Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи і моделі первинної обробки інформації з елементами самоподібності

(тема)

Виконав:
студент II курсу, групи КСМм-19-1
Демяненко А.Є.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерні системи та мережі
(повна назва освітньої програми)

Керівник: проф. Завизіступ Ю.Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Демяненку Андрію Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи і моделі первинної обробки інформації з елементами самоподібності

затверджена наказом по університету від " 30 " жовтня 2020 р. № 1487 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14 грудня 2020 р.

3. Вхідні дані до роботи
Документація стандарту мови програмування JavaScript ECMAScript
Документація HTML, CSS
Документація мови програмування Scala
Документація фреймворку Apache Spark
Документація бібліотеки Akka

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області(первинна обробка інформації з елементами самоподібності)
Аналіз існуючих методів обробки даних
Інструментарій для реалізації

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд презентація – 15 слайдів.

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд стану проблеми та постановка задачі	03.11- 11.11	
2	Підбір літератури за напрямком магістерської	11.11 – 13.11	
	роботи		
3	Вибір методів рішення для реалізації та їх	14.11 – 30.10	
	обґрунтування		
4	Розробка додатку за поставленою задачею	16.11 – 28.11	
5	Оформлення пояснювальної записки	22.11 – 01.12	
6	Оформлення графічної частини	01.12 – 02.12	

Дата видачі завдання 02 листопада 2020р.

Студент _____
(підпис)

Керівник роботи _____ проф. Завизіступ Ю.Ю.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 78 с., 30 рис., 2 табл., 1 дод., 20 джерел.

КЛІЄНТ, СЕРВЕР, WEB, ДАНІ, ІНФОРМАЦІЯ, САМОПОДІНІСТЬ, ІНТЕРНЕТ, ПРОТОКОЛ, СЕРВЕР, SCALA, APACHE, SPARK, КЛАСТЕРИЗАЦІЯ.

Дана атестаційна робота присвячена створенню методів і моделей для первинної обробки інформації на основі існуючих алгоритмів та методів обробки даних.

Мета даної роботи полягає у розробці модифікованого методу обробки інформації на основі існуючих методів та алгоритмів кластеризації даних. Даний метод буде протестований на даних у текстовому форматі завдяки розробленому веб-додатку з використанням новітніх технологій веб-розробки.

ABSTRACT

Master’s thesis: 78 pages, 30 figures, 2 tables, 1 appendices, 20 sources.

CLIENT, SERVER, WEB, DATA, INFORMATION, САМОПОДІНІСТЬ, ІНТЕРНЕТ, ПРОТОКОЛ, СЕРВЕР, SCALA, APACHE, SPARK, CLUSTERING.

This thesis is devoted to the creation of methods and models for primary information processing based on existing algorithms and data processing methods.

The purpose of this work is to develop a modified method of information processing. This method will be tested on data in text format thanks to the developed web application using the latest web development technologies.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 10
1 АНАЛІЗ МОЖЛИВОГО РІШЕННЯ ПРОБЛЕМИ 11
1.1 Аналіз проблеми обробки даних 11
1.2 Визначення та роль Big Data 12
1.3 Опис видів зберігання текстової інформації 14
1.3.1 Опис формату даних CSV 14
1.3.2 Опис формату даних XML 15
1.3.3 Опис формату даних JSON 15
1.4 Властивості інформації з елементами самоподібності 15
2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ 19
2.1 Опис принципів роботи алгоритмів кластеризації даних 19
2.2 Метод кластеризації k-means 21
2.2.1 Етапи виконання алгоритму k-means 22
2.2.2 Переваги методу k-means 23
2.2.3 Недоліки методу k-means 24
2.3 Метод кластеризації з використанням моделей Гауса 26
3 АЛГОРИТМ РОБОТИ МОДИФІКОВАНОГО МЕТОДУ КЛАСТЕРИЗАЦІЇ 29
3.1 Основні положення та умовних позначень 29
3.2 Визначення поняття вбудованого та внутрішнього вимірів 30
3.3 Визначення фрактального набору даних та його фрактальної розмірності 31
3.4 Опис алгоритму підрахунку фрактальної розмірності 32
3.5 Опис алгоритму вибору атрибутів 33
3.6 Основні положення кластеризації k-means з розподіленням 34

3.7 Переваги методу k-means з розподіленням 37
4 РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ПЕРВИННОЇ ОБРОБКИ ІНФОРМАЦІЇ 39
4.1 Використовувані технологій розробки інтерфейсу для користувача 39
4.1.1 Аналіз мови програмування JavaScript 39
4.1.2 Опис роботи движкаV8 для JavaScript 41
4.1.3 Бібліотека для створення графічного інтерфейсу 44
4.2 Використовувані технології розробки веб-серверу 47
4.2.1 Вибір мови програмування серверної частини 47
4.2.2 Scala у роботі з «великими даними» 50
4.2.3 Опис фреймворку Apache Spark 53
4.2.4 Опис інструментів для веб-розробки Akka 56
4.3 Реалізація методу первинної обробки інформації 60
4.3.1 Аналіз інформації для обробки 60
4.3.2 Опис роботи інтерфейсу користувача 60
4.3.3 Опис обробки даних на сервері 62
ВИСНОВКИ 67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 68
ДОДАТОК А Графічний матеріал атестаційної роботи 70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

ВНЗ – вищий навчальний заклад

ДСТУ – Державний стандарт України

ЕОМ – електронна обчислювальна машина

КР – курсова робота

КП – курсовий проект

НП – навчальний план

РГР – розрахунково-графічна робота

СРС – самостійна робота студентів

СУБД – система управління базами даних

Таблиці сутності – основні таблиці, в яких міститься інформація, що динамічно змінюється

AJAX – підхід до побудови користувацьких веб інтерфейсів (англ., Asynchronious JavaScript and XML)

Akka – інструментарій з відкритим вихідним кодом і середовище виконання, що спрощує створення паралельних і розподілених додатків на JVM

CSS – спеціальна мова, що використовується для опису зовнішнього (англ., Cascading Style Sheets)

CSV – текстовий формат, призначений для представлення табличних даних, які розділяються через кому (англ., Comma-Separated Values)

HTML – мова розмітки гіпертекстових документів, стандартна мова розмітки для створення веб-сторінок і веб-додатків (англ., Hypertext Markup Language)

JSON – простий формат обміну даними, зручний для читання і написання як людиною, так і комп'ютером (англ., JavaScript Object Notation)

UI – користувацький інтерфейс вигляду сторінок, написаних мовами розмітки даних (англ., User Interface)

Spark – фреймворк з відкритим вихідним кодом для реалізації розподіленої обробки неструктурованих і слабоструктурованих даних

XML – розширювана мова розмітки (англ., eXtensible Markup Language)

ВСТУП

Збільшення популярності мережі інтернет з кожним роком призводить до збільшення обсягу інформації, якими оперують користувачі по всьому світу кожен день. Дослідження різних наукових співтовариств показують, що до 2013 року дев'яносто відсотків інформації по всьому світу було створено протягом двох останніх років. Також прогнозується, що до кінця 2020 року об`єм створеної інформації досягне значення в 44 Збайт.

Таким чином, зі збільшенням переданої кількості інформації в мережі інтернет стали з'являтися різні методи і моделі з обробки інформації. На даний момент існує багато рішень і алгоритмів по обробці даних. Тому вибір і використання різних статистичних методів для аналізу і добування інформації з даних стали основними завданнями щодо організації інформації в згруповані і відсортовані кластери даних.

Кластерний аналіз або кластеризація даних – це завдання угруповання набору об'єктів таким чином, щоб об'єкти в одній групі (званої кластером) були більш схожі один на одного, ніж на об'єкти в інших групах (кластерах). Це основне завдання дослідницького інтелектуального аналізу даних і загальний метод статистичного аналізу даних, який використовується в багатьох областях, включаючи розпізнавання образів, аналіз зображень, пошук інформації, біоінформатику, стиснення даних, комп'ютерну графіку і машинне навчання. Приведення інформації в такий вид сприяє спрощенню роботи з нею і мінімізації витрат часу на її програмну обробку.

Однак при обробці вхідного потоку інформації можуть виникати деякі її особливі властивості. Наприклад, однією з таких властивостей є самоподібність інформації. Зберігання інформації в необробленому вигляді може бути скрутним при подальшій обробці і аналізу даних. Тому використовуючи дану властивість можна розробити алгоритм, який значно спростить зберігання та аналіз інформації.

1 АНАЛІЗ МОЖЛИВОГО РІШЕННЯ ПРОБЛЕМИ

1.1 Аналіз проблеми обробки даних

На даний момент обсяг усієї створеної інформації сягає близько 44 Збайт і він зростає у геометричній прогресії кожного дня. Тому різні вчені та спеціалісти в області обробки та зберігання даних почали створювати нові алгоритми, методи і моделі для роботи з різним типом інформації. Наприклад, обробка великих масивів зображень, обробка відео та аудіо даних тощо.

Основними можливостями таких алгоритмів є:

- обробляти великі в порівнянні з «стандартними» сценаріями обсяги даних;

- вміти працювати з швидко надходять даними в дуже великих обсягах. Тобто даних не просто багато, а їх постійно стає все більше і більше;

- вміти працювати зі структурованими і слабо структурованими даними паралельно і в різних аспектах ».

Вважається, що ці можливості дозволяють виявити приховані закономірності, що вислизають від обмеженого людського сприйняття. Це дає безпрецедентні можливості оптимізації багатьох сфер нашого життя: державного управління, медицини, телекомунікацій, фінансів, транспорту, виробництва і так далі.

Таким чином одною із таких можливостей є самоподібність інформації, якій приділяють не так багато уваги поза межами обробки графічного типу даних. Самоподібність та фрактали – це поняття, запроваджені Бенуа Б. Мандельбротом [2]. Вони описують явище, коли певна властивість об’єкта – наприклад, природний образ, збіжний піддомен певних динамічних систем, часовий ряд або відношення однієї частини структури даних до іншої – зберігається щодо масштабування в просторі і часі.

Тому проблема виявлення самоподібності в інших форматах даних все ще залишається актуальною. Наприклад формати даних JSON, XML та CSV. У даній роботі буде представлене можливе рішення даної проблеми за допомогою алгоритмів кластеризації, сучасних інструментів веб-розробки та методів Big Data.

1.2 Визначення та роль Big Data

Беручи до уваги усі вищезазначені тези можна зробити висновок, що одним з можливих рішень проблеми самоподібності інформації є первинна обробка інформації з використанням визначених методів і моделей для обробки інформації. Для реалізації даного рішення був обраний модифікований алгоритм з кластеризації даних, реалізований у середовищи веб-серверу.

Сервер даного веб-додатку, на якому будуть проводитися усі обчислення, повинен мати здібність виконувати багато операцій з аналізу та обробки великої кількості інформації (Big Data).

Великі дані (Big Data) – визначення структурованих та неструктурованих даних великих обсягів та значущої багатообразованості, ефективно обробляються горизонтально масштабованими програмними інструментами.

Великі дані – це сукупність технологій, які призвані здійснити три операції:

- обробляти більші у порівнянні зі стандартними сценаріями обсягів даних;
- працювати із даними, які надходять швидко у дуже великих обсягах;
- вміти працювати із структурованими та не досить даними паралельно та в різних аспектах.

Дані операції дозволяють виявити приховані закономірності, які неможливо виявити звичайним аналізом. Це дає безпрецедентні можливості

оптимізації багатьох служб для обробки даних: телекомунікації, медицина, фінанси, виробництво та ін.

Основними ознаками великих даних є фізичний обсяг, швидкість приросту даних і необхідності їх швидкої обробки та можливість одночасно обробляти дані різних типів.

Виходячи з вищенаведених визначень, основні принципи роботи з великими даними такі:

- горизонтальна масштабованість є базовим принципом обробки великих даних. Як вже говорилося, великих даних з кожним днем стає все більше. Відповідно, необхідно збільшувати кількість обчислювальних вузлів, за якими розподіляються ці дані, причому обробка повинна відбуватися без погіршення продуктивності;

- принцип відмовостійкості випливає з попереднього. Оскільки обчислювальних вузлів в кластері може бути багато (іноді десятки тисяч) і їх кількість, не виключено, буде збільшуватися, зростає і ймовірність виходу машин з ладу. Методи роботи з великими даними повинні враховувати можливість таких ситуацій і передбачати превентивні заходи;

- локальність даних. Так як дані розподілені по великій кількості обчислювальних вузлів, то, якщо вони фізично знаходяться на одному сервері, а обробляються на іншому, витрати на передачу даних можуть стати невиправдано великими. Тому обробку даних бажано проводити на тій же машині, на якій вони зберігаються.

Ці принципи відрізняються від тих, які характерні для традиційних, централізованих, вертикальних моделей зберігання добре структурованих даних. Відповідно, для роботи з великими даними розробляють нові підходи і технології.

1.3 Опис видів зберігання текстової інформації

На даний момент існує дуже багато типів зберігання різної інформації. Так, для зберігання графічної інформації використовують такі типи інформації як “PNG” , “JPG” тощо. Для зберігання та передачі інформації у мережі Інтернет використовують типи “CSV”, “JSON” та “XML”.

1.3.1 Опис формату даних CSV

Як випливає з назви («comma separated values»), цей формат даних в основному представляє собою список елементів, розділених комами.

Основною перевагою цього формату даних є компактність. Формати CSV приблизно вдвічі менше форматів XML і JSON. Це головна перевага CSV, тому що він допомагає зменшити пропускну здатність.

Мінусом цього формату даних є найменша універсальність з усіх трьох форматів. Це пов'язано з тим, що для перетворення даних CSV у власну структуру даних потрібен саморобний синтаксичний аналізатор. В результаті, якщо структура даних змінюється, виникають витрати, пов'язані з необхідністю зміни або навіть перепроектувати синтаксичні аналізатори. Більш того, оскільки програма, яка створює CSV, і програма, що аналізує CSV, часто знаходяться на різних машинах, обидві програми повинні оновлюватися одночасно, щоб запобігти збій програми, яка приймає й обробляє дані.

Також CSV формат не підтримує внутрішніх структур даних. Для того щоб уникнути подібної поведінки необхідно вводити спеціальний символ для розділення основних даних і вкладених структур. Проблемою такого рішення є необхідність надання оброблювачу інформації про новий символі і його призначення.

1.3.2 Опис формату даних XML

XML означає «розширена мова розмітки». Він був створений для кращого представлення форматів даних з ієрархічною структурою.

Даний формат даних повністю підтримує ієрархічні структури даних і дуже підходить при отриманні складних даних в якості відповіді. Більшість браузерів мають вбудовані засоби читання XML, що дозволяють перевіряти файли XML. Оскільки XML був першим стандартним форматом ієрархічних даних, більшість API-інтерфейсів мають вбудовану функціональність для автоматичного перетворення потоків даних XML у власні структури даних, такі як об'єкти. Однак цей формат даних приблизно в три рази більше, ніж CSV. Це пов'язано з тим, що кожен елемент даних має пов'язаний тег параметра відкриття і закриття.

1.3.3 Опис формату даних JSON

JSON означає «представлення об'єкта Javascript». JSON формат створювався як альтернатива XML. Однак, на відміну від XML, він представляє ієрархічні дані з використанням ком, фігурних дужок і дужок.

Формат даних JSON підтримує ієрархічні дані, але за розміром він менше XML. Як випливає з назви, він також був створений для більш легкого приведення даних у стандартний для Javascript формат, що робить його дуже корисним для веб-додатків. JSON простий і компактний, як CSV, і підтримує ієрархічні дані, так як і XML. На відміну від XML, формати JSON приблизно в два рази більше форматів CSV.

