МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА БІБЛІОТЕКИ ДЛЯ ОПТИМІЗАЦІЇ ОБРОБКИ ВЕЛИКИХ ДАНИХ НА ПЛАТФОРМІ APACHE SPARK МОВОЮ JAVA»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
К.П. Чорний
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Горбенко В.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Чорному Кирилу Павловичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка бібліотеки для оптимізації обробки великих даних на платформі Apache Spark мовою Java

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н, доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка бібліотеки, яка допоможе інженерам даних в їх дослідженнях.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	18.03.2024	
2.	Збір вихідних даних.	22.04.2024	
3.	Обробка методичних та теоретичних	03.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	26.08.2024	
5.	Розробка третього розділу.	11.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	13.12.2024	

Студент _____ К.П. Чорний
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка бібліотеки для оптимізації обробки великих даних на платформі Apache Spark мовою Java»: 36 с., 7 рис., 9 джерел, 1 додаток.

ІНЖЕНЕРІЯ ДАНИХ, ПАРАЛЕЛЬНЕ ОБЧИСЛЕННЯ, ПРОГРАМНА БІБЛІОТЕКА, APACHE SPARK, JAVA.

Об’єкт дослідження – способи обробки даних, двигуни обробки даних, Apache Spark.

Мета роботи: дослідження сучасних способів обробки даних, дослідження двигунів обробки даних, розробка бібліотеки для оптимізації обробки даних, дослідження Apache Spark.

У кваліфікаційній роботі показано виклики сучасних задач обробки даних. Проаналізовано засоби пришвидшення обробки даних. Розроблено бібліотеку оптимізації обробки великих даних на платформі Apache Spark мовою Java.

SUMMARY

Master’s qualifying paper « Development of a Library for Optimization of Big Data Processing on the Apache Spark Platform in Java »: 36 pages, 7 figures, 9 references, 1 supplement.

DATA ENGINEERING, PARALLEL COMPUTING, SOFTWARE LIBRARY, APACHE SPARK, JAVA.

Object of the study: data processing methods, data processing engines, Apache Spark.

Aim of the study: research of modern data processing methods, research of data processing engines, development of a library for optimizing data processing, research of Apache Spark.

The qualification work demonstrates the challenges of modern data processing tasks. The means of accelerating data processing have been analyzed. A library for optimizing big data processing on the Apache Spark platform using Java has been developed.

ЗМІСТ

Завдання на кваліфікаційну роботу……...2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз сучасних способів обробки даних..... 8
1.1 Аналіз існуючих способів обробки даних 8
1.2 Аналіз двигуна обробки даних Apache Spark..... 11
1.3 Постановка задачі 15
2 Проєктування архітектури бібліотеки 17
2.1 Аналіз вимог до бібліотеки 17
2.2 Проєктування модулів бібліотеки 18
2.3 Реалізація модулів бібліотеки..... 20
2.3.1 Реалізація модуля чисел 20
2.3.2 Реалізація модуля рядків 21
2.3.3 Реалізація модуля загальних класів 22
3 Практичне використання бібліотеки..... 24
3.1 Датасет для аналізу 24
3.2 Тестові задачі..... 25
3.3 Виконання тестування за допомогою розробленої бібліотеки 26
Висновки 30
Перелік посилань..... 31
Додаток А Коди інтерфейсів та класів підмодуля common..... 33

ВСТУП

Сучасні задачі аналізу даних є доволі складними та громіздкими. Дані, які потрібно проаналізувати, можуть мати будь-який вигляд починаючи від звичайних текстових форматів, до більш складних як медичні дані, чи просторові дані. Окрім складності вигляду даних операції, які проводяться над даними, також можуть бути доволі складними, витрачати багато ресурсів та часу. Обидві проблеми можна вирішити через розподілення роботи. Якщо процес перетворення даних буде виконувати не один процесор, а одразу декілька, швидкість перетворення виросте в декілька разів. Для розподілення обробки даних існують так звані двигуни обробки даних. Найбільш популярним є Apache Spark.

Apache Spark – це спеціально розроблений паралельний двигун обробки великих масивів даних. Двигун написано на Scala, проте має багато API для багатьох мов програмування як: Python, R, Java, C#. Apache Spark підтримує кластерну модель для паралельного обчислення. У кластері може бути багато вузлів які будуть виконувати роботу, проте обов’язково має бути вузол, який буде виступати у ролі диригенту та буде розподіляти роботу поміж робочими вузлами і отримувати результат роботи з вузлів.

Мета цієї роботи – це розробити бібліотеку яка буде допомагати інженерам даних які використовують Apache Spark в їх дослідженнях.

