певні дні вживались частіше, а в інші частота вживання цих слів користувачами Twitter згасала (наприклад, Covid19 bccacaaacb). Це може бути спричинене певними зовнішніми подіями, які стосувались тих чи інших слів та їхньої популярності.

7. Висновки

В рамках магістерської роботи я дослідив основні методи створення систем інформаційного пошуку та деякі методи контентного аналізу соціальних мереж. Система була розроблена на основі даних соціальної мережі Twitter. Дані завантажувались у форматі JSON. Для побудови індексу було використано близько 1 мільйону твітів (січень – лютий 2021 року). Рис. 7.1. графічно ілюструє етапи процесу розробки системи ІП.

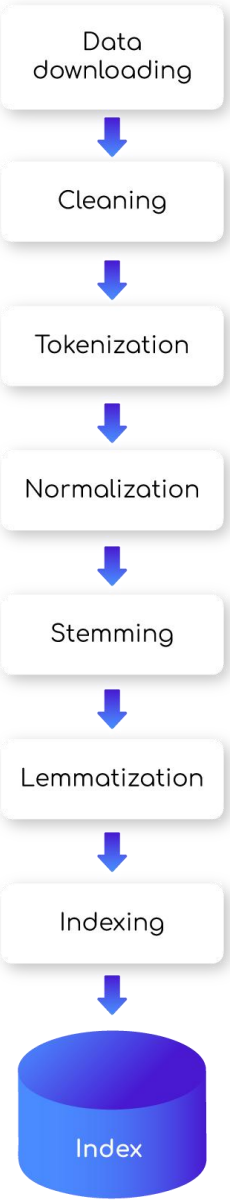


Рис. 7.1. Етапи розробки системи ІП

Я дослідив та описав процес розробки методів обробки тексту, які я використовував для підготовки даних до процесу індексації: токенізацію, нормалізацію, стеммінг та лемматизацію. Описав їхні переваги та недоліки і

показав результат почергового їх застосування до вхідних даних. Розглянув всі етапи розробки системи ІП починаючи зі скачування та очищення даних до розробки інвертованого індексу та створив програмні модулі для виконання цих завдань. Описав та порівняв старі (матриця інцидентності “слово-документ”) та нові (постінг-список) підходи до створення індексу.

Пояснив як з допомогою методу ранжування результатів пошуку можна проводити контентний аналіз соціальної мережі. Запропонував та описав модель контентного аналізу з допомогою ранжування результатів. Для цього була застосована статистична метрика TF-IDF.

Показав отримані в такий спосіб результати. Одними з найпопулярніших слів були covid, vaccine та coronavirus. Ці результати контентного аналізу можна вважати репрезентативними та об’єктивними через великий розмір вхідних даних. Враховуючи пандемічну ситуацію у світі, ці результати були очікуваними та прогнозованими.

Далі було описано та імплементовано концепцію часових рядів для токенів. Було обрано часовий проміжок та пораховано частоту вживання для кожного токену щоденно. Часові ряди були переведені в форму SAX. Отримані результати представлені графічно.

Майбутніми напрямками для дослідження є проведення додаткового аналізу даних соціальної мережі використовуючи алгоритми на графах, розробка клієнт-серверного застосунку та обробка даних в режимі “реального часу”.

Ціль магістерської роботи було досягнуто. Мною було створено оптимізовану систему інформаційного пошуку з допомогою мови програмування Java та проведено базовий контентний аналіз соціальної мережі з допомогою ранжування результатів пошуку в інвертованому індексі та обчислення часових рядів частоти вживання кожного слова щоденно.

8. Список використаних джерел

1. Singhal A. Modern Information Retrieval: A Brief Overview // Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. – 2001 – 24 (4): 35–43 – [Electronic resourse]–Available at: http://singhal.info/ieee2001.pdf

2. The Text Retrieval Conference (TREC); organized by National Institute of Standards and Technology (NIST), USA – 1992 – [Electronic resourse]– Available at: https://trec.nist.gov/

3. Jansen B. The Seventeen Theoretical Constructs of Information Searching and Information Retrieval / Jansen, B. Rieh, S. // Journal of the American Society for Information Sciences and Technology–2010–61(8), 1517-1534–[El. resourse]– Available at: https://faculty.ist.psu.edu/jjansen/academic/jansen_theoretical_constructs.pdf

4. Manning C. CS276/LING286: Information Retrieval and Web Search / Manning C., Nayak P. // Stanford cource – 2019 – [Electronic resourse] – Available from: https://web.stanford.edu/class/cs276/

5. Frakes, William B.; Baeza-Yates, Ricardo. ”Information Retrieval Data Structures & Algorithms”. Publisher: Pearson; Facsimile edition – June 12, 1992 – 512 p.

6. Teodorescu M. Machine Learning Methods for Strategy Research / Teodorescu M.H. // HBS Working Paper 18-011. – Harvard Business School – 2017 – 59 p.

7. Teofili T. Deep Learning for Search / T. Teofili // Manning Publications Co – 2020 – 317p.

8. Skorupskyy O. Information retrieval system development / Skorupskyy O., Muzychuk Y. // International student scientific conference on applied mathematics and computer science (ISSCAMCS – 2022), Lviv – 5-6 May, 2022 – 4-7 p. – [Electronic resourse]. – Available from: https://ami.lnu.edu.ua/wp-content/uploads/2022/05/ISSCAMCS-2022.pdf

9. Twitter Inc. Twitter Api documentation – Cited on 05.11.2021– [Electronic resourse] – Available at: https://developer.twitter.com/en/docs/twitter-api

10. Manning C. An Introduction to Information Retrieval / Manning C., Raghavan P., Schütze H. // Online edition, Cambridge UP – 2009 – 544 p.
11. Ahmed S. Transformers in Time-series Analysis: A Tutorial / S. Ahmed , I. Nielsen, A. Tripathi, S.Siddiqui, G.Rasool, R.Ramachandran // Rowan University – 28 April 2022 – [Electronic resourse]. – Available from: https://arxiv.org/pdf/2205.01138.pdf
12. Lkhagva B. Extended SAX: Extension of Symbolic Aggregate Approximation for Financial Time Series Data Representation / B. Lkhagva, Y. Suzuki, K. Kawagoe // Nagoya University, Ritsumeikan University – January 2006 – [Electronic resourse]. – Available from: https://www.researchgate.net/profile/Yu-Suzuki-2/publication/229046404_Extended_SAX_extension_of_symbolic_aggregate_approximation_for_financial_time_series_data_representation/links/570b819d08ae8883a1ffa123/Extended-SAX-extension-of-symbolic-aggregate-approximation-for-financial-time-series-data-representation.pdf
13. Yu Y. A novel trend Symbolic Aggregate Approximation for Time Series / Yu Y., Y. Zhu, D. Wan,Q. Zhao, H. Liu // College of Computer and Information, Hohai University, China. Computer Science and Engineering, Arizona State University,U.S.A – [Electronic resourse]. – Available from: https://arxiv.org/ftp/arxiv/papers/1905/1905.00421.pdf