1.4 Властивості інформації з елементами самоподібності

Як було описано вище, самоподібність та фрактали описують явище, коли певна властивість об’єкта – наприклад, природний образ, збіжний піддомен певних динамічних систем, часовий ряд або відношення однієї

частини структури даних до іншої – зберігається щодо масштабування в просторі і часі [1].

Багато об'єктів реального світу, наприклад, берегові лінії, мають властивість статистичного самоподібності: їх частини статистично однорідні в різних шкалах вимірювання. Також самоподібність є характерною властивістю фракталу.

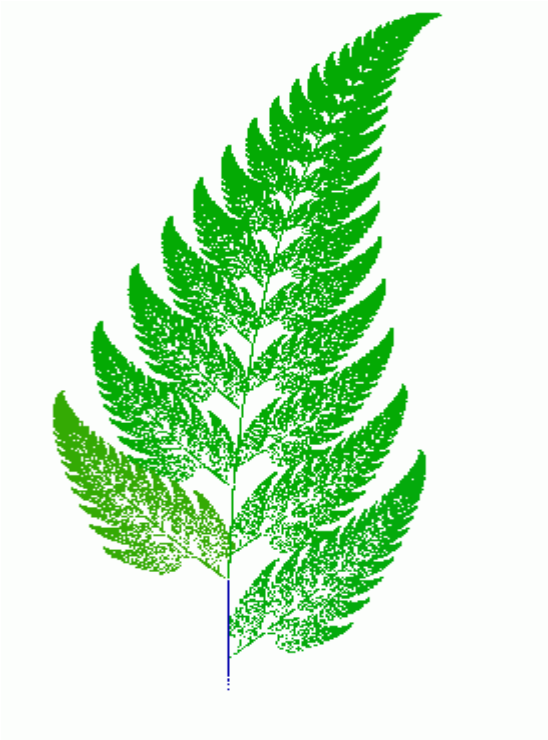

Рисунок 1.1 – Фрактал у вигляді листа папороті

Інваріантність щодо зміни шкали є однією з форм самоподібності, при якій при будь-якому наближенні знайдеться принаймні одна частина основної фігури, подібна цілої фігурі [3].

Якщо предмет є самоподібним або фрактальним, його частини при збільшенні нагадують форму цілого. Наприклад, двовимірна множина Кантора на $A = [0,1] \times [0,1]$ отримується, починаючи з суцільного або чорного одиничного квадрата, масштабуючи його розмір на 1/3, а потім розміщуючи чотири копії масштабованого квадрата біля чотирьох кутів A.

Фрактали відомі вже майже століття, добре вивчені і мають численні застосування в житті. Один з головних атрибутів, шо описує фрактал є фрактальна розмірність.

Фрактальна розмірність – один із способів визначення розмірності множини в метричному просторі. Це коефіцієнт, що описує фрактальні структури або безлічі на основі кількісної оцінки їх складності, як коефіцієнт зміни в деталі зі зміною масштабу [2]. Фрактальну розмірність n-мірного безлічі можна визначити за допомогою формули :

$$D = -\lim_{\varepsilon \to 0} \frac{\ln(N_\varepsilon)}{\ln(\varepsilon)}, \tag{1.1}$$

де N_ε, мінімальне число n-мірних «куль» радіусу, необхідних для покриття множини.

Фрактальна розмірність може приймати не ціле числове значення. На відміну від топологічної розмірності, фрактальний коефіцієнт може приймати не цілочисельне значення, показуючи те, що фрактальна безліч заповнює простір не так як його заповнює звичайна геометрична множина.

Ідея фрактальної розмірності має багато різних (але послідовних) математичних визначень. Інтуїтивно можна сприймати розмір фігури як міру того, наскільки шорсткою є фігура, або оцінку, яка відображає, наскільки фігура заповнює навколишній простір.

Ці інтуїтивні ідеї можна зробити математично точними. Щоб проілюструвати дробовий вимір, можна взяти аркуш паперу, який (практично) є двовимірним. Тверда куля є тривимірною і займає більше місця, ніж аркуш паперу.

Якщо пом’яти папір у кульку, то вийде фрактальна форма, яка заповнює більше простору, ніж папір, але не стільки місця, скільки суцільна сфера. Він оцінюється приблизно в 2,5 за свого розміру.

Так само легені мають розмір приблизно 2,97 – їх фрактальна геометрія дозволяє їм упакувати велику площу поверхні (кілька тенісних кортів) у невеликий об’єм (кілька тенісних м’ячів).

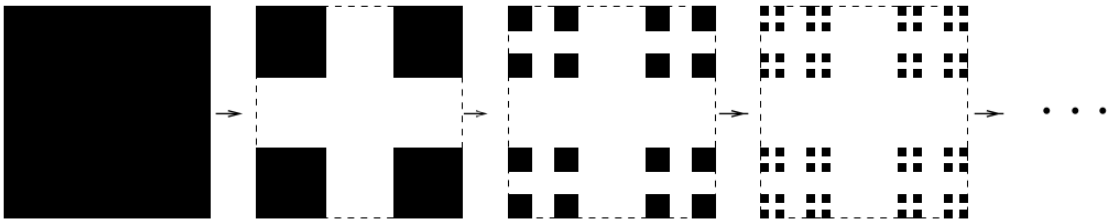

Рисунок 1.2 – Двовимірна множина Кантора

Наприклад, крива з фрактальної розмірністю дуже близькою до 1, скажімо 1.10, поводиться цілком як звичайна лінія, але крива з фрактальної розмірністю 1.9 намотана в просторі, майже як поверхня. Ідея фрактальної розмірності лежить в нетрадиційному поданні масштабу і розмірності.

Якщо застосовується той самий процес масштабування з подальшим перекладом рекурсивно до отриманих об'єктів на нескінченність, встановлений таким чином ліміт визначає двомірну множину Кантора.

Обмежувальний об'єкт, який визначається як нескінченний перетин ітерацій, має властивість самоподібності, якщо розділені певним чином елементи є копією вихідного елемента на першій ітерації.

Застосування фракталів варіюється від економіки до географії, медичної візуалізації та мистецтва. Фрактали забезпечують мову для опису природи та інших систем.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

На даний момент існує безліч алгоритмів з аналізу інформації, які використовуються в різних цілях по всьому світу. Одним із прикладів таких алгоритмів є алгоритми кластеризації даних (Clustering Data Algorithms).

2.1 Опис принципів роботи алгоритмів кластеризації даних

Збільшення обсягів оброблюваної інформації в різних сферах діяльності стало причиною появи алгоритмів кластеризації даних. З появою необхідності обробки інформації у великих розмірах, їх стали використовувати у самих різних додатках, включаючи обробку зображень, обчислювальну біологію, мобільний зв'язок, медицину і економіку.

Основна проблема алгоритмів кластеризації даних в тому, що їх не можна стандартизувати. Розроблений алгоритм може дати найкращий результат з одним типом набору даних, але може дати збій або дати поганий результат з набором даних інших типів. Хоча було багато спроб стандартизувати алгоритми, які можуть добре працювати у всіх випадках та сценаріїв. До теперішнього часу було запропоновано безліч алгоритмів кластеризації. Однак кожен алгоритм має свої переваги і недоліки і не може працювати у всіх ситуаціях.

Кластеризація – це процес, який розділяє даний набір інформації на однорідні групи на основі заданих параметрів, так що схожі об'єкти зберігаються в групі, а різнорідні об'єкти – у різних групах [19].

Окрім терміна кластеризація, існує ряд термінів з подібними значеннями, включаючи автоматичну класифікацію, числову таксономію, ботріологію типологічний аналіз та виявлення спільноти. Тонкі відмінності часто полягають у використанні результатів: тоді як при видобуванні даних

представляють інтерес результуючі групи, в автоматичній класифікації представляє інтерес результуюча дискримінаційна сила.

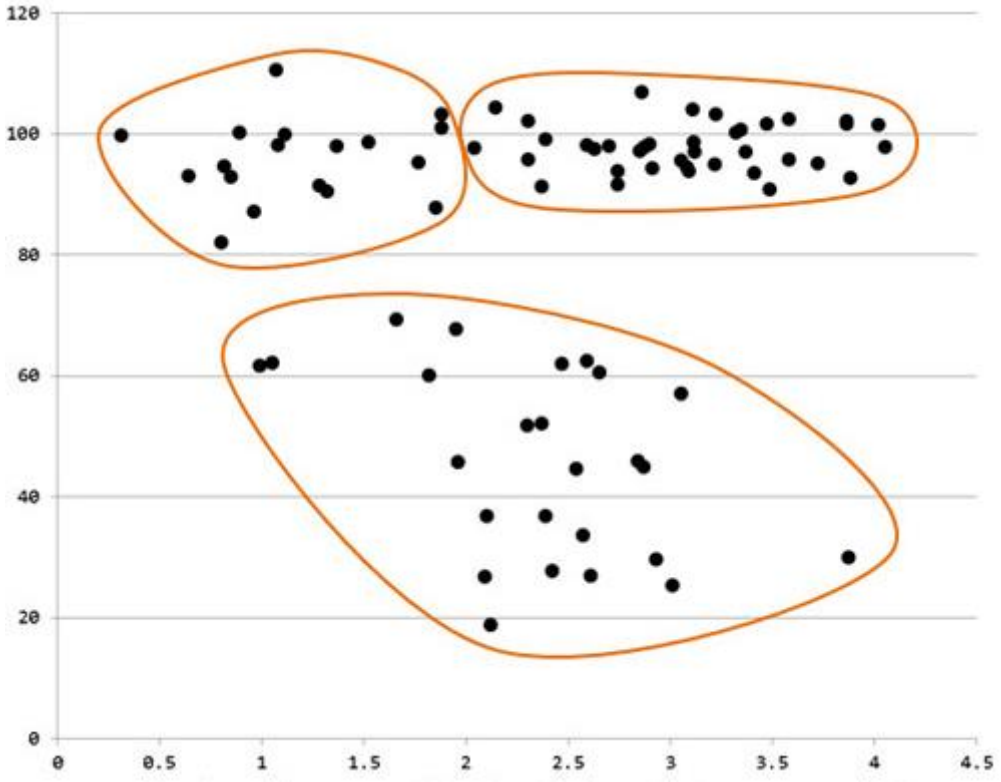

Рисунок 2.1 – Візуалізація кластеризації набору даних.

Щоб алгоритм кластеризації був вигідним і ефективним, необхідно виконання деяких умов:

- масштабованість – дані повинні бути масштабованими, інакше ми можемо отримати невірний результат. В іншому випадку результат роботи алгоритму може бути помилковим;
- алгоритм кластеризації повинен вміти працювати з різними типами атрибутів;
- алгоритм кластеризації повинен вміти знаходити кластерні дані довільної форми;
- алгоритм кластеризації повинен бути нечутливим до шумів і викидів;
- інтерпретованість і зручність використання – отриманий результат

повинен мати здатність бути інтерпретованим і придатним для використання, щоб можна було отримати максимальні знання про вхідних параметрах;

- алгоритм кластеризації повинен вміти працювати з набором даних високої розмірності.

Існує безліч алгоритмів кластеризації даних, кожен з яких має свої плюси і мінуси, тому для отримання більш детальної картини дослідження і розробка буде проводитися на прикладі 3 алгоритмів кластеризації даних, а саме:

- метод кластеризації k-means;
- метод кластеризації з використанням моделей Гауса;
- метод кластеризації k-means з розподіленням.

Для того щоб визначити для яких завдань використовувати той чи інший алгоритм, необхідно провести їх детальне дослідження.

2.2 Метод кластеризації k-means

Алгоритм кластеризації k-means – один з найпростіших алгоритмів розбиття вхідних даних по подібним ділянкам (кластерів) для їх подальшого використання. Даний алгоритм часто використовується для підготовки даних для машинного навчання.

Процедура використовує простий і легкий спосіб класифікації заданого набору даних через певну кількість [20] кластерів (припустимо, k кластерів), фіксованих апріорі. Основна ідея полягає в тому, щоб визначити k центрів, по одному для кожного кластера. Ці центри варто розставляти дуже уважно, адже різне розташування дає різний результат. Отже, кращий вибір – розмістити їх якомога далі один від одного

Наступний крок - взяти кожну точку, що належить заданому набору даних, і пов'язати її з найближчим центром. Коли немає точок, що очікують, перший крок завершено, і розпочато ранній груповий етап. На цьому етапі

нам необхідно перерахувати k нових центроїдів як барицентр кластерів, отриманих на попередньому кроку.

Після того, як у нас будуть ці k нових центроїдів, необхідно виконати нову прив'язку між тими ж точками набору даних і найближчим новим центром. Був створений цикл. В результаті цього циклу ми можемо помітити, що k центрів крок за кроком змінюють своє становище, поки не перестануть відбуватися зміни або, іншими словами, центри більше не будуть переміщатися [19]. Нарешті, цей алгоритм спрямований на мінімізацію цільової функції, відомої як функція квадрата помилки, визначається таким чином:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (||x_i - v_i||)^2 \qquad (2.1)$$

де $||x_i - v_i||$ – Евклідова дистанція між x_i та v_i;

c_i – кількість точок даних на i-тому кластері;

c – кількість кластерних центрів.

2.2.1 Етапи виконання алгоритму k-means

Нехай $X = \{x1, x2, x3, \ldots\ldots\ldots, xn\}$ – набір точок даних, а $V = \{v1, v2, \ldots, Vc\}$ – набір центрів.

1. Необхідно випадковим чином вибрати центри кластерів «с».

2. Обчислити відстань між кожною точкою даних та центрами кластера.

3. Призначити точку даних центру кластера, відстань якого від центру кластера мінімальна від усіх центрів кластера.

4. Перерахувати новий центр кластера, використовуючи:

$$v_i = \frac{1}{c_i}\sum_{j=1}^{c_i} x_i \qquad (2.2)$$

5. Розрахувати відстань між кожною точкою даних та новими отриманими центрами кластера.

6. Якщо жодна точка даних не була перепризначена, необхідно зупинитися, інакше виконання алгоритму повинно перейти на третій крок.

2.2.2 Переваги методу k-means

Метод кластеризації k-means для інформації з елементами самоподібності має наступні переваги:

- швидкий, надійний і легкий для розуміння;
- відносно ефективний: O (n);
- дає найкращий результат, коли набори даних відрізняються або добре відокремлюються один від одного.

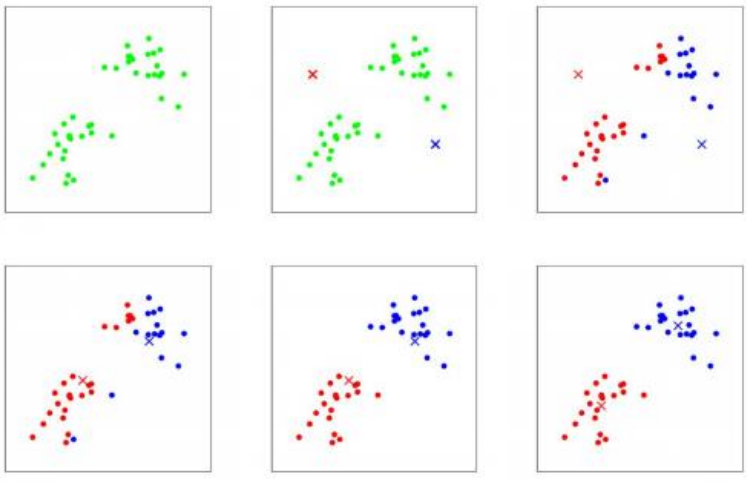

Рисунок 2.2 – Етапи виконання методу k-means.

На рисунку 2.2 зображене зразкове виконання методу k-means, на якому можна побачити виконання різних етапів, які описувалися вище.