1 АНАЛІЗ СУЧАСНИХ СПОСОБІВ ОБРОБКИ ДАНИХ

1.1 Аналіз існуючих способів обробки даних

Обробка даних – це процес перетворення даних з непридатного для подальшого аналізу вигляду до вигляду, в якому з даних можна отримати певну цінність. Через велику варіативність типів даних задача обробки даних є доволі складною та ресурсозатратною. Задачі обробки даних широко розповсюджені серед науковців.

Науковці постійно оброблюють дані, які надходять їм як з різного роду приладів, так і те, що вони отримують власним шляхом. Дані можуть мати нечіткий формат та шум, що заважає їх розуміння. Тому їх пропускають через певні алгоритми, що зменшують або видаляють шуми, а також приводять до необхідного формату.

Самі дані можуть мати будь-який вигляд. Це можуть бути звичайні текстові формати як JSON [1] чи XML [2], так і більш складні, формати даних специфічного направлення як медичні чи просторові дані. Чим більш складніша структура даних яка оброблюється, тим довше буде вона оброблюватися. Окрім цього дані частіше всього вимірюються в гігабайтах чи в терабайтах даних. У деяких випадках як в великих конгломератах як Google чи Microsoft, може вимірюватися навіть в петабайтах. Таким чином складність обробки даних та час за який дані будуть оброблені виростає в декілька разів.

Обробку даних можна поділити на 2 категорії: обробка даних в реальному часі, та пакетна обробка даних.

Обробка даних в реальному часі – це процес швидкого та негайного аналізу даних, які надходять від різних джерел. Вона дозволяє системам реагувати на події під час надходження даних, що є критично важливим для багатьох застосунків, таких як біржові торги чи управління трафіком. Такі системи часто використовуються в транспортних мережах для швидкого

реагування на зміни дорожніх умов. У медицині це дозволяє лікарям отримувати актуальну інформацію про стан пацієнтів у режимі реального часу.

Фінансові установи використовують обробку даних в реальному часі для виявлення шахрайства та запобігання підозрілим транзакціям. У сфері розваг, стрімінгові платформи обробляють дані про перегляди та вподобання користувачів для надання персоналізованих рекомендацій. Завдяки цьому зменшуються затримки у передачі даних і підвищується ефективність обробки. Розробники можуть використовувати платформи реального часу для моніторингу та оптимізації продуктивності своїх застосунків. Обробка даних в реальному часі також має значення у сфері енергетики, де вона дозволяє оперативно керувати енергоресурсами.

Пакетна обробка даних (batch processing) – це метод обробки великих обсягів даних за допомогою автоматизованих завдань, які виконуються в певний час або за розкладом. Під час пакетної обробки дані збираються в групи або пакети і обробляються одночасно, а не індивідуально. Це дозволяє ефективно використовувати ресурси комп’ютера та знижувати витрати на обробку. Зазвичай пакетна обробка виконується у неробочий час, коли навантаження на систему мінімальне, що дозволяє уникнути конфліктів із іншими завданнями.

Прикладом пакетної обробки може бути щоденна обробка фінансових транзакцій у банках, коли всі транзакції за день обробляються вночі наступного дня одним великим датасетом. Це також використовується у великих організаціях для обробки даних про заробітну плату та податки. Пакетна обробка є важливою в наукових дослідженнях для аналізу великих обсягів даних, які накопичується за певний проміжок часу таких як геноміка чи кліматичні моделі. Вона дозволяє виконувати складні обчислення за менший час і з мінімальними перервами.

Ще однією перевагою пакетної обробки є можливість автоматизації рутинних завдань, що знижує ризик помилок та підвищує продуктивність

праці. Незважаючи на те, що результати пакетної обробки доступні з деякою затримкою, це підходить для завдань, де негайна відповідь не є критично важливою. Таким чином, пакетна обробка даних залишається одним із основних методів обробки інформації в багатьох сферах.

Для більш ефективної обробки даних часто використовують спеціалізовані обчислювальні системи, відомі як двигуни обробки даних. Серед них найбільш популярним є Apache Spark [3], який забезпечує швидку та масштабовану обробку великих обсягів даних. Spark розподіляє дані на кілька частин та паралельно відправляє їх на обробку різним процесам, що значно підвищує швидкість та ефективність обробки.

Apache Spark підтримує роботу з різними джерелами даних, включаючи бази даних, хмарні сховища та потокові дані. Він дозволяє аналізувати великі обсяги інформації в режимі реального часу, що є особливо важливим для фінансових установ та телекомунікаційних компаній. Spark підтримує обробку потокових даних за допомогою модуля Spark Streaming, який оброблює дані з високою швидкістю і забезпечує оперативну реакцію на події.

Крім того, Spark має модуль MLlib для машинного навчання, що дозволяє виконувати складні аналітичні завдання на великих даних. Застосування Spark дозволяє значно підвищити продуктивність та знизити витрати на інфраструктуру, оскільки він оптимізує використання обчислювальних ресурсів. Spark також забезпечує надійність та масштабованість систем, що є ключовим для компаній, які працюють з великими обсягами даних.

Завдяки своїм потужностям, Apache Spark виконує аналіз даних та отримує цінні інсайти швидше, ніж традиційні методи обробки. Spark також підтримує інтеграцію з іншими інструментами обробки даних, такими як Hadoop, що дозволяє ефективно використовувати існуючу інфраструктуру.

Таким чином, Spark є одним із найпопулярніших та найбільш зручних двигунів обробки даних, що дозволяє компаніям швидко адаптуватися до змінних умов ринку та приймати обґрунтовані рішення на основі даних.

1.2 Аналіз двигуна обробки даних Apache Spark

Apache Spark – це паралельний двигун обробки даних розроблений спеціально для обробки великих масивів даних. Двигун написаний на Scala, проте має API для багатьох мов програмування як: Python, R, Java, C#.

Apache Spark підтримує кластерну модель для паралельного обчислення. В кластері може бути багато вузлів, які будуть виконувати роботу, проте обов'язково має бути вузол, який буде виступати у ролі диригенту, що буде розподіляти роботу поміж робочими вузлами та отримувати результат роботи з вузлів.

Основна перевага Apache Spark в тому, що він може працювати на будь-якому комп'ютері де встановлена Java. Якщо комп'ютер підключений як підлеглий в мережу кластерів, то менеджер кластеру оцінить вузол за його можливостями. Оцінка проводиться за наявною кількістю пам'яті, яка доступна підлеглому, та кількість потоків, яку підлеглий може використати.

Коли робота надходить до менеджеру кластеру, він розподіляє її поміж усіма підлеглими кластеру, за їх можливостями. Алгоритм розподілення роботи між підлеглими дивиться, на всіх працівників, дивиться, на кількість потоків та пам'яті, яка доступна кожному з них, та дає кожному певну частину від загального пулу роботи, яка зможе максимально використати всю доступну пам'ять та всі доступні потоки підлеглого. Відповідно, якщо в кластері є підлеглий з найбільшою кількістю пам'яті та потоків, то менеджер кластеру дасть йому більший відсоток від загальної кількості даних ніж іншим підлеглим кластеру.

Для передачі роботи в кластер Apache Spark існує 3 підходи: використання локального Java процесу, передача даних та пайплайн перетворення до кластеру через підключення до кластеру та через spark-submit. Кожен з цих підходів має свої недоліки.

Використання локального Java процесу означає, що вся робота буде виконуватися на тому комп'ютері, на якому буде створено запит на виконання

цієї роботи. В цьому випадку кількість потоків буде залежати від параметру, який буде передано в SparkContext, а кількість оперативної пам’яті це загальна пам’ять всього Java процесу. Головний недолік цього підходу це те, що використовується Scala код в Java процесі. Scala використовує рефлексію щоб зчитувати дані з роботи, яку передали, проте для цього мають бути відкриті певні модулі стандартної бібліотеки Java. Цей недолік не є критичним і його легко обійти, проте командний рядок під час запуску Java процесу буде доволі довгим. Цей підхід краще використовувати коли проводиться перевірка пайплайну, якщо дані, які треба обробити доволі невеликого розміру, або потужностей комп’ютера вистачить на обробку всіх даних.

Передача даних до кластеру це найбільш незручний підхід з усіх. В цьому випадку кластер до якого під’єднується комп’ютер який посилає роботу не знає про класи які використовує цей комп’ютер. У випадку, якщо комп’ютер використовує класи, які були написані інженером, та їх не має в стандартній бібліотеці Java, кластер просто не знає цих класів та не зможе виконати роботу з помилкою ClassNotFoundException.

Використання spark-submit це найбільш вживаним підходом після використання локального Java процесу. В цьому випадку кластер має уявлення про всі класи, які використовуються в пайплайні та не треба відкривати модулі, як у випадку з локальним Java процесом. Під час запуску через spark-submit потрібно передати ім’я класу, в якому знаходиться main метод та ім’я менеджеру кластеру.

На рисунку 1.1 показано взаємодію між основними компонентами Apache Spark та підпорядкованими вузлами [4].

Apache Spark складається з декількох модулів: Spark SQL and DataFrames, Spark Streaming, Spark Mllib, Spark GraphX.