2.2.3 Недоліки методу k-means

Однак даний метод не є ідеальним на усіх типах інформації. Даний метод має наступні недоліки:

- алгоритм навчання вимагає апріорі вказувати кількість центрів кластера;

- використання ексклюзивного призначення, якщо у двох кластерах сильно перекриваються дані, тоді k-means не зможуть визначити наявність двох кластерів;

- алгоритм навчання не є інваріантним щодо нелінійних перетворень, тобто з різним поданням даних, які ми отримуємо різні результати (дані, представлені у формі декартових координат і полярних координат, дадуть різні результати);

- евклідова міра відстані може нерівномірно зважувати основні фактори.

Алгоритм k-means добре фіксує структуру даних, якщо кластери мають сферичну форму [19]. Завжди намагаються побудувати гарну сферичну форму навколо центроїда. Це означає, що щомиті кластери мають складні геометричні фігури, kmeans погано справляється з кластеризацією даних. Ми проілюструємо три випадки, коли kmeans не буде працювати добре.

По-перше, алгоритм kmeans не дозволяє точкам даних, які знаходяться далеко один від одного, спільно використовувати один і той же кластер, хоча вони, очевидно, належать одному кластеру. Нижче наведено приклад точок даних на двох різних горизонтальних лініях (див рис 2.3), що ілюструє, як kmeans намагається згрупувати половину точок даних кожної горизонтальної лінії.

Алгоритм вважає точку «B» ближче до точки «A», ніж точку «C», оскільки вони мають сферичну форму. Отже, точки «A» і «B» будуть знаходитись в одному кластері, а точки «C» – в іншому кластері. Зверніть увагу, що метод ієрархічної кластеризації єдиного зв’язку отримує це право,

оскільки він не розділяє подібні точки). Як результат алгоритм k-means, не зможе правильно зрозуміти скупчення. Оскільки він намагається мінімізувати варіації всередині кластера, це надає більшої ваги більшим кластерам, ніж меншим. Іншими словами, точки даних у менших кластерах можуть залишатися подалі від центроїда, щоб більше зосередитись на більшому кластері.

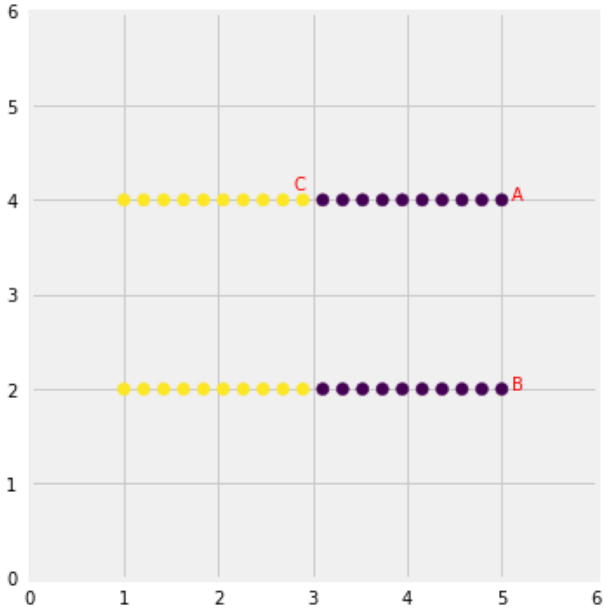

Рисунок 2.3 – приклад набору точок на горизонтальних лініях

Метод k-means – це швидкий, надійний і простий алгоритм, який дає надійні результати, коли набори даних відрізняються або добре відокремлюються один від одного лінійно. Найкраще використовувати, коли кількість центрів кластера вказана через чітко визначений перелік типів.

Однак важливо мати на увазі, що кластеризація k-means може не працювати належним чином, якщо вона містить дані, що сильно перекриваються, якщо евклідова відстань погано вимірює основні фактори, або якщо дані шумні або повні викидів.

2.3 Метод кластеризації з використанням моделей Гауса

Метод гаусових моделей (ГМ) надає нам більшу гнучкість, ніж метод k-means. При використанні ГМ ми припускаємо, що точки даних розподілені по Гауссу. Таким чином, у нас є два параметри для опису форми скупчень: середнє та стандартне відхилення. Беручи приклад у двох вимірах, це означає, що скупчення можуть приймати будь-яку еліптичну форму (оскільки ми маємо стандартне відхилення в обох напрямках *x* та *y*). Таким чином, кожен розподіл Гауса присвоюється одному кластеру.

Модель кластеризації k-means, розглянута в попередньому розділі, проста і порівняно легка для розуміння, але її простота призводить до практичних проблем у її застосуванні. Зокрема, неімовірнісний характер k-means та використання ним простої віддаленості від центру кластера для призначення членства в кластері призводить до низької ефективності для багатьох ситуацій у реальному світі. У цьому розділі ми розглянемо моделі Гаусова суміші (ГММ), які можна розглядати як продовження ідей, що лежать в основі k-середніх значень, але також можуть бути потужним інструментом для оцінки, окрім простої кластеризації.

Основна складність у вивченні методу моделей гаусса на основі невизначених даних полягає в тому, що, як правило, невідомо, які пункти прийшли з якого прихованого компонента (якщо хтось має доступ до цієї інформації, стає дуже легко встановити окремий розподіл Гауса до кожного набору точок даних). Максимізація очікувань – це обґрунтований статистичний алгоритм, щоб обійти цю проблему за допомогою ітераційного процесу. Спочатку передбачаються випадкові компоненти (випадково відцентровані на точках даних, вивчені з k-середніх значень, або навіть просто розподілені навколо координат координат) і обчислюється для кожної точки ймовірність генерування кожним компонентом моделі. Потім можна налаштувати параметри, щоб максимізувати ймовірність даних, що

отримують ці призначення. Повторення цього процесу гарантовано завжди сходиться до локального оптимуму.

Існує дві ключові переваги використання ГМ. По-перше, ГМ набагато гнучкіші з точки зору кластерної коваріації, ніж метод k-means; завдяки параметру стандартного відхилення скупчення можуть приймати будь-яку форму еліпса, а не обмежуватися колами. По-друге, оскільки ГМ використовують ймовірності, вони можуть мати кілька кластерів на точку даних. Отже, якщо точка даних знаходиться всередині двох кластерів, що перекриваються, ми можемо просто визначити її клас, сказавши, що він належить X-відсотку до класу 1 і Y-відсотка до класу 2. Тобто ГМ підтримують змішане членство. У даному методі є один недолік, який стосується його складної реалізації.

Щоб знайти параметри Гауса для кожного кластера (наприклад, середнє та стандартне відхилення), ми використаємо алгоритм оптимізації, який називається Очікування – Максимізація (ОМ). Таким чином ми можемо продовжити процес кластеризації очікувань – максимізації за допомогою ГМ.

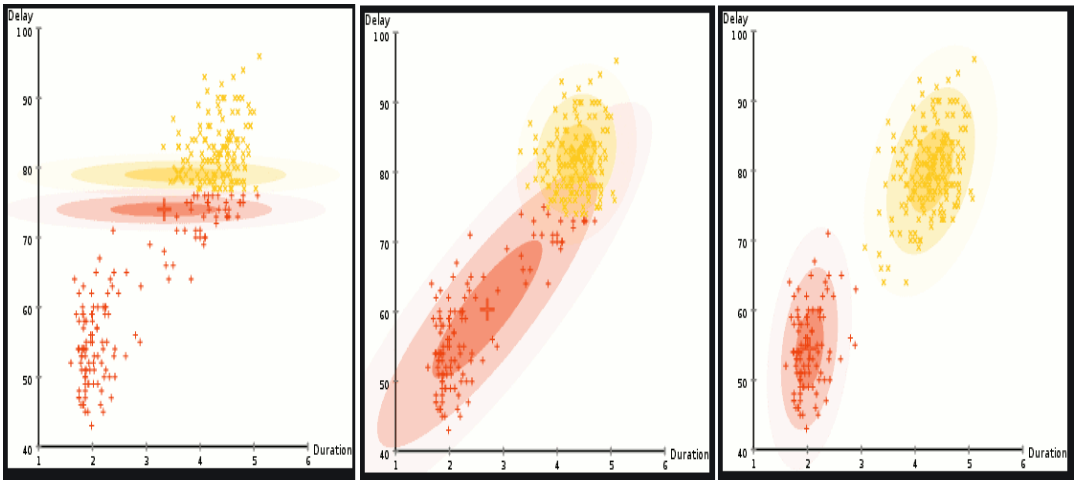

Рисунок 2.4 – Кластеризація даних, з використанням ГМ

Враховуючи ці розподіли Гауса для кожного кластера, необхідно обчислити ймовірність того, що кожна точка даних належить певному

кластеру. Чим ближче точка до центру Гаусса, тим більша ймовірність того, що вона належить до цього скупчення [20]. Це має мати інтуїтивний сенс, оскільки при гауссовому розподілі ми припускаємо, що більшість даних лежить ближче до центру кластера.

На основі цих ймовірностей обчислюється новий набір параметрів (формула 3.3) для розподілу Гауса таким чином, щоб максимізувати ймовірність знаходження точок даних у кластерах.

$$\mu_i(t+1) = \frac{\sum_k \quad P(\omega_i|x_k,\lambda_i)x_k}{\sum_k \quad P(\omega_i|x_k,\lambda_i)} \tag{2.3}$$

де $\sum_k \quad P(\omega_i|x_k,\lambda_i)$ – це кількість точок даних.

Далі необхідно обчислити ці нові параметри, використовуючи зважену суму позицій точки даних, де ваги – це ймовірності точки даних, що належить до цього конкретного кластера. Прикладом візуалізації цього процесу, можна поглянути на графік, зокрема на жовте скупчення (рисунок 2.5). Розподіл розпочинається випадковим чином на першій ітерації, але ми бачимо, що більшість жовтих точок розташовані праворуч від цього розподілу. Коли ми обчислюємо суму, зважену за ймовірностями, навіть незважаючи на те, що біля центру є кілька точок, більшість із них знаходиться праворуч. Таким чином, природно середнє значення розподілу зміщується ближче до набору точок. Ми також можемо помітити, що більшість пунктів на графіку (рисунок 2.5) розташовані у правому верхньому кутку та лівому нижньому. Тому стандартне відхилення змінюється, щоб створити еліпс, який більше підходить до цих точок, щоб максимізувати суму, зважену за ймовірностями.

Етапи 2 і 3 повторюються повторно до збіжності, де розподіли не сильно змінюються від ітерації до ітерації.

3 АЛГОРИТМ РОБОТИ МОДИФІКОВАНОГО МЕТОДУ КЛАСТЕРИЗАЦІЇ

3.1 Основні положення та умовних позначень

Найпоширеніший спосіб зберігання даних – це таблиці з визначеною кількістю стовпців, що позначають властивості даних, і рядків, які використовуються для зберігання елементів даних. У цій роботі позначимо таблиці як набори даних, властивості даних як атрибути та елементи даних (або об'єкти) як точки в просторі властивостей. Таким чином, набір даних розглядається як точки в E-мірному просторі, де E – число атрибутів. У даній роботі будуть розглянуті набори даних, які описують складні набори інформації, як правило, що складаються з числових атрибутів.

Таблиця 3.1 – Умовні позначення

Символи	Визначення
E	Вкладена розмірність (Евклідова розмірність)
D	Фрактальна розмірність (внутрішня розмріність)
N	Кількість точок у наборі даних
$C_{r,i}$	Кількість точок в i-клітинці таблиці сторони r
r	Межа клітинки у таблиці
$S(r)$	Загальна кількість заповнень для конкретної сторони r клітинки у таблиці
R	Кількість сторін r для позначення $S(r)$

Властивості, які були отримані із зображень – відомі приклади багатопросторових наборів даних, які використовуються в системах вилучення інформації на основі їх змісту (content-based image retrieval systems) [17]. Для таких наборів даних важко обрати набір атрибутів, які можуть бути призначені ключовими для даного набору даних. Таким чином,

якщо хтось зацікавлений у створенні індексної структури для набору даних, необхідно розглянути усі можливі атрибути. Це призводить до великих затрат часу та можливої недостовірності подальшої кластеризації.

3.2 Визначення поняття вбудованого та внутрішнього вимірів

Мета першого етапу запропонованого методу обробки даних – знайти підмножину атрибутів, які можна відкинути під час створення індексів або застосовування методів кластеризації без шкоди для результатів. Атрибути що розраховуються з інших атрибутів, є першими кандидатами на видалення, якщо способи їх обчислення відомі кінцевому користувачеві. Однак загалом ця кореляція невідома. Таким чином, мета перетворюється на виявлення співвідношення між атрибутами в наборі даних і скільки зайвих атрибутів має набір даних. Це веде до визначення вбудованих та внутрішніх вимірів.

Вбудована розмірність E набору даних – це розмірність адресного простору даних. Тобто це кількість атрибутів набору даних. Набір даних може представляти просторовий об’єкт, розмірність якого менша за простір, де він «вбудований». Наприклад, лінія має внутрішню розмірність 1, незалежно від того, знаходиться вона у більшому розмірному просторі.

Внутрішня розмірність набору даних – це розмірність просторового об’єкта, представленого набором даних, незалежно від простору, де він вбудований [17].

Концептуально, якщо в наборі даних є всі його змінні, незалежні від інших, то його внутрішня розмірність дорівнює вбудованій розмірності. Однак, коли існує взаємозв'язок між двома або більше змінних, внутрішня розмірність набору даних відповідно зменшується.

Зазвичай розмірність вбудованого набору даних приховує фактичні характеристики, і взагалі кореляції між змінними в реальних наборах даних невідомі і навіть існування кореляцій. Знаючи внутрішню розмірність можна вирішити, скільки атрибутів насправді характеризують набір даних.

3.3 Визначення фрактального набору даних та його фрактальної розмірності

Фрактальний набір даних – це набір даних, який має відповідну характеристику самоподібності. Це означає, що набір даних має приблизно однакові властивості при зміні масштабу або розміру, тобто деталі фракталу будь-якого розміру подібні (точно або статистично) до цілого фракталу. Ця ідея проілюстрована на малюнку 3.1, де показано перші три кроки для побудови Трикутника Серпінського. Трикутник Серпінського будується за допомогою рівностороннього трикутника ABC, виключаючи його середній трикутник A’B’C’ і рекурсивно повторюючи цю процедуру для кожного з отриманих менших трикутників.

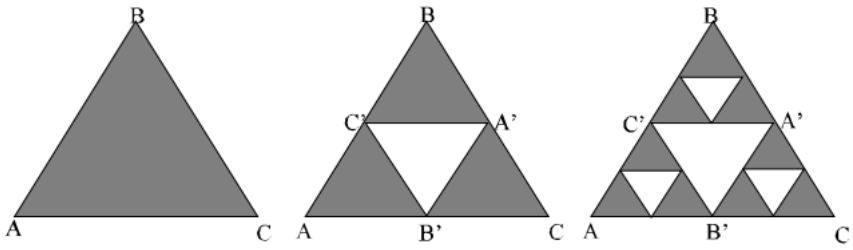

Рисунок 3.1 – Трикутник Серпінського

Трикутник Серпінського генерується після нескінченних ітерацій цієї процедури. Він має нескінченний периметр, тому він не є одновимірним об’єктом. І він не має площі, тому він не є двовимірним об’єктом. Насправді він має внутрішній вимір, рівний 1,58 [17]. Для дійсного набору точок ми вимірюємо фрактальну розмірність за допомогою графіку підрахунку блоків, що є основою запропонованого алгоритму, який буде описаний далі.