Apache Spark SQL та DataFrames – це потужні інструменти для роботи з структурованими даними [5]. Spark SQL забезпечує можливість виконувати SQL-запити на даних, що зберігаються у Spark, забезпечуючи зручний спосіб інтерактивного аналізу великих обсягів даних. DataFrames, у свою чергу,

надають високорівневий абстрактний клас для роботи з розподіленими колекціями даних, подібний до таблиць в реляційних базах даних.

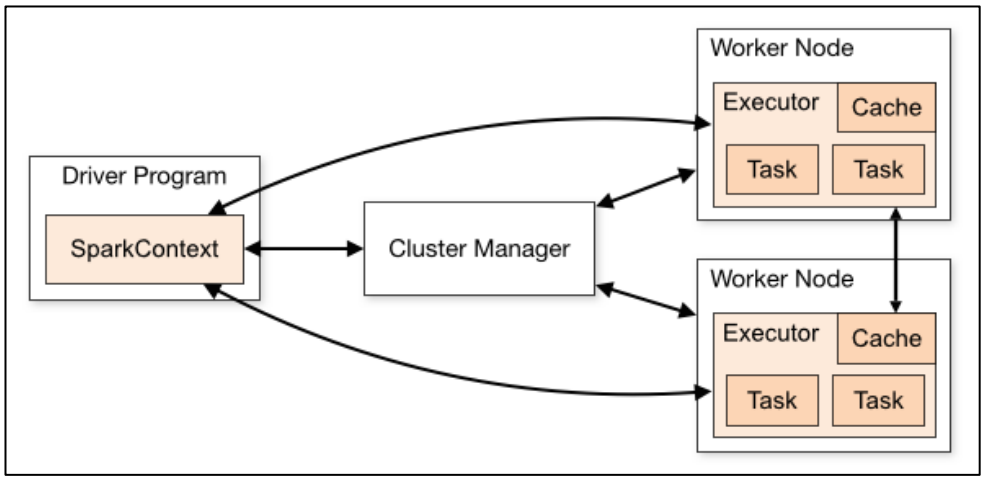


Рисунок 1.1 – Спрощене зображення комунікації кластеру Apache Spark

Основною перевагою Spark SQL є можливість використовувати SQL для доступу до даних з різних джерел, таких як HDFS, S3, Hive, та інших. DataFrames надають оптимізацію виконання завдяки використанню Catalyst optimizer, що автоматично покращує продуктивність виконання запитів.

DataFrames дозволяють легко маніпулювати та перетворювати дані, що спрощує аналіз та підготовку даних. Spark SQL та DataFrames інтегруються з іншими компонентами Spark, такими як MLlib для машинного навчання та GraphX для графових обчислень, що забезпечує єдину платформу для обробки даних. Вони також підтримують роботу з потоковими даними, що дозволяє здійснювати обробку в реальному часі.

Таким чином, Spark SQL та DataFrames забезпечують потужний та гнучкий інструментарій для роботи зі структурованими даними у великих масштабах.

Apache Spark Streaming – це компонент Apache Spark, який надає функціонал обробки потокових даних в режимі реального часу [6]. Він підтримує інтеграцію з багатьма джерелами потокових даних, такими як Kafka, Flume, та HDFS. Spark Streaming обробляє дані в невеликих мікро-батчах, що дозволяє здійснювати обчислення з високою продуктивністю.

Ця бібліотека надає можливість розробникам створювати складні поточні аналітичні додатки, комбінуючи функції обробки потокових та пакетних даних. Spark Streaming забезпечує безперебійну інтеграцію з іншими компонентами Spark, такими як MLlib для машинного навчання та GraphX для графових обчислень. Завдяки цьому можливе використання однієї платформи для обробки як історичних, так і реальних даних.

Важливою особливістю є висока масштабованість, яка дозволяє обробляти величезні обсяги даних у реальному часі без втрати продуктивності. Використання Spark Streaming допомагає у створенні додатків для моніторингу, аналітики та виявлення подій у реальному часі.

Таким чином, Apache Spark Streaming є потужним інструментом для обробки потокових даних, що надає можливість компаніям швидко реагувати на зміни та приймати обґрунтовані рішення.

Apache Spark MLlib – це бібліотека машинного навчання в екосистемі Apache Spark, яка забезпечує масштабовані машинно-навчальні алгоритми [7]. Вона інтегрується з іншими компонентами Spark, надаючи можливість здійснювати високопродуктивну обробку даних та їх аналіз. MLlib підтримує різні алгоритми, такі як класифікація, регресія, кластеризація та рекомендаційні системи.