Кореляційна фрактальна розмірність D2 набору даних, який має властивість самоподібності в діапазоні {r1, r2}, вираховується по формулі:

$$D_2 = \frac{\partial \log \sum_i {C_{r,i}}^2}{\partial \log r}, \qquad r \in [r_1, r_2] \tag{3.1}$$

Відтепер необхідно використати кореляційну фрактальну розмірність D_2 як внутрішню розмірність *D*.

Для евклідових об'єктів їх фрактальна розмірність відповідає їх евклідовій розмірності, та їх фрактальна розмірність завжди є цілим числом. Наприклад, лінії, окружності та всі стандартні криві мають $D = 1$; площини, кола, квадрати та поверхні мають $D = 2$ тощо. Дійсно, відрізок у будь-якому n-мірному простір завжди матиме $D = 1$, як і квадрат завжди матиме $D = 2$, навіть якщо точки знаходяться у просторі вищих розмірів [18]. Однак, фрактальна розмірність набору даних не може бути більшою за розмірність його вбудованої розмірності. Багато реальних наборів даних – це фрактали [17, 18]. Таким чином, для заданого набору даних, який буде описаний нижче, ми можемо скористатися перевагами роботи з їх кореляційною фрактальною розмірністю як зі своєю власною розмірністю *D*.

3.4 Опис алгоритму підрахунку фрактальної розмірності

Практичним способом оцінки *D* внутрішнього набору даних є використання методу підрахунку блоків (box-counting approach). Теоретично цей метод дає близьке наближення фрактальної розмірності. Розроблений алгоритм має асимптотичну складність *O (N)* [18].

Розглянемо адресний простір множини точок в *E*-мірному просторі та накладемо сітку з розмірністю *E*, у якій розмір клітинок буде дорівнювати *r*. Зосереджуючись на і-й клітинці, нехай $C_{r,i}$ - кількість точок у кожній клітинці. Потім необхідно підрахувати значення $S(r) = \sum_i {C_{r,i}}^2$. Фрактальна розмірність буде розраховуватись як похідна від $\log S(r)$ відносно логарифму радіуса. Припускаємо що дані самоподібні, тому результат похідної є константним значенням. Таким чином ми можем підрахувати фрактальну

розмірність *D* заданого набору даних, наносячи на графік $S(r)$ для різних значень радіусу *r*, та підраховуючи нахил прямої результатів.

Щоб уникнути повторного читання набору даних для кожного значення радіуса, пропонується створити багаторівневу структуру сітки, де кожен рівень має радіус, який дорівнює половині розміру попереднього рівня (*r* = 1, 1/2, 1/4, 1/8 тощо). Кожному рівню структури відповідає різний радіус, тому глибина конструкції дорівнює кількості точок на отриманому графіку. Ця структура створюється в основній пам'яті, тому кількість точок на графіку обмежується кількістю доступною в основній пам’яті. Якщо цей графік є лінійним для відповідного діапазону радіусів, набір даних є фрактальним і його фрактальна розмірність *D* є значенням відхилення на прямій результатів.

3.5 Опис алгоритму вибору атрибутів

Описуваний алгоритм використовує підхід зворотного видалення атрибутів. Запропонована ідея полягає в обчисленні кореляційного фракталу вимірювання цілого набору даних, а також для обчислення його *pD*, відкидаючи по одному з атрибутів *E*.

Таким чином, це призведе до *E* часткових фрактальних розмірів. Алгоритм буде продовжувати вибирати атрибут, який веде до мінімальної різниці в *pD* для всього набору даних. Якщо ця різниця знаходиться в межах невеликого порогу, можна бути впевнені, що цей атрибут майже нічого не сприяє загальним характеристикам набору даних. Тому цей атрибут можна виключити зі списку важливих атрибутів, що характеризують набору даних. Поріг залежить від того, наскільки точним повинен бути результуючий набір даних для збереження характеристики вихідного набору даних.

Можна виділити наступні ключові кроки виконання алгоритму вибору атрибутів:

1. Підрахувати фрактальну розмірність *D* усього набору даних.

2. Умовно зазначити усі атрибути як важливі.

3. Встановити початкове значення фрактальної розмірності *D*.

4. Підрахувати часткову фрактальну розмірність pD_i, використовуючи усі важливі атрибути, виключаючи атрибут *i*.

5. Виконати крок 4 для кожного атрибуту.

6. Відсортувати часткові фрактальні розмірності та обрати атрибут *a*, який веде до найменшої різниці $D - pD_i$.

7. Встановити отримане значення pD_i як початкове значення розмірності *D* та виключити атрибут *a* зі списку атрибутів.

8. Виконувати крок 5, 6, 7 доки не закінчиться кількість атрибутів *E*.

За допомогою даного алгоритму була встановлена необхідна кількість полів, які можна використати для задання ключового поля “features” в алгоритмі кластеризації, який описаний у наступному розділі.

3.6 Основні положення кластеризації k-means з розподіленням

Кластеризація k-means з розподіленням – це гібридний підхід між розділовою ієрархічною кластеризацією (кластеризація зверху вниз) та кластеризацією k-means. Замість того, щоб розділити набір даних на K кластери в кожній ітерації, алгоритм ділення k-means розділяє один кластер на два підкластери на кожному кроці бісекціонування (за допомогою k-середніх), поки не буде отримано k кластерів.

Стандартний алгоритм ієрархічної агломеративної кластеризації має часову складність $O(n^3)$ і вимагає $\Omega(n^2)$ пам’яті, що робить її занадто повільною навіть для середніх наборів даних. Однак для деяких особливих випадків відомі оптимальні ефективні агломеративні методи (складності $O(n^2)$): SLINK для одинарного зв’язку та CLINK для кластеризації з повним зв’язком. За допомогою купи, час виконання загального випадку можна зменшити до $O(n^2 \log n)$, покращення вищезгаданої межі $O(n^3)$, за рахунок подальшого збільшення вимоги до пам'яті. У багатьох випадках накладні

витрати на пам’ять цього підходу занадто великі, щоб зробити їх практично придатними для використання.

За винятком особливого випадку єдиного зв’язку, жоден з алгоритмів (крім вичерпного пошуку в $O(2^n)$) можна гарантовано знайти оптимальне рішення.

Групова кластеризація з вичерпним пошуком – це $O(2^n)$, але загальновживаним є використання швидша евристика для вибору розбиття, наприклад, k-означає.

Замість того, щоб розділити набір даних на K кластерів в кожній ітерації, алгоритм ділення k-means розділяє один кластер на два підкластери на кожному кроці бісекціонування (за допомогою k-means), поки не буде отримано k кластерів.

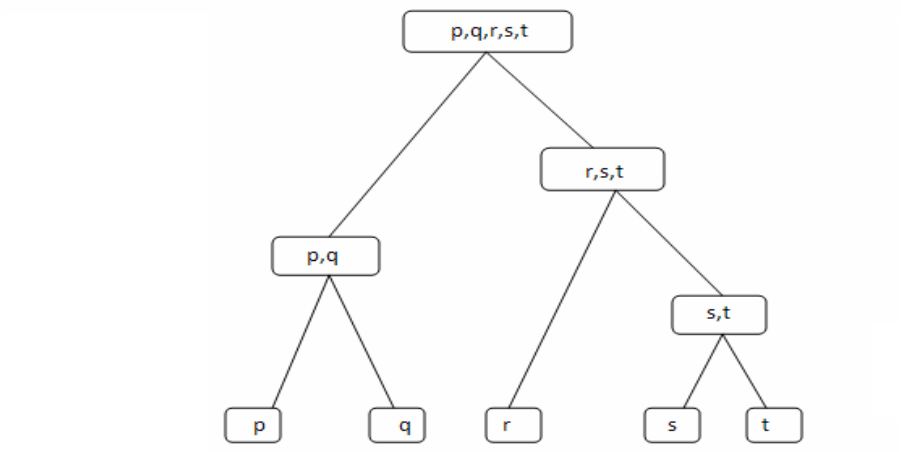

Рис 3.2 – Ієрархічне розподілення зверху вниз

Розділювальний алгоритм кластеризації (кластеризація зверху вниз) – це підхід кластеризації зверху вниз, спочатку всі точки набору даних належать одному кластеру, і розділення виконується рекурсивно, коли один рухається вниз по ієрархії.

Розділювальна кластеризація починається на верхньому рівні з одного кластера і ділить його до нижнього рівня. Для того, щоб вирішити, які

кластери слід розділити, необхідна міра відмінності між наборами даних. Це досягається використанням міри відстані між парами спостережень.

Етапи виконання алгоритму k-means з розподіленням:

1. Встановлення K, щоб визначити номер кластеру.

2 Призначення усіх даних до одного кластеру.

3 Використання k-means з коефіцієнтом K = 2, щоб розділити кластер.

4 Виміряти відстань для кожного внутрішнього кластера по формулі (2.4):

$$\sum_{i=0}^{n} (X_i - \underline{X})^2 \qquad (2.4)$$

5. Обрати кластер, що має найбільшу відстань, і розділити його на 2 кластери, з використанням k-means.

6. Повторювати кроки 3-5, доки кількість елементів в ієрархії не буде дорівнюватися K=1.

Існує 4 основних методів вимірювання відстані між кластерами, які називаються методами зв'язування:

- одне зв’язування: обчислює мінімальну відстань між кластерами перед їх об’єднанням;

- повне зв'язування: обчислює максимальну відстань між кластерами перед їх об'єднанням;

- середнє зв'язування: обчислює середню відстань між кластерами перед їх об'єднанням;

- зв'язок центроїдів: обчислює центроїди для обох кластерів, потім обчислює відстань між ними перед їх об'єднанням.

Ідея усього алгоритму полягає в ітеративному розділенні на випадкове двійкове дерево, де кожне розбиття (вузол з двома дочірніми елементами) відповідає розбиттю точок кожного кластеру на 2.

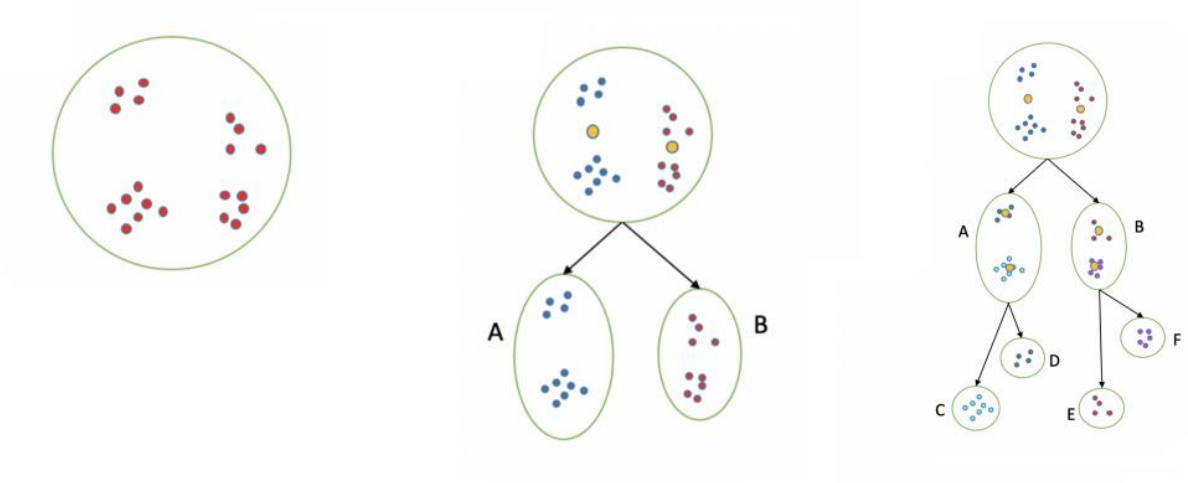

Рисунок 3.3 – Етапи виконання алгоритму k-means з розподіленням

Після кожного етапу розподілення необхідно виміряти відстань до кожного внутрішнього кластеру по наступній формулі:

$$\sum_{i=0}^{n} (X_i - \underline{X})^2 \qquad (2.4)$$

Робота алгоритму зупиняється, коли алгоритм не зможе розподілити дані на задану кількість точок даних в останньому елементі дерева.

3.7 Переваги методу k-means з розподіленням

Метод k-means з розподіленням більш ефективний при великому значенні коефіцієнту K. Для алгоритму k-means обчислення включає кожну точку набору даних і k-центроїдів. З іншого боку, на кожному кроці розподілення k-means у обчисленні беруть участь лише точки даних одного кластера та двох центроїдів. Таким чином, час обчислень скорочується.

Якщо порівнювати звичайний алгоритм k-means та k-means з розподіленням, то можна виділити наступні положення:

- поділення k-середніх ефективніше, коли K велике. Для алгоритму kmeans обчислення включає кожну точку даних набору даних і k центроїдів.

З іншого боку, на кожному кроці розподілення кластеру на дві частини значень у обчисленні беруть участь лише точки даних одного кластера та двох центроїдів. Таким чином, час обчислень скорочується;

- поділення k-середніх утворює кластери однакових розмірів, тоді як k-середні, як відомо, утворюють кластери дуже різних розмірів.

Таблиця 3.2 – Порівняння ефективності кластеризації даних

Алгоритм	Кількість рядків з даними	Кількість правильних розподілень	Кількість помилок	Час виконання
k-means	100	83	17	0.237
k-means з розподіленням	100	73	22	0.672

Також при виконанні алгоритму на однаковій кількості даних можна зазначити, що при однакових K алгоритм k-means з розподіленням справляється з поставленою задачею кластеризації даних трохи швидше та з меншою кількістю помилок. Останній фактор є найважливішим при роботі з даними такого формату бо при неправильному розподіленні на великому масиві даних, подальший аналіз та виявлення залежності між різними кластерами може стати доволі скрутним та незручним.

4 РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ПЕРВИННОЇ ОБРОБКИ ІНФОРМАЦІЇ

4.1 Використовувані технологій розробки інтерфейсу для користувача

4.1.1 Аналіз мови програмування JavaScript

Для розробки інтерфейсу користувача було прийняте рішення використовувати мову програмування JavaScript.

JavaScript – це легка, інтерпретована або компільована “just-in-time” (у час виконання) мова програмування, яка використовує функції першого класу. Хоча вона найбільш відома як мова сценаріїв для веб-сторінок, її використовують також багато середовищ, що не належать до браузера, такі як Node.js, Apache CouchDB та Adobe Acrobat. JavaScript є однопотоковою динамічною мовою, що містить у собі багато парадигм:

- об’єктно-орієнтоване програмування;
- імперативне програмування;
- декларативне програмування (функціональне програмування).

При генерації сторінок в мережі Інтернет виникає дилема, пов'язана з архітектурою «клієнт-сервер». Сторінки можна генерувати як на стороні клієнта, так і на стороні сервера. Таким чином, JavaScript є мовою управління сценарію перегляду гіпертекстових сторінок Web на стороні клієнта.

Основна ідея JavaScript полягає в можливості зміни значень атрибутів HTML-контейнерів і властивостей середовища відображення в процесі перегляду HTML-сторінки користувачем [7]. При цьому перевантаження сторінки не відбувається.

На практиці це виражається в тому, що можна, наприклад, змінити колір фону сторінки або інтегровану в документ картинку, відкрити нове вікно або видати попередження.

Для створення механізму управління сторінками на клієнтській стороні було запропоновано використовувати об'єктну модель документа. Суть моделі в тому, що кожен HTML-контейнер – це об'єкт, який характеризується трійкою:

- властивості;
- методи;
- події.

Об'єктну модель можна представити як спосіб зв'язку між сторінками і браузером. Об'єктна модель – це представлення об'єктів, методів, властивостей і подій, які присутні і відбуваються в програмному забезпеченні браузера, у вигляді, зручному для роботи з ними коду HTML і вихідного тексту сценарію на сторінці [10].

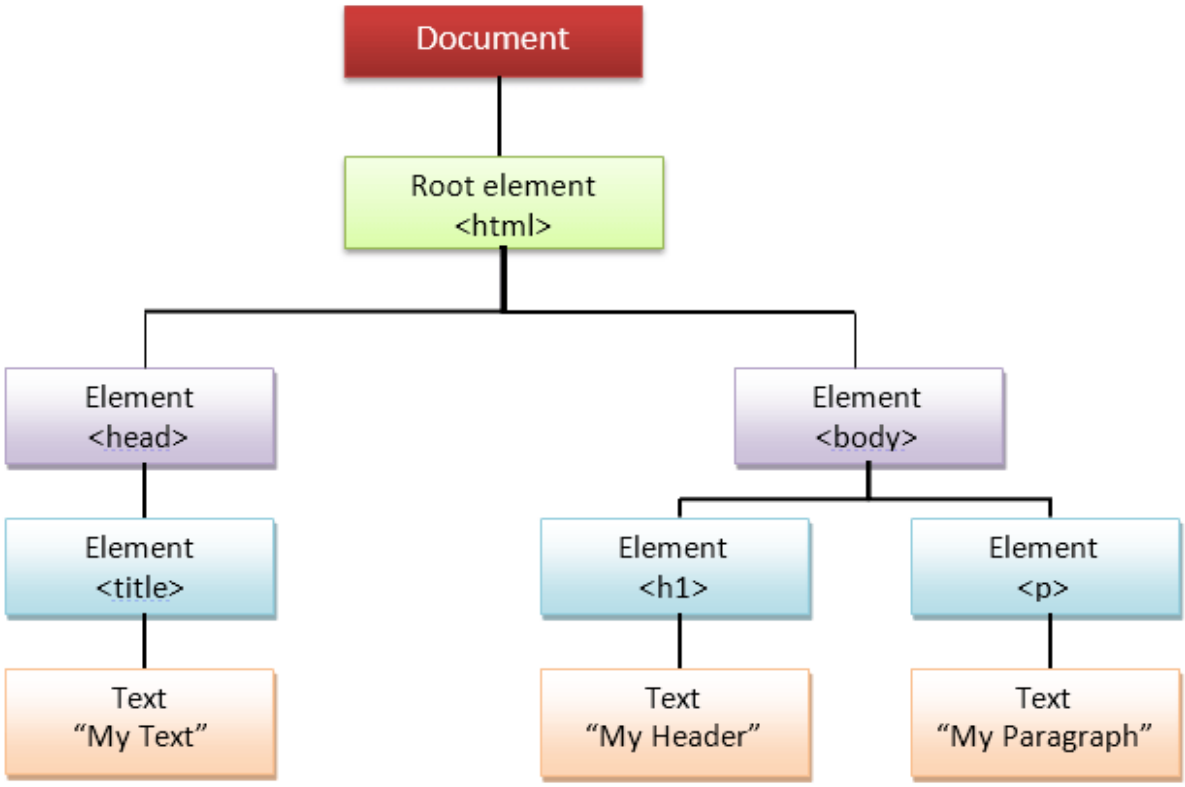

Рисунок 4.1 – Об'єктна модель JavaScript для HTML –сторінки

Ми можемо з її допомогою повідомляти наші побажання браузеру і далі – відвідувачеві сторінки. Браузер виконає наші команди і відповідно змінить сторінку на екрані.

Об'єкти з однаковим набором властивостей, методів і подій об'єднуються в класи однотипних об'єктів. Класи – це описи можливих об'єктів. Самі об'єкти з'являються тільки після завантаження документа браузером або як результат роботи програми. Про це потрібно завжди пам'ятати, щоб не звернутися до об'єкта, якого немає.

4.1.2 Опис роботи движкаV8 для JavaScript

JavaScript-движок – це програма, або, іншими словами, інтерпретатор, що виконує код, написаний на JavaScript. Движок може бути реалізований з використанням різних підходів: у вигляді звичайного інтерпретатора, у вигляді динамічного компілятора (або JIT-компілятора), який, перед виконанням програми, перетворює вихідний код на JavaScript в байт-код певного формату [8].

Движок з відкритим кодом V8 був створений компанією Google, він написаний на C ++. Движок використовується в браузері Google Chrome. Крім того, що відрізняє V8 від інших движків, він застосовується в популярному серверному середовищі Node.js.

Усередині движка використовуються кілька потоків:

- головний потік, який займається тим, що від нього можна очікувати: читає вихідний JavaScript-код, компілює його і виконує;
- потік компіляції, який займається оптимізацією коду в той час, коли виконується головний потік;
- profiler-потік , який повідомляє системі про те, в яких методах програма витрачає найбільше часу, як результат, Crankshaft може ці методи оптимізувати;
- кілька потоків, які підтримують механізм збору сміття.

А також два компілятори:

- full-codegen – простий і дуже швидкий компілятор, який видає порівняно повільний машинний код;

- Crankshaft – більш складний JIT-компілятор для оптимізації, який генерує добре оптимізований код.

При першому виконанні JS-коду V8 використовує компілятор full-codegen, який безпосередньо, без будь-яких додаткових трансформацій, транслює розібраний їм JavaScript-код в машинний код. Це дозволяє дуже швидко приступити до виконання машинного коду. Зверніть увагу на те, що V8 не використовує проміжне представлення програми у вигляді байт-коду, таким чином, усуваючи необхідність в інтерпретаторі.

Після певного часу виконання коду, profiler-потік збере достатньо даних для того, щоб система могла зрозуміти, які методи потрібно оптимізувати.

Далі, в іншому потоці, починається оптимізація за допомогою Crankshaft. Він перетворює абстрактне синтаксичне дерево JavaScript в високорівневе уявлення, що використовує модель єдиного статичного присвоювання (static single-assignment, SSA) [8]. Це уявлення називається Hydrogen. Потім Crankshaft намагається оптимізувати граф потоку керування Hydrogen. Більшість оптимізацій виконується на цьому рівні.

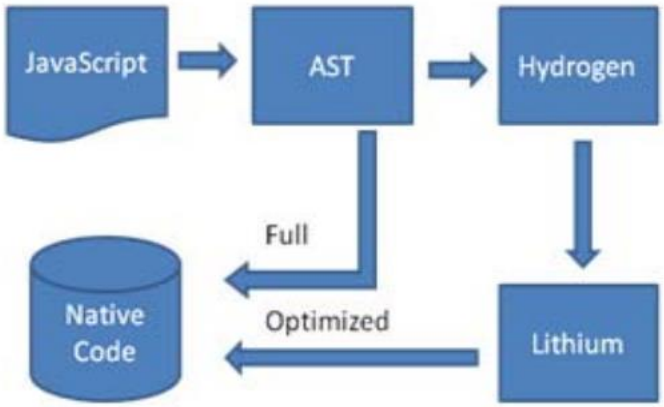

Рисунок 4.2 – Архітектура V8

Після усіх необхідних оптимізації графу Hydrogen, Crankshaft переводить його в низькорівневе уявлення, яке називається Lithium. Більшість реалізацій Lithium залежить від архітектури системи. На цьому рівні, наприклад, відбувається виділення регістрів.

В результаті Lithium-уявлення компілюється в машинний код. Потім відбувається те, що називається заміщенням в стеці (on-stack replacement, OSR). Перед компіляцією і оптимізацією методів, в яких програма витрачає багато часу, потрібно буде попрацювати з їх неоптимізованими варіантами. Потім, не перериваючи роботу, V8 трансформує контекст (стек, регістри) таким чином, щоб можна було перемкнутися на оптимізовану версію коду.

Це дуже складне завдання, з огляду на те, що крім інших оптимізацій, V8 спочатку виконує вбудовування коду. V8 – не єдиний движок, здатний це зробити.

Також движок надає захист від невдалої оптимізації – так звана деоптімізація. Вона спрямована на зворотну трансформацію, яка повертає систему до використання неоптимізованого коду в тому випадку, якщо припущення, зроблені движком і покладені в основу оптимізації, більше не відповідають дійсності.

Як і всі компілятори різних мов програмування, движок V8 має свій збирач сміття. Для збору сміття V8 використовує традиційний генеалогічний підхід «познач і викинь» (mark-and-sweep) для маркування та очищення попередніх поколінь коду. Фаза маркування передбачає зупинку виконання JavaScript. Для того, щоб контролювати навантаження на систему, створювану складальником сміття і зробити виконання коду більш стабільним, V8 використовує інкрементний алгоритм маркування: замість того, щоб обходити всю купу (“Heap”), він намагається помітити все, що зможе, обходячи лише частину купи. Потім нормальне виконання коду поновлюється. Наступний прохід збирача сміття по купі починається там, де закінчився попередній.

Це дозволяє домогтися дуже коротких пауз в ході звичайного виконання коду. Як вже було сказано, фазою очищення пам'яті займаються окремі потоки.

4.1.3 Бібліотека для створення графічного інтерфейсу

React (іноді React.js) – JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів. React розробляється і підтримується Facebook, Instagram і співтовариством окремих розробників і корпорацій. React може використовуватися для розробки односторінкових і мобільних додатків. Його мета – надати високу швидкість, простоту і масштабованість. Як бібліотеки для розробки призначених для користувача інтерфейсів React часто використовується з іншими бібліотеками, такими як MobX, Redux і GraphQL.

React може функціонувати як веб-бібліотека для розробки односторінкових додатків в реактивному стилі. Це прогресивна бібліотека для створення користувацьких інтерфейсів. Ядро React в першу чергу вирішує завдання рівня уявлення (view), що спрощує інтеграцію з іншими бібліотеками та існуючими проектами [13]. З іншого боку, React повністю підходить і для створення складних односторінкових додатків (SPA, Single-Page Applications), якщо використовувати його спільно з сучасними інструментами для створення складної логіки відображення інтерфейсу.

Основною зручністю веб-бібліотек, подібних React, є реактивність, тобто оновлення даних в реальному часі. Так само дана бібліотека дозволяє реалізувати архітектуру SPA, спростити створення веб-додатків і як підсумок поліпшити читабельність вихідного коду разом зі зменшенням витраченого на розробку часу.

React підходить для невеликих проектів, яким необхідно додати трохи реактивності, уявити форму за допомогою AJAX, відобразити значення при введенні даних користувачем, авторизація або інші аналогічні завдання.

React легко масштабується і добре підходить для об'ємних проектів, тому його називають однією з найліпших бібліотек для створення інтерфейсу користувача. React також відмінно підходить для великих односторінкових додатків завдяки своїм основним компонентам, таким як Router і Vuex.

Однією з основних переваг у швидкості є віртуальна об'єктна модель (“Virtual DOM” або “VDOM”). Віртуальний DOM (VDOM) – це концепція програмування, в якій ідеальне або «віртуальне» уявлення призначеного для користувача інтерфейсу зберігається в пам'яті і синхронізується з «реальним» DOM. Даний процес називається узгодженням.

Даний підхід забезпечується декларативним React API. Основним принципом роботи даного патерну є узгодження з движком React в якому стані повинен перебувати UI, і він буде гарантувати, що DOM відповідає цьому стану. Це звільнює програміста від необхідності прямих маніпуляцій з атрибутами, обробку подій і ручне оновлення DOM.

«Віртуальний DOM» вважається патерном, ніж конкретної технологією. У світі JavaScript-фреймворків та бібліотек термін «віртуальний DOM» зазвичай асоціюється з елементами React, Vue.js або Angular.js , оскільки вони є об'єктами, що представляють призначений для користувача інтерфейс.

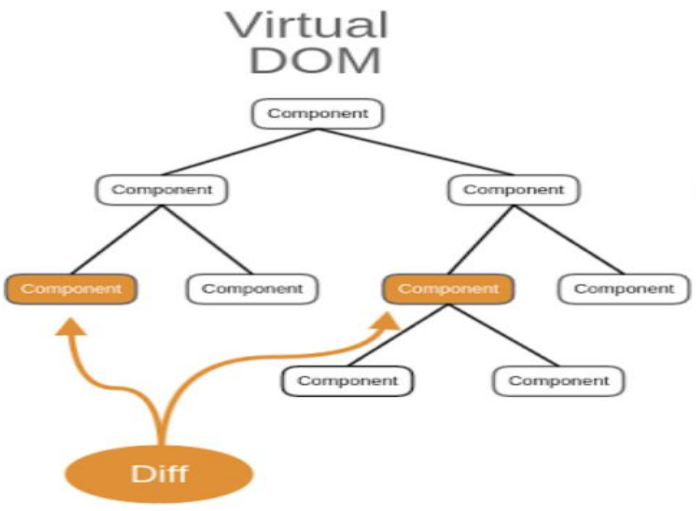

Рисунок 4.3 – Порівняння компонентів в VDOM

VDOM «стежить» за змінами між поточним елементом і елементом, який міг зазнати будь-яких змін, і якщо ці зміни були зроблені то даний компонент буде змінений, не порушуючи загальної структури HTML – сторінки додатка.

Приблизно таким чином відбуваються зміни сторінки додатка при використанні React.

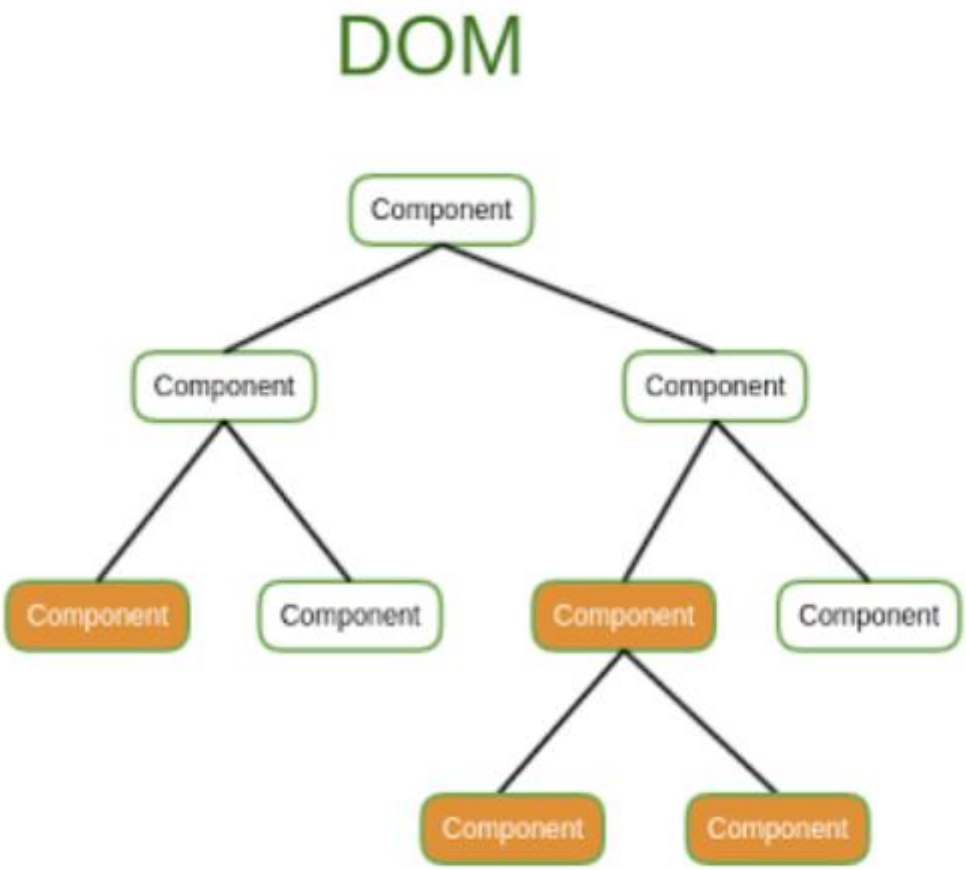

Рисунок 4.4 – Зміна елементів в VDOM після порівняння

Даний патерн проектування додатків дозволяє прискорити роботу програми за рахунок зменшення затраченого часу на отрисовку HTML – сторінки.