Бібліотека також містить інструменти для підготовки та перетворення даних, такі як нормалізація, вибір характеристик та зниження розмірності. Використовуючи MLlib, розробники можуть створювати складні машинно-навчальні моделі, що масштабуються на великі обсяги даних. Завдяки своїй інтеграції зі Spark, MLlib надає функціонал здійснення розподілених обчисленнь, що підвищує продуктивність моделювання.

Mllib доволі зручна бібліотека машинного навчання, що робить її доступною для широкого кола користувачів. Вона включає прості API для створення та тренування моделей, а також для їх оцінки та застосування. MLlib сприяє швидкому розвитку машинного навчання у великих даних, забезпечуючи високу продуктивність та масштабованість.

Apache Spark GraphX – це бібліотека в екосистемі Apache Spark, призначена для обробки та аналізу графів [8]. Вона надає API для роботи з графами та граф-орієнтованими обчисленями, полегшуючи аналіз та візуалізацію даних у вигляді графів. З GraphX можна будувати та трансформувати графи, виконувати обчислення на вершинах та ребрах, а також використовувати різні графові алгоритми.

GraphX інтегрується з іншими компонентами Spark, надаючи можливість використовувати потужні можливості обробки даних Spark разом із графовою аналітикою. Вона підтримує як обчислення, орієнтовані на вершини, так і на ребра, забезпечуючи гнучку та ефективну обробку графів. Застосування GraphX включає аналіз соціальних мереж, системи рекомендацій та аналіз біологічних мереж.

GraphX також надає вбудовані алгоритми, такі як PageRank, Connected Components та Triangle Counting, які допомагають отримати корисну інформацію з комплексних графових даних. Вона використовує ефективне представлення графів, що забезпечує високу продуктивність та масштабованість. Комбінуючи потужності розподілених обчислень Spark з обробкою графів, GraphX пропонує надійну платформу для аналізу великих даних у вигляді графів.

Apache Spark – це доволі різноманітний та зручний інструмент обробки даних. В ньому присутня велика кількість вбудованих модулів обробки даних що робить його легким у використанні прямо з коробки.

1.3 Постановка задачі

Окрім використання двигуна обробки даних ще треба створити пайплайн обробки даних.

Пайплайн обробки даних – це послідовність етапів, через які проходять дані від моменту їх отримання до остаточної обробки та аналізу. На

початковому етапі дані збираються з різних джерел, таких як бази даних, файли, API або сенсори. Потім відбувається очищення даних, де видаляються помилки, пропущені значення та інші аномалії.

Наступний етап включає перетворення даних, де вони приводяться до єдиного формату, що спрощує подальший аналіз. Після цього дані можуть бути збагачені додатковою інформацією з інших джерел, якщо це необхідно. Потім відбувається завантаження даних в сховище або аналітичну систему для подальшого аналізу.

На фінальних етапах пайплайну дані аналізуються за допомогою різних методів, таких як статистичний аналіз, машинне навчання або візуалізація. Після аналізу результати можуть бути представлені у вигляді звітів, графіків або дашбордів. Важливою частиною пайплайну є також моніторинг та логування, які дозволяють відслідковувати роботу кожного етапу та виявляти можливі проблеми.

Apache Spark вирішує проблему повільної обробки даних, проте побудова пайплайну перетворення даних це задача інженера даних. Оскільки кожен датасет є унікальним, то і пайплайн, який його оброблює є унікальним. Проте в обох пайплайнах можуть бути спільні речі як витягування даних з певного джерела даних, обробка чи нормалізація даних та збереження в інше джерело даних.

Мета цієї роботи – це розробка бібліотеки яка містить деякикі частин пайплайну, які можна використовувати для пришвидшення побудови пайплайну. Зокрема, створення модульних компонентів, які можуть легко інтегруватися в існуючі системи та адаптуватися до вимог певних задач. Це дозволить автоматизувати рутинні завдання і зменшити час, необхідний для налаштування та запуску пайплайну.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ БІБЛІОТЕКИ

2.1 Аналіз вимог до бібліотеки

Бібліотека, що розробляється, має бути розділена на декілька пакетів в залежності від того, що певний пакет представляє. Бібліотека може розширюватися в залежності від майбутніх потреб. Кожен пакет повинен мати чітко визначену функціональність та бути незалежним від інших, щоб забезпечити легкість у використанні. Зокрема, пакети можуть включати модулі для збору даних, їх попередньої обробки, аналізу та візуалізації результатів.