4.2 Використовувані технології розробки веб-серверу

4.2.1 Вибір мови програмування серверної частини

Для успішного виконання цього завдання сервер повинен мати:

- стабільну роботу при навантаженні високою кількістю даних;
- високу продуктивність.

Тому для написання серверної частини додатку було обрано використати язик програмування Scala. Scala – це мова програмування загального призначення, що забезпечує підтримку як об’єктно-орієнтованого програмування, так і функціонального програмування. Мова має потужну систему статичного типу. Багато дизайнерських рішень Scala спрямовані на вирішення проблем Java.

Вихідний код Scala призначений для компіляції в байт-код Java, щоб отриманий код працював на віртуальній машині Java. Scala забезпечує взаємодію мови з Java, так що бібліотеки, написані будь-якою мовою, можуть мати посилання безпосередньо в Scala або коді Java. Як і Java, Scala об'єктно-орієнтована і використовує синтаксис фігурних дужок, що нагадує мову програмування C. На відміну від Java, Scala має багато можливостей функціональних мов програмування, таких як Scheme, Standard ML та Haskell [11], включаючи каррінг(currying), незмінність, «ледаче виконання» («lazy evaluation») та узгодження шаблонів. Він також має вдосконалену систему типів, що підтримує алгебраїчні типи даних, коваріацію та контраваріацію, типи вищого порядку (але не типи вищого рангу) та анонімні типи. Інші особливості Scala, яких немає в Java, включають перевантаження оператора («overriding»), необов’язкові параметри, іменовані параметри та необроблені рядки. І навпаки, особливістю Java, яка не є в Scala, є перевірка винятків, що виявилось суперечливим [12].

Scala має таку ж саму модель компіляції, як Java і C #, а саме окреме компілювання та динамічне завантаження класу, так що код Scala може викликати бібліотеки Java.

Експлуатаційні характеристики Scala такі ж, як і у Java. Компілятор Scala генерує байт-код, який майже ідентичний коду, створеному компілятором Java. Насправді код Scala можна декомпілювати до читабельного коду Java, за винятком певних операцій конструктора. Для віртуальної машини Java (JVM) код Scala та код Java неможливо розпізнати. Єдина відмінність - одна додаткова бібліотека виконання, «scala-library.jar».

Scala додає велику кількість функцій порівняно з Java і має деякі принципові відмінності в основній моделі виразів і типів, що робить мову теоретично чистою та усуває деякі проблеми у Java.

Одним з основних плюсів Scala є «лінива оцінка» програмних виразів. За замовчуванням оцінювання виразів має властивість бути «строгим» («нетерплячим»). Іншими словами, Scala оцінює вирази, як тільки вони стають доступними, а не в міру необхідності. Однак можна оголосити змінну не вимогливою («ледачою») за допомогою ключового слова «lazy», що означає, що код для створення значення змінної не буде оцінюватися до першого звернення до змінної. Також існують нестрогі колекції різних типів (наприклад, тип Stream, нестрогий пов'язаний список), і будь-яку колекцію можна зробити нестрогою за допомогою методу уявлення. Нестрогі колекції забезпечують хорошу семантичну відповідність таким речам, як дані, створювані сервером, де оцінка коду для генерації більш пізніх елементів списку (що, в свою чергу, запускає запит до сервера, можливо, розташованому десь ще у сеті інтернеті) відбувається, коли елементи дійсно потрібні.

Як було описано вище, Scala є однією з мов JVM, і її найбільшими перевагами є підтримка як об'єктно-орієнтованого, так і функціонального програмування [16]. Обидва підходи до програмування спрямовані на створення читабельного коду без помилок, але вони використовують це по-

різному. Там, де об’єктно-орієнтоване програмування поєднує структури даних із діями, які ви хочете виконати над ними, функціональне програмування розділяє їх.

Кожен підхід має свої переваги. Для багатьох людей об’єктно-орієнтована парадигма має інтуїтивний сенс, і поєднання поведінки зі структурами даних, з якими вони будуть взаємодіяти, може полегшити з’ясування того, що відбувається в незнайомій кодовій базі. У той же час, переваги функціонального програмування для чітко відокремлених та незмінних структур даних та дискретної поведінки часто дозволяють зробити більше, використовуючи менше коду. Функціональне програмування спрямоване на використання лямбда-виразів. Сенс усіх лямбд – відкладене виконання. Зрештою, якби ви хотіли виконати якийсь код прямо зараз, ви б це зробили, не загортаючи його в лямбду.

Є багато причин для відкладеного запуску коду, наприклад:

- запуск коду в окремому потоці;
- запуск коду кілька разів;
- запуск коду в потрібній точці алгоритму (наприклад, операція порівняння при сортуванні);
- запуск коду, коли щось трапляється (надходження даних, сигнал від користувача тощо);
- запуск коду лише за необхідності.

Усі сучасні мови програмування мають динамічну типізацію, однак Scala перевіряє типи під час компіляції, що означає, що багато тривіальних, але дорогих помилок можна виявити під час компіляції, а не під час тестування. У той же час Scala має майже досконалу систему типів, що означає, що під час розробки гарантується безпека перевірки типу під час компіляції, для того щоб кожен раз не вказувати кожен тип [15].

Мова програмування Scala стисла. Кілька циклів можна замінити одним словом, що робить його значно менш багатослівним, ніж стандартна

Java. Крім того, його статично типовий та функціональний характер робить його безпечним для типу.

```
List<String> reversedValues = new ArrayList<String>();

  for (String n : nameList)

   {

     reversedValues.add(n.reverse());

   }

return reversedValues;
```

```
nameList.map(_.reverse)
```

Рисунок 4.5 – Порівняння однакового функціоналу на Java та Scala

Хоча Hadoop MapReduce може паралельно обробляти та генерувати великі масиви даних, його критикують за нездатність обробляти потокову обробку в режимі реального часу. Spark дає Scala перевагу над іншими мовами програмування для обробки потоків у режимі реального часу. Це зробило Scala одним з основних обчислювальних механізмів для швидкої обробки даних.

4.2.2 Scala у роботі з «великими даними»

Великі дані, машинне навчання, статистика, статистичне машинне навчання – такі терміни з’являються останнім часом в усьому світі. За даними сайту “Forbes”, за нинішніх темпів, щодня створюється 2,5 квінтильйона байтів даних із зростанням IOT. IOT – концепція обчислювальної мережі фізичних предметів («речей»), оснащених вбудованими технологіями для взаємодії один з одним або з зовнішнім середовищем, яка розглядає організацію таких мереж як явище, здатне перебудувати економічні та суспільні процеси, що виключає з частини дій і операцій необхідність участі людини.

На основі цих даних створюється ідея ведення нового бізнесу. Процес аналізу та вилучення інформації для отримання уявлення з цієї великої кількості структурованих, напівструктурованих, неструктурованих даних називається великими даними.

Тепер за допомогою штучного інтелекту та алгоритмів системи здатні автоматично навчатися та вдосконалюватися, щоб генерувати доцільну інформацію з усього потоку даних, без явного втручання людини або програмування на основі правил.

Щоденне збільшення кількості даних у мережі Інтернет породило абсолютно нові формати даних та бази даних великого масштабу. Це явище породило можливість аналізувати витяги та генерувати на основі статистичних даних пов’язати великі дані з машинним навчанням та статистичним аналізом.

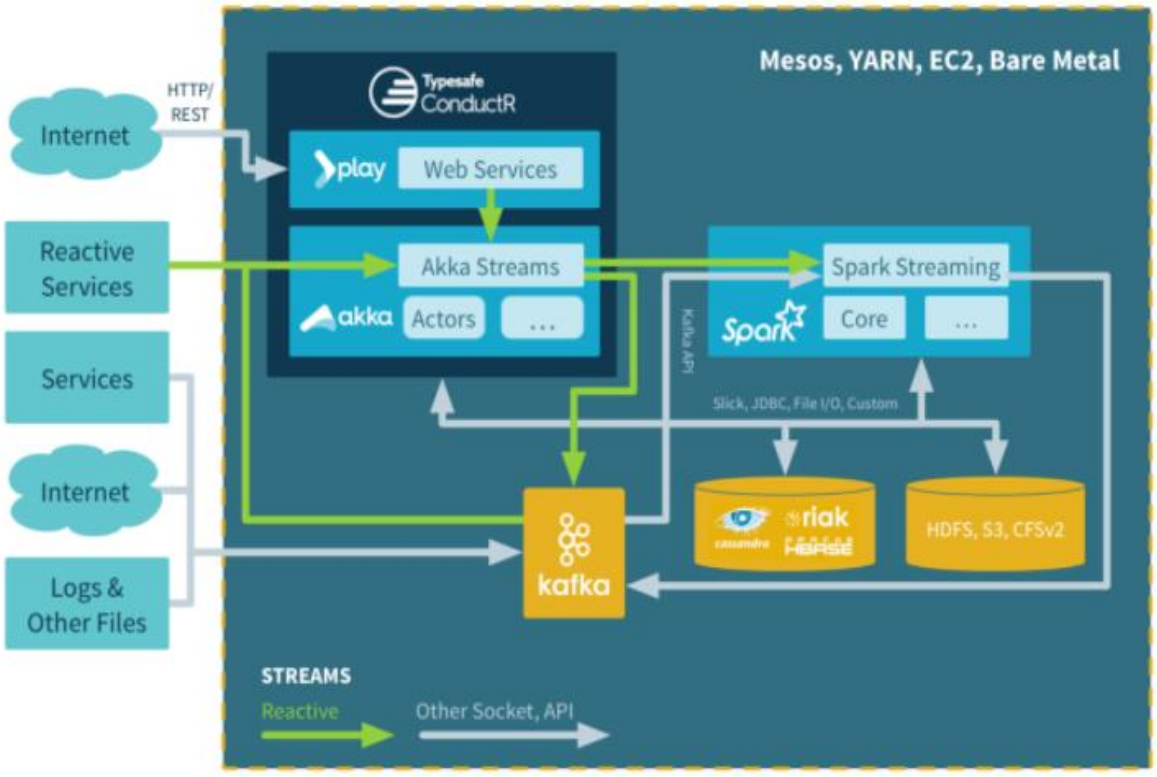

Рисунок 4.6 – Основні складові проекту для роботи з великими даними на Scala

Як правило, точність пошуку шаблонів даних або видобування даних або виявлення знань алгоритму машинного навчання залежить від обсягу даних, які алгоритм обробив.

Python та R – найвидатніші мови програмування для машинного навчання та наук про дані. Зараз Scala швидко набирає популярності через використання Apache Spark.

Зараз існує досить велика кількість бібліотек та фреймворків для роботи з великою кількістю даних, які написані на мові програмування Scala. З їх допомогою можна реалізувати методи для обробки інформації з елементами самоподібності. Однак ці бібліотеки мають різний функціонал та призначення:

- Apache Spark – це уніфікований аналітичний механізм для обробки великих даних з найбільшою кількістю функцій, таких як SQL та DataFrames, MLlib для машинного навчання, GraphX та Spark Streaming. Apache Spark, побудований на Scala, отримав велике визнання і широко використовується у виробництвах [4]. Еластичні розподілені набори даних (“RDD”) є основною структурою даних Spark. Незмінність, розподілення, ледача оцінка є його загальними властивостями;
- Apache Kafka – розподілена потокова платформа для обробки потоків даних у режимі реального часу. Дана платформа написана на мовах Java та Scala. Kafka – це швидка, масштабована, довговічна та відмовостійка система обміну повідомленнями для публікації та підписки. Вона працює в поєднанні з Apache Storm, Apache HBase та Apache Spark для аналізу в режимі реального часу та рендерингу потокових даних;
- Apache Samza – фреймворк обробки потоків, розроблений на Scala. Apache Samza використовує Apache Kafka для обміну повідомленнями та Apache Hadoop YARN для забезпечення відмовостійкості, ізоляції процесора, безпеки та управління ресурсами. Даний фреймворк схожий на Apache Storm, хоча простіший в експлуатації. Робота обробки потоків Samza була також написана на Scala [4];

- Apache scalding – це API Scala для технології Cascading, бібліотеки Java, яка в свою чергу є абстракцією для Hadoop MapReduce. Scalding спрощує написання завдань MapReduce у Scala;

- Apache Flink – фреймворк для розподіленої потокової та пакетної обробки даних. Ядро Flink – це гібридний (Real-Time Streaming та Batch) розподілений механізм обробки даних, написаний на Java та Scala. Flink містить кілька API для пакетної обробки (DataSet API), потокового передавання в реальному часі (DataStream API) та реляційних запитів (API таблиці), а також бібліотеки для машинного навчання (FlinkML – чиста Scala), обробки складних подій (CEP) та обробка графіка (Gelly).

4.2.3 Опис фреймворку Apache Spark

Apache Spark – це Big Data фреймворк з відкритим кодом для розподіленої пакетної і потокової обробки неструктурованих і слабоструктурованих даних, що входить в екосистему проектів Apache.

Даний фреймворк складається з наступних компонентів:

- ядро (Core);
- SQL – інструмент для аналітичної обробки даних за допомогою SQL-запитів;
- Streaming – надбудова для обробки потокових даних;
- MLlib – набір бібліотек машинного навчання;
- GraphX – модуль розподіленої обробки графів.

Spark може працювати як в середовищі кластера Hadoop під керуванням YARN, так і без компонентів ядра Hadoop, наприклад, на базі системи управління кластером Mesos. Спарк підтримує кілька популярних розподілених систем зберігання даних (HDFS, OpenStack Swift, Cassandra, Amazon S3) і мов програмування (Java, Scala, Python, R), надаючи для них API-інтерфейси [5].

Варто відзначити, що Spark Streaming, на відміну від, наприклад, Apache Storm, Flink або Samza, і не обробляє потоки Big Data цілком. Замість цього реалізується мікропакетний підхід (micro-batch), коли потік даних розбивається на невеликі пакети тимчасових інтервалів. Абстракція Spark для потоку називається DStream (discretized stream, дискретизований потік) і являє собою мікро-пакет, що містить кілька відмовостійких розподілених датасета, RDD (resilient distributed dataset) [4].

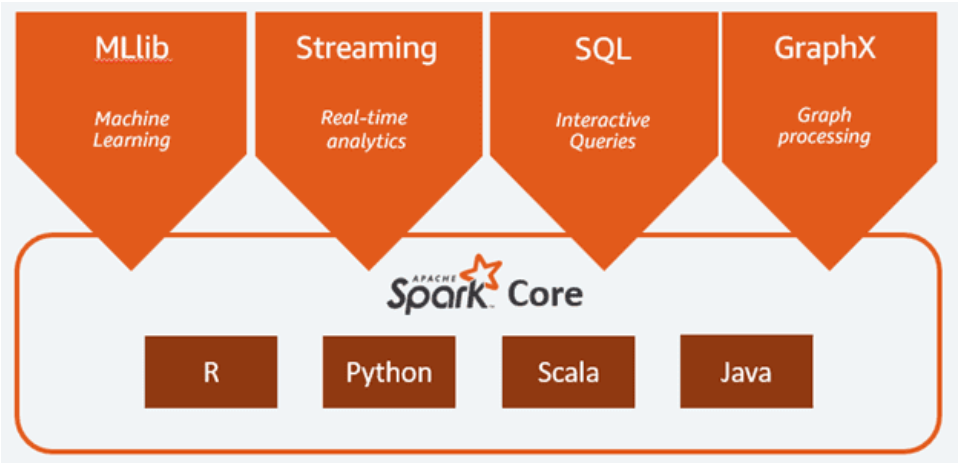

Рисунок 4.7 – Компоненти Apache Spark

Саме RDD є основним обчислювальним примітивом Apache Spark, над яким можна робити паралельні обчислення і перетворення за допомогою вбудованих і довільних функцій, в тому числі за допомогою тимчасових вікон (window-based operations) [4]. Детальніше про тимчасові вікна ми розповідали тут на прикладі Apache Kafka Streams.