Важливо також, щоб бібліотека підтримувала можливість інтеграції з вже існуючими пайплайнами, що використовуються за допомогою двигуна Apache Spark. Для досягнення цього необхідно співпадіння сигнатур класів та методів бібліотеки, що унеможливить помилки під час компіляції, чи вже під час запуску пайплайну. Для цього кожен з методів, що створюється в бібліотеці, виведено в окремий клас. Кожен такий клас імплементує інтерфейс Function з пакету Apache Spark, сигнатура якого відповідає сигнатурі, яку приймає метод map в пайплайні. Для того щоб клас можна було сериалізувати має бути імплементовано інтерфейс Serializable. Це дозволить забезпечити гнучкість і зручність його використання.

Для підвищення продуктивності розробки та тестування, слід використовувати юніт тести, що перевіряють окремі компоненти кожного пакету. Це допоможе швидше виявляти та виправляти помилки.

Основними підпакетами бібліотеки є: підпакет number для маніпуляцій з числами, підпакет string для маніпуляцій з рядками та підпакет common для загальних класів.

Підпакет number – підмодуль бібліотеки, що відповідає за операції над числами. Операціями можуть бути: парсинг чисел зі строкових значень,

обчислення за формулою, округлення чисел до цілої частини, чи до певної сотої після крапки.

Підпакет string – це підмодуль бібліотеки, що відповідає за різні операції над рядками. Операціями можуть бути злиття, розбиття, обрізання, заміна, тощо.

Підпакет common – підмодуль бібліотеки, в якому містяться класи та інтерфейси що використовуються в бібліотеці. Кожен розроблений клас в бібліотеці наслідує певний інтерфейс з цього підпакету.

До переваг бібліотеки можна віднести те, що будь-який її елемент інженер даних може замінити своїм. Для створення своєї реалізації буде достатньо імплементувати потрібний інтерфейс, який знаходиться в підпакеті common. Таким чином, клас буде реалізовувати метод з необхідною сигнатурою, а пайплайн даних може бути скомпільованим та запускатись, як на головному комп’ютері, так і на підпорядкованих комп’ютерах.

2.2 Проєктування модулів бібліотеки

На рисунках 2.1 – 2.3 зображено спроєктовані діаграми підпакетів чисел, рядків та підпакет common в якому знаходяться загальні класи для обох підмодулів.

На рисунку 2.1 зображено діаграму підпакету common. В цьому підпакеті присутні класи та інтерфейси які використовуються в бібліотеці та які інженер даних може використати для розробки своїх частин пайплайну.

На рисунку 2.2 відображено діаграму підпакету number. Цей підпакет відповідає за операції над числами. Операціями можуть бути: парсинг зі строкових значень, обчислення за певною формулою (лямбдою), тощо.

На рисунку 2.3 зображено діаграму підпакету strings. В цьому підпакеті присутні класи, що відповідають за обробку рядків.

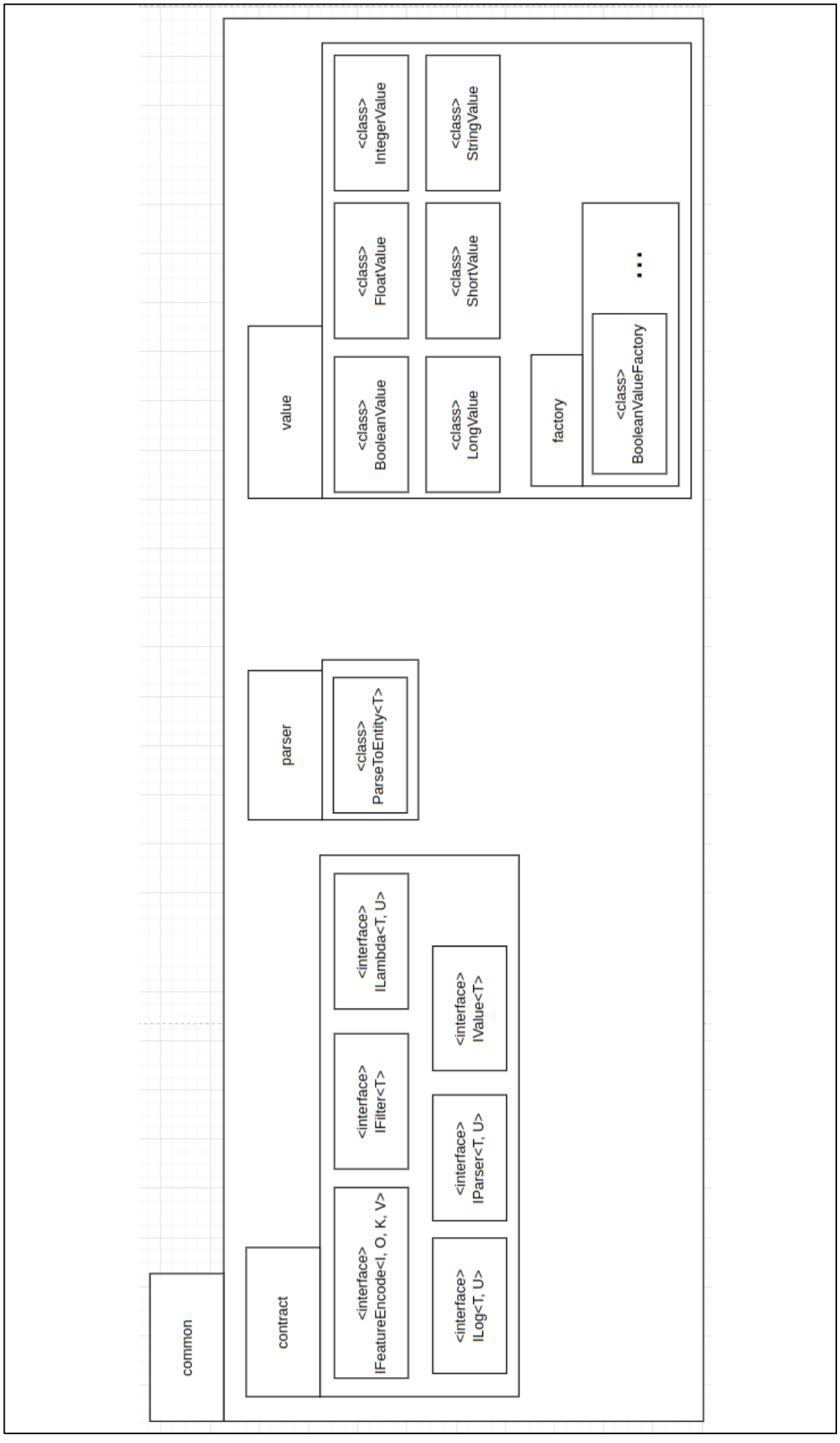


Рисунок 2.1 – Діаграма підпакету common

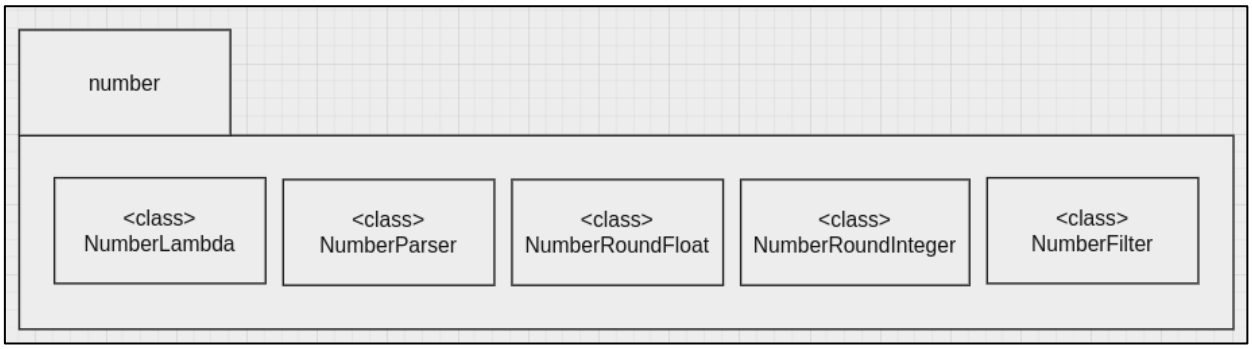


Рисунок 2.2 – Діаграма підпакету number

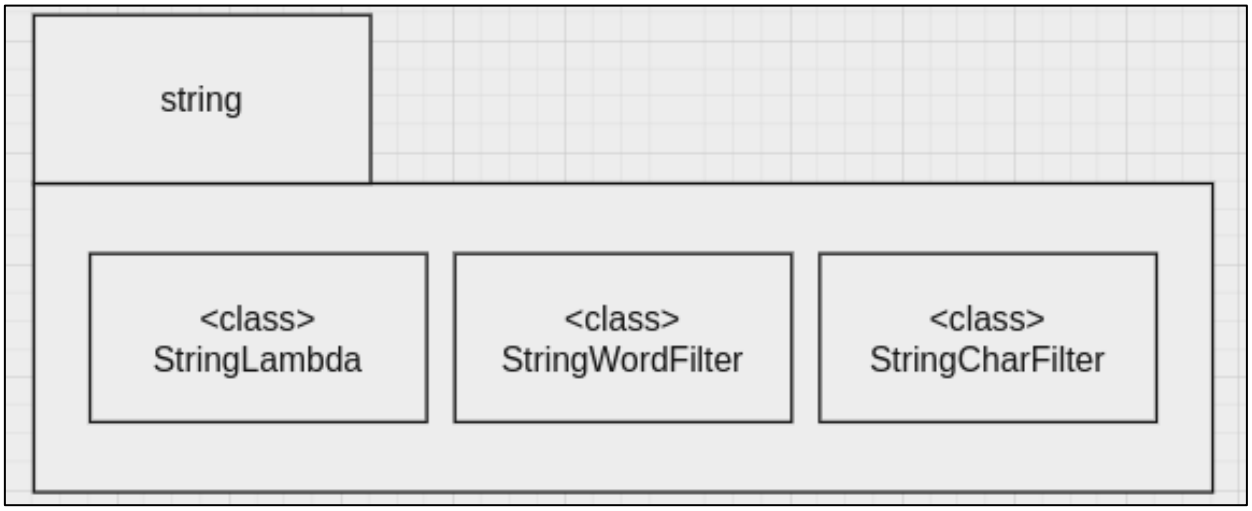


Рисунок 2.3 – Діаграма підпакету string

2.3 Реалізація модулів бібліотеки

2.3.1 Реалізація модуля чисел

У підмодулі number знаходяться 5 класів, які реалізують відповідні інтерфейси з підмодуля common. Розглянемо принципи роботи класів підмодуля number.

Клас NumberLambda відповідає за певну лямбду, яка буде запускатися на числових значеннях. У класі присутнє тільки одне поле method, яке відповідає за саму лямбду, що буде виконуватися в пайплайні. Клас імплементує інтерфейс ILambda з підпакету common, тому може бути використаний при побудові пайплайну.

Клас NumberParser відповідає за парсинг чисел зі строкових значень. Клас імплементує інтерфейс IParser, в якому в якості першого типу даних стоїть строкове значення, а в якості другого – чисельне. Таким чином, при використанні в пайплайні цей клас перетворює рядки у числа.

Клас NumberRoundFloat відповідає за округлення чисел з плавучою точкою. Клас імплементує ILambda, в якому в якості обох типів даних використовується FloatValue. В класі присутні 2 поля roundScale та roundingMode, які відповідають за кількість чисел після точки та за режим округлення відповідно. При використанні цього класу в пайплайні він буде змінювати кількість чисел після коми, тим самим зменшуючи чи збільшуючи точність обрахунків.

Клас NumberRoundInteger, як і попередній клас, округює числа, проте в цьому випадку він округлює одразу до цілого числа. Клас імплементує інтерфейс ILambda, в якому як перший тип використовується FloatValue, а другий IntegerValue. Під час використання цього класу в пайплайні всі дані з плавучою точкою будуть переведені в цілі числа.

2.3.2 Реалізація модуля рядків