Spark був створений для того, щоб допомагати вирішувати широке коло завдань з аналізу даних, починаючи з простого завантаження даних і SQL-запитів і закінчуючи машинним навчанням і потоковими обчисленнями, за допомогою одного і того ж обчислювального інструменту з незмінним набором API. Головний інсайт цієї програмної багатозадачності в тому, що завдання з аналізу даних в реальному світі – будь вони інтерактивної аналітикою в такому інструменті, як Jupyter Notebook, або ж звичайним

програмуванням для випуску додатків – мають тенденцію вимагати поєднання безлічі різних типів обробки і бібліотек. Цілісна природа Spark робить рішення цих завдань простіше й ефективніше. Наприклад, якщо кінцевий користувач завантажуєте дані за допомогою SQL-запиту і потім оцінюєте модель машинного навчання за допомогою бібліотеки Spark ML, движок може об'єднати всі ці кроки в один прохід за даними. Більш того, для дослідників даних може бути вигідно застосовувати об'єднаний набір бібліотек (наприклад, Python або R) при моделюванні, а веб-розробникам знадобляться уніфіковані фреймворки, такі як Node.js або Django.

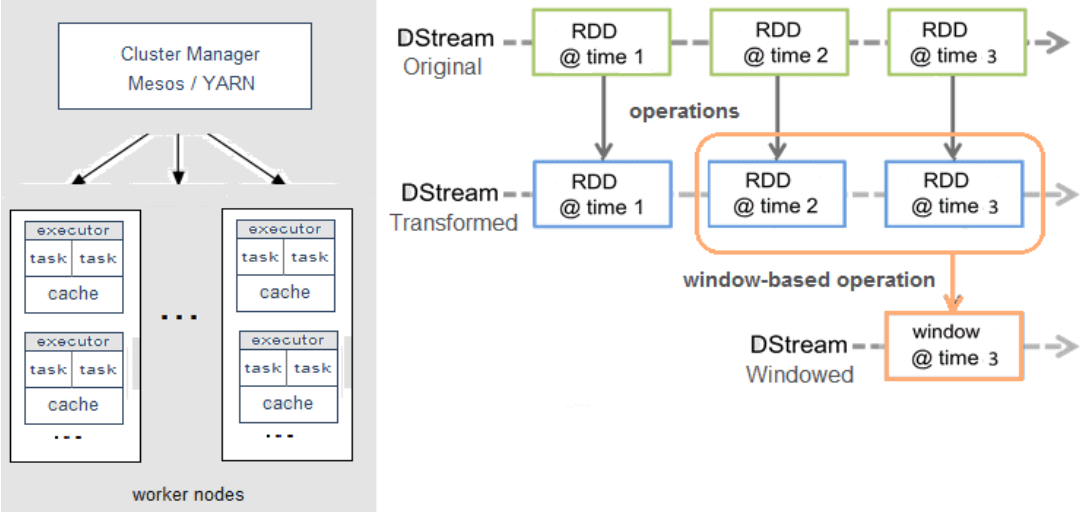

Рисунок 4.8 – Принцип роботи Apache Spark

Spark керує завантаженням даних з систем зберігання і виробляє обчислення над ними, але сам не є кінцевим постійним сховищем. Spark використовують, коли мають справу з широким розмаїттям постійних систем зберігання, включаючи системи хмарного типу за прикладом Azure Storage і Amazon S3, розподілені файлові системи, такі як Apache Hadoop, простору для зберігання ключів, як Apache Cassandra, і послідовностей повідомлень, як Apache Kafka [6]. Однак, у Spark не зберігаються дані надовго і не підтримує жодну з цих систем. Головною причиною цього є те, що більшість даних вже знаходиться в декількох системах зберігання. Переміщення даних

використовує багато ресурсів системи, тому Spark тільки обробляє дані за допомогою обчислювальних операцій, і не має значення, де вони при цьому знаходяться.

Сфокусованість Spark на обчисленнях відрізняє його від більш ранніх програмних платформ по обробці великих даних, наприклад від Apache Hadoop. Це ПО включає в себе і систему зберігання (HFS, зроблену для недорогих сховищ на кластерах продуктових серверів Defining Spark 4) і обчислювальну систему (MapReduce) [6]. Між собою вони інтегруються досить добре. Однак це важко реалізувати за участю тільки однієї частини без застосування другої або, що важливіше, написати додатки, які мають доступ до даних, що зберігаються в іншому місті. Spark також широко застосовується зараз в середовищах, де в архітектурі Hadoop немає великого сенсу. Наприклад, на публічному хмарному сховищі або в потокових додатках.

4.2.4 Опис інструментів для веб-розробки Akka

Akka – це безкоштовний фреймворк із відкритим кодом та алгоритмом виконання, що спрощує побудову одночасних та розподілених програм на JVM. Akka підтримує кілька моделей програмування для паралельності, але вона підкреслює паралельність на основі акторів, які були створені під натхненням Erlang. Він надає готову середу для побудови високонавантажених систем, які ефективно використовують ресурси машини.

Фреймворк Akka має наступний функціонал:

- кластеризація, яка реалізована безпосередньо у самому фреймворку;
- опис логіки потокової обробки даних;
- побудова відмовостійких реактивних систем;

Також за допомогою Akka можна простіше реалізувати реактивний веб-додаток.

Щоб називатися реактивними, додатки повинні бути чуйними, пружними, еластичними і спілкуватися асинхронним обміном повідомлень:

- чуйність – це швидка реакція на заданий запит. Тобто програма має завжди і завжди обробляти і відповідати на запити швидко, навіть при великих навантаженнях. Це також означає, що якщо десь стався збій, система повинна швидко обробити ситуацію і дати відповідну відповідь;

- гнучкість – це означає, що додаток повинен залишатися чуйним навіть при виникненні відмови в деякій частині системи. Для цього відмова одного компонента обробляється іншим, який в свою чергу теж може делегувати обробку своєї відмови іншого компонента. Таким чином, відмова одного компонента в системі не призводить до відмови всієї системи;

- еластичність – говорить про те, що такі додатки можуть масштабироваться по потребі, що призводить до архітектури, в якій не залишається вузьких місць і підвищується загальна чуйність системи цілком;

- асинхронний обмін повідомленнями для комунікації між компонентами системи – це зменшує залежність компонентів системи один від одного, дозволяє масштабувати їх незалежно і розгортати в різних місцях;

У підсумку великі системи будуються з менших систем, підтримуючи їх реактивні властивості в собі. Такі додатки і називаються реактивними.

Є моделі програмування, які отримують користь із кластера серверів, перетворюючи їх в один суперкомп'ютер. Одна з таких моделей програмування – модель акторів, про яку ми і будемо говорити.

Актор – це обчислювальна одиниця. У відповідь на повідомлення, яке він отримав, актор може:

- відправити певну кількість повідомлень іншим акторам;
- створити певну кількість нових акторів;
- визначити поведінку для обробки наступного повідомлення, яке він отримає.

Немає певної послідовності перерахованих вище дій, вони можуть бути виконані паралельно. Філософія моделі акторів говорить, що «все навколо

актори». При цьому об'єктно-орієнтовані програми зазвичай виконуються секвентально, коли актори працюють паралельно. Звичайно, ви можете використовувати потоки, щоб додати конкурентність в об'єктно-орієнтовані програми. В 64-бітної системі потік Java займає 1 мегабайт пам'яті, коли актор займає до 300 байтів. Потоки будуть обмежені в кількості, і доведеться користуватися специфікою об'єктно-орієнтованого підходу. Використовуючи модель акторів, розробник думає про акторів замість об'єктів і потоків. Можна мати набагато більше акторів, ніж потоків.

Актори інкапсулюють стан і поведінку і спілкуються виключно за допомогою обміну повідомленнями. Повідомлення, отримані актором, лягають в його поштову скриньку. Акторів краще розглядати як людей. Моделюючи рішення з моделлю акторів, можна уявити групу людей, яким призначаються підзадачі. Підзадачі – це результат ділення однієї великої завдання на дрібні шматки, які делегуються акторам для виконання. Ви можете призначити ці завдання акторам, організовуючи їх у структуру, немов членів команди в економічній організації.

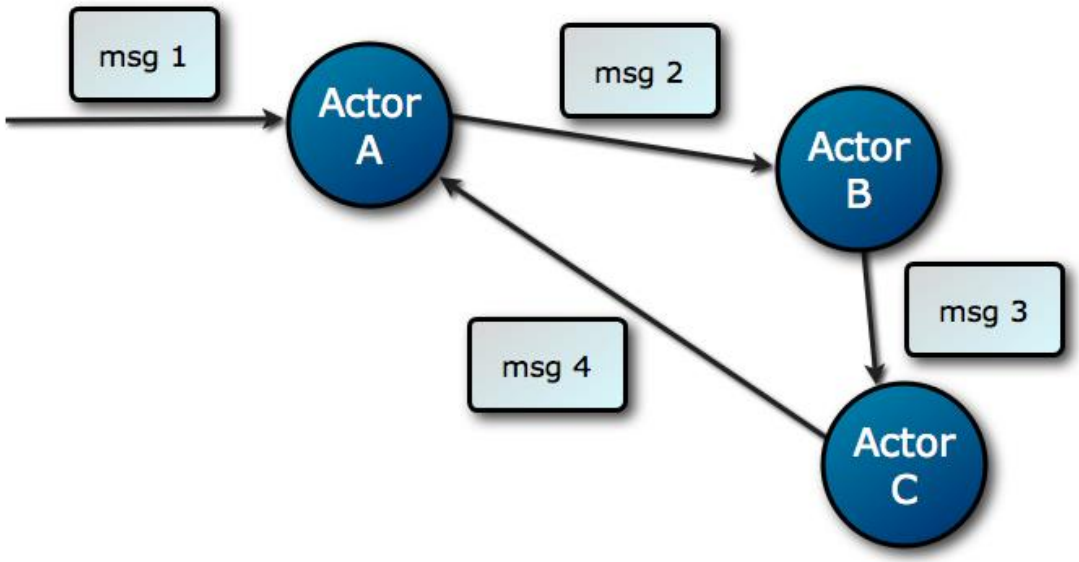

Рисунок 4.10 – Приклад роботи акторів

Кожен актор може створювати дочірні акторів і бути їх супервізором. У практиці краще буде розділити задачу актора на дрібні керовані підзадачі і дозволити підлеглим виконати їх в паралелі. Якщо актор має важливий стан, який не можна втратити (наприклад, отримати помилку виконання), тоді всю задачу цілком краще віддати на виконання підлеглим і просто стежити за процесом виконання. Тому актори ніколи не використовуються поодинці. Вони працюють в системах і утворюють ієрархії (як файли в UNIX). Системи акторів також можуть формувати кластери. Якщо уявити актора як зірку, тоді система акторів – це кластер зірок. Кластери систем акторів – це кластери кластерів зірок. Таким чином, за допомогою акторів можна створити велику систему кластерів. Ця система після може бути розгорнута на наданому обладнанні і виконувати запрограмовану логіку.

Система акторів працює по принципу «Let It Crash» («дозволь йому відмовити») і так званого шаблоном Error Kernel Pattern (шаблон ядра помилок). Цей шаблон веде нас до системи акторів, де супервизор кожного рівня ієрархії у відповіді за локальні помилки. В даному контексті, помилки – це “exceptions”, що виникли у підлеглих акторів, тобто самі актори не повинні їх обробляти. У свою чергу, супервізор при виникненні помилки може відловити її і, як правило, продовжити, перезавантажити, зупинити роботу актора або ж надати “exception” своєму супервізору, повідомляючи про падіння всієї своєї ієрархії підпорядкування. Ядро помилок самої системи акторів формується з системних акторів, які відповідають за життєвий цикл всієї системи. При цьому кожен рівень ієрархії в організаційній структурі є локальним ядром помилок, які відповідають за стратегію обробки помилок, яка визначить, як потрібно реагувати на певні помилки дочірніх акторів.

4.3 Реалізація методу первинної обробки інформації

4.3.1 Аналіз інформації для обробки

Для первинної обробки було обрано взяти дані у форматі CSV з Міністерства охорони здоров’я і соціальних служб США щодо прогнозування кількості зайнятих ліжок з хворими на COVID-19 у різних штатах:

- state – двозначний код штату;
- collection_date – приблизна дата;
- inpatient_beds_occupied – прогнозована кількість зайнятих ліжок для даної дати та часу;
- count_ll – прогнозована кількість зайнятих ліжок, нижня межа;
- count_ul – прогнозована кількість зайнятих ліжок, верхня межа;
- percentage_of_inpatient_beds_occupied_estimated – прогнозований процент зайнятих ліжок для даної дати та часу;
- percentage_ll – прогнозований процент зайнятих ліжок, нижня межа;
- percentage_ul – прогнозований процент зайнятих ліжок, верхня межа;

По-перше необхідно проаналізувати усі поля даних і знайти два ключових поля для аналізу: визначення часу та поле для першої ітерації розподілення. Після проведення очного аналізу було встановлено, що полем для першої ітерації розподілення буде поле зі строковим значенням кожного штату. Полем для визначення часу стало поле “collection_date” (алгоритм дозволяє тільки одне поле із типом “date”).

4.3.2 Опис роботи інтерфейсу користувача

Як було зазначено вище, інтерфейс користувача повинен використовувати мову програмування JavaScript та бібліотеку для створення інтерфейсів React.

Даний інтерфейс повинен виконувати задачу зі створення мета-даних для CSV файлів з інформацією для обробки. Спочатку користувачу необхідно розмістити файл з даними до своєї системи (якщо його ще там нема). За допомогою вбудованих функцій браузеру, завантажити необхідний файл до форми для передачі даних в розробленому інтерфейсі користувача. Однак інтерфейс заблокує спробу завантаження файлу до розробленого веб-серверу без виконання наступного необхідного етапу в алгоритмі роботи розробленого інтерфейсу.

Наступним етапом алгоритму роботи розробленого інтерфейсу є опис полів даних, які знаходяться у файлі (див рис 4.12). Наприклад, якщо користувач хоче завантажити файл, в якому є поле містить цілі числа, то користувачу необхідно записати ім’я цього поля та описати формат даних для цілого числа (“integer”).

Інтерфейс користувача може приймати наступні формати даних для полів даних:

- “integer” – формат даних для цілих чисел;
- “float” – формат даних для чисел з плаваючою комою;
- “boolean” – логічний тип даних (“true” або “false”);
- “date” – формат даних для дат;
- “string” – строковий формат даних.

Подібний опис даних дозволить веб-серверу правильно реалізувати логіку первинної обробки даних. У разі, якщо необхідні поля не були заповнені або були заповнені невірно, інтерфейс сповістить користувача про помилку в описі полів даних та заблокує відправку усіх даних до веб-серверу. Також інтерфейс має опцію додавання полів даних, якщо у файлі знаходиться більше ніж 6 полів з даними. Після заповнення усіх необхідних полів даних необхідно задати значення, по якому алгоритм кластеризації буде вираховувати наближення до того чи іншого кластеру. Наприклад, якщо обробляти даних поїздок таксі, то таким значенням може бути вектор напрямку пересування від стартової точки таксі до точки кінця маршруту.