Підмодуль string відповідає за операції над строковими значеннями. В підмодулі знаходяться 3 класи. Переглянемо деталі роботи класів підмодуля string.

Клас StringCharFilter відповідає за фільтрацію строкових значень від певних символів. Клас імплементує ILambda інтерфейс з типом StringValue, та присутнє поле blackListedChars в яке інженер даних передає всі символи які мають бути видалені зі строк.

Клас StringLambda – це клас що відповідає за виконання певної лямбди на строкових значеннях. Клас реалізує ILambda інтерфейс і обидва типи даних це StringValue.

Клас StringWordFilter – це клас що відповідає за фільтрацію строкових значень від певних слів. Якщо у рядку зустрічаються певні слова, які присутні в полі blackListedWords, вони не будуть пропущені далі. Клас реалізує ILambda інтерфейс, в якому перший тип даних це StringValue.

2.3.3 Реалізація модуля загальних класів

Підмодуль common відповідає за класи та інтерфейси, які використовуються в бібліотеці та які користувач бібліотеки може імплементувати, щоб інтегрувати в бібліотеку. Розглянемо принципи роботи підмодуля common.

Інтерфейс IValue<T> – це інтерфейс, що відповідає за абстракцію даних. Для використання бібліотеки з нестандартними типами даних, кожен тип має імплементувати цей інтерфейс. Методи, які мають бути імплементовані: getValue – щоб отримати інкапсульоване значення T, setValue – щоб задати інкапсульоване значення T, isNull – щоб перевірити чи інкапсульоване значення T нульове чи ні. В підмодулі присутні класи огортки для кожного Java примітиву та підпакет factory, в якому для кожного примітиву є класи фабрики, що перетворюють примітиви на огортки.

Інтерфейс IFeatureEncode<I, O, K, V> – це інтерфейс, що відповідає за кодування ознак даних. В інтерфейсі присутні одразу 4 невідомі типи даних. I – тип вхідного об’єкту, O – тип вихідного об’єкту, K – тип поля, яке