У нашому випадку буде використовуватися середнє значення між нижньою межею процентів зайнятих ліжок по штату (percentage_ll) та верхньою межею (percentage_ll). Дане значення буде використовуватися як додаткове поле “features” в оброблюваних даних.

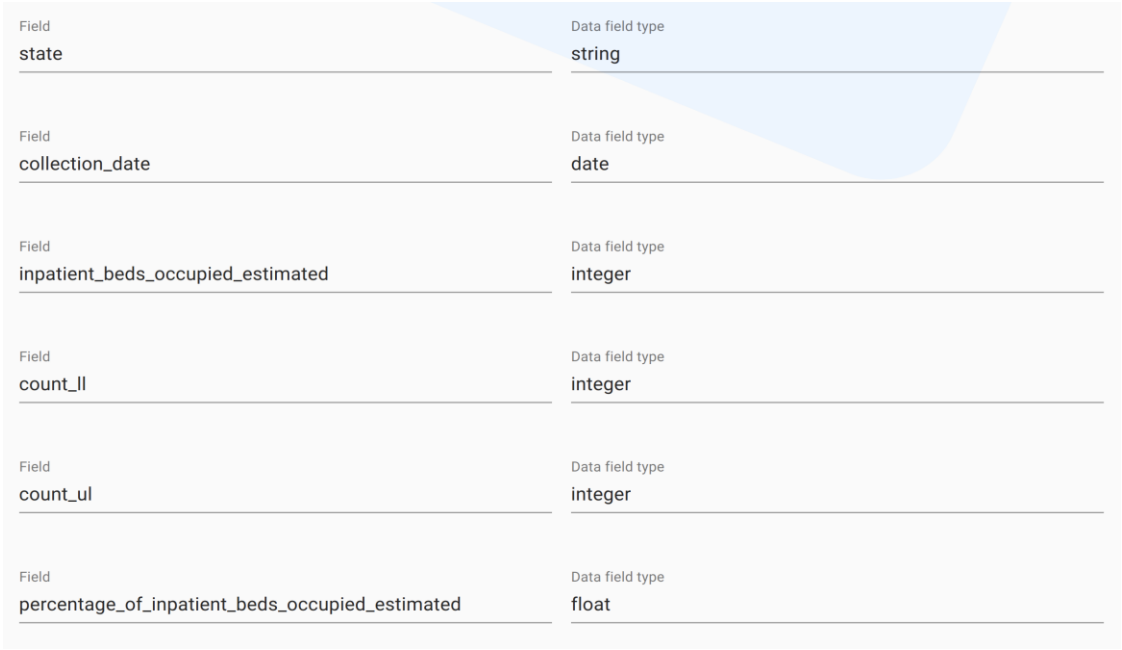

Рисунок 4.12 – Приклад заповнення опису полів даних

Якщо усі дані пройшли валідацію на стороні клієнта, вони відправляються на сервер для подальшої обробки.

4.3.3 Опис обробки даних на сервері

Для того щоб обробити необхідну інформацію для цього необхідно спочатку привести її до вигляду, з яким можуть працювати обрані інструменти розробки.

Оскільки формат даних CSV є текстовим форматом даних, його можна перетворити у формат, з яким може працювати Apache Spark. В даному фреймворку є інструмент для створення інструкції до внутрішньої структури даних, він називається “StructType”.

```
def createDataStructure: StructType = {

  val dataScheme = StructType(
    StructField("state", StringType, false) ::
      StructField("collection_date", DateType, false) ::
      StructField("inpatient_beds_occupied_estimated", IntegerType, false) ::
      StructField("count_ll", IntegerType, false) ::
      StructField("count_ul", IntegerType, false) ::
      StructField("percentage_of_inpatient_beds_occupied_estimated", FloatType, false) ::
      StructField("percentage_ll", IntegerType, false) ::
      StructField("percentage_ul", IntegerType, false) ::
      StructField("total_inpatient_beds", IntegerType, false) ::
      StructField("total_ll", IntegerType, false) ::
      StructField("total_ul", IntegerType, false) :: Nil
  )
  dataScheme
}
```

Рисунок 4.13 – Використання StructType для інструкції перетворення даних

Після цього дану інструкцію необхідно додати до методу для зчитування відправлених даних.

Усі відправлені дані зберігаються у системі директорій на серверній машині, у нашому прикладу машина буде запускатися на операційній системі Ubuntu 20.04. Тому необхідно за допомогою інструментів Apache Spark створити систему директорій, по якій алгоритм зможе з легкістю знаходити оброблені дані для кожного користувача.

Директорія, в якій будуть зберігатися оброблені файли складається з наступних частин:

- datasets_v1 – початкова директорія, у якій будуть зберігатися директорії користувачів;

- user_id – директорія, яка співпадає з ідентифікатором користувача при створені облікового запису;

- inbeds_data_1606611904000 – назва проекту, та дата у мілісекундах від 1970-го року.

```
def readFile(userId: String, timestamp: String, processPath: String): Unit = {
  val spark = SparkSession
    .builder
    .master( master = "local[*]")
    .appName( name = "Self-similarity data initial processing")
    .getOrCreate()
  println(s"Reading: ${processPath}/${userId}_${timestamp}_data.csv")

  val processDf = spark.read.format( source = "csv")
    .option("header", value = true)
    .option("delimiter", ",")
    .option("mode", "DROPMALFORMED")
    .schema(createDataStructure)
    .load( path = s"${processPath}/${userId}_${timestamp}_data.csv")
    .cache()
```

Рисунок 4.14 – Зчитування файлу з директорії користувача

Після створення усіх необхідних директорій починається наступний етап алгоритму, який передбачає собою перетворення даних на так звані фрейми даних (“Data Frames”) або набори даних (“Datasets”).

DataFrame – це набір даних, організований у іменовані стовпці. Концептуально це еквівалентно таблиці в реляційній базі даних або кадру даних у R / Python, але з більш багатими оптимізаціями. Фрейми даних можуть бути побудовані з широкого спектру джерел, таких як: структуровані файли даних, таблиці у вулику, зовнішні бази даних або існуючі RDD. API DataFrame доступний у Scala, Java, Python та R. У Scala та Java DataFrame представлений набором даних рядків. В API Scala DataFrame – це просто псевдонім типу Dataset [Row].

Одним із варіантів використання Spark SQL є виконання SQL-запитів. Spark SQL також може використовуватися для зчитування даних із існуючої інсталяції Hive. При запуску SQL з іншої мови програмування результати повертаються як набір даних (“Dataset”) або фрейм даних. Розробник також можете взаємодіяти з інтерфейсом SQL за допомогою командного рядка або через JDBC / ODBC.

Набір даних – це розподілена колекція даних. Набір даних є новим інтерфейсом, доданим у Spark 1.6, який забезпечує переваги RDD (строга

типізація, можливість використання потужних лямбда-функцій) з перевагами оптимізованого механізму виконання Spark SQL. Набір даних може бути побудований з об'єктів JVM, а потім маніпулювати за допомогою функціональних перетворень (map, flatMap, filter тощо). API набору даних доступне у Scala і Java. Python не підтримує API набору даних. Але завдяки динамічній природі Python, багато переваг API набору даних уже доступні (тобто розробник можете отримати доступ до поля рядка за назвою).

У даному випадку будуть використовуватися фрейми даних, тому за допомогою інструкції для перетворення даних, яка була описана вище, алгоритм перетворює необхідні дані на фрейми даних. Після цього алгоритму необхідно перевірити чи правильно були перетворені дані, тому за допомогою внутрішніх функцій Scala для перетворення одного типу даних на інший, проводиться перевірка на можливість приведення створених даних до потрібного нам формату, і якщо ні, то відловити помилку та повернути стандартне значення замість старого. Даний процес називається валідацією.

У якості стандартних використовуються наступні значення:

- для цілих чисел використовується «0»;
- для чисел з плаваючою комою «0.0»;
- для булевих значень “false”;
- для дати використовується дата створення запиту на обробку;
- для строкових значень використовується порожня строка.

За допомогою таких перетворень усі невірні значення потім будуть відведені алгоритмом в окрему групу, яку можна знайти у результатах, як дані з полем “has_errors”.

Після проходження всіх етапів перетворень даних, алгоритм почне виконувати основний метод обробки даних. У даному методі виділяються наступні основні етапи його виконання:

- перша ітерація розподілення інформації. Як було описано вище користувач може задати одне поле основного розподілення, яке не використовує методи кластеризації. У нашому випадку таким поле стане

строкове значення штату (state);

- на другій ітерації до кожного розподіленого блоку даних застосовується метод кластеризації k-means з заданою кількістю кластерів K=6. Після виконання кластеризації усі кластери відокремлюються у свою групу. На програмному рівні це означає, що до даних додається ще одне додаткове поле під назвою “group_prediction_id”, яке містить у собі номер кластеру, до якого була віднесена дана строка даних CSV;

- після завершення другого етапу починається виконання етапу розділення сформованих даних по різним кластерам в залежності від їх значення поля “group_prediction_id”.

Це зроблено для того, щоб весь алгоритм не зупинив свою роботу через одне або декілька неправильно приведених полів.

```
case class Validate(field: String) {
  def integer: Int = try { field.toInt } catch { case exception: Exception => 0 }
  def double = try { field.toDouble } catch  { case exception: Exception => 0.0 }
  def boolean = try { field.toBoolean } catch  { case exception: Exception => false }
  def date: Serializable with Comparable[_ >: ... <: ...] = try { Date.valueOf(field)} catch { case exception: Exception => LocalDate.now }
}

object Validations {
  def validateInteger(field: String): Int = Validate(field).integer
  def validateDouble(field: String) = Validate(field).double
  def validateBoolean(field: String) = Validate(field).boolean
  def validateDate(field: String): Serializable with Comparable[_ >: ... <: ...] = Validate(field).date
}
```

Рисунок 4.15 – Валідація даних

По завершенню усіх етапів основного алгоритму, ми отримаємо оброблені дані, які потім можна використовувати для подальшого статистичного аналізу або для машинного навчання. Усі сформовані дані необхідно передати до кінцевого користувача. Дані можна передати у вигляді графічного інтерфейсу у браузері або як посилання для завантаження користувачем оброблених даних у вигляді архіву даних.

ВИСНОВКИ

У рамках даної випускної атестаційної роботи був проведений аналіз предметної області, були висунуті певні вимоги, а також поставлено мету виконання.

Для вирішення цієї проблеми був проведений аналіз технологій та найоптимальніших технологій для даного рішення. За допомогою новітніх технологій веб-розробки (JavaScript, Scala, Apache Spark, Akka) був створений концепт роботи веб-додатку зі зручним функціоналом та користувацьким веб-інтерфейсом, який використовує модифікований метод кластеризації даних.

Дані методи і моделі первинної обробки даних дозволять різним аналітикам та вченим отримувати результати обробки даних у такому вигляді, який потім зручно використовувати для машинного навчання та статистичного аналізу.

Також у ході розробки методу первинної обробки даних на основі веб-додатку, який використовує метод k-means з розподіленням, був зроблений аналіз покращення роботи даного методу. Приведені методи і моделі можуть бути масштабовані без великих затрат часу та ресурсів. Оскільки кожний тип даних може бути обробленим по-різному, необхідно також додати й інші методи кластеризації до основного алгоритму виконання розробки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Горелик Г. Размерность пространства М.: Изд-во Моск.ун-на – 1983

2. Barabasi A. Stanly H. Fractal Concept in Surface Growth, N.Y.: Cambridge Univ Press – 1995

3. Федер Е. Фракталы, М.: Мир – 1991

4. Х. Кара, Е. Конвінскі, П. Венделл, М. Захарія. Вивчаємо Spark. Блискавичний аналіз даних = Learning Spark: Lightning-Fast Big Data Analytics (O'Reilly, 2015). – ДМК Пресс, 2015. – 304 с. – ISBN 978-5-97060-323-9.

5. С. Різа, У. Лезерсон, Ш. Оуен, Д. Уиллс. Spark для професіоналів: сучасні патерни обробки великих даних – Advanced Analytics with Spark. Patterns for Learning from Data at Scale (O'Reilly, 2015). – Пітер, 2017. – 272 с. – ISBN 978-5-496-02401-3.

6. Уоррен Р., Кара Х. Ефективний Spark. Масштабування і оптимізація = High Performance Spark. Best Practices for Scaling and Optimizing Apache Spark. – Пітер, 2018. - 352 с. – ISBN 978-5-4461-0705-6.

7. Девід Фленаган. 15.4.6. Незалежні від мови DOM-інтерфейси // JavaScript. Детальний керівництво – JavaScript. The Definite Guide / Переклад А. Кисельова. – 5-е изд. - СПб .: «Символ-Плюс», 2008. - С. 332-334.

8. JavaScript. Біблія користувача – JavaScript. Bible / Денні Гудман (Danny Goodman), Майкл Моррісон (Michael Morrison); пер. з англ. І. В. Василенко. – 5-е изд. – Москва, Санкт-Петербург, Київ: Ділектіка, 2006. - С. 3, 26. - 1184 с.

9. Е. Фрімен, Е. Фрімен. Вивчаємо HTML, XHTML і CSS – Head First HTML with CSS & XHTML. – П .: «Питер», 2010. - 656 с.

10. Стівен Шафер. HTML, XHTML і CSS. Біблія користувача, 5-е видання – HTML, XHTML, and CSS Bible, 5th Edition. – М .: «Диалектика», 2010. - 656 с.

11. Joshua D. Suereth. Scala in Depth (unspecified). – Manning Publications (English) Russian .. – S. 225. - ISBN 978-1-935182-70-2.

12. Gregory Meredith. Monadic Design Patterns for the Web (unspecified). – 1st. - 2011 .-- S. 300.

13. Мардан Азат. React швидко. Веб-додатки на React, JSX, Redux і GraphQL. – СПб .: «Пітер», 2019. – С. 560. – ISBN 978-5-4461-0952-4.

14. Бенкс Алекс, Порселло Єва. GraphQL: мова запитів для сучасних веб-додатків. – СПб .: «Пітер», 2019. – С. 240. – ISBN 978-5-4461-1143-5.

15. Прокопець А. Конкурентне програмування на SCALA. – ДМК прес, 2017. - 342 с. - ISBN 978-5-97060-572-1.

16. Scala в прикладах, переклад керівництва Мартін Одерски в Вікіпідручник

17. C. Traina, A. J. M. Traina, C. Faloutsos, “Показник відстані: нова концепція оцінки селективності в метричних деревах” Університет Карнегі Меллона, Пітсбург, Пенсільванія CMU-CS-99-110, 1 березня 1999 р.

18. К. Фалуцос та К.-І. Лін, “FastMap: Швидкий алгоритм індексації, видобування даних та візуалізації традиційних та мультимедійних наборів даних”, у ACM SIGMOD Intl. Конф. з управління даними, Сан-Хосе, Каліфорнія, 1995.

19. Мандель И. Д. Кластерный анализ. — М.: Финансы и статистика, 1988. — 176 с.

20. Хайдуков Д. С. Применение кластерного анализа в государственном управлении// Философия математики: актуальные проблемы. — М.: МАКС Пресс, 2009. — 287 с.

21. Завизіступ Ю. Ю. Метод обработки и классификации ГРВ подобных изображений / Ю. Ю. Завизиступ, О .Ф. Михаль, А. С. Свиридов // Тези доповідей четвертої міжнародної науково-технічної конференції «Проблеми інформатизації», Черкаси – Баку – Бєльсько-Бяла – Полтава, 3-4 листопада 2016 р. – С. 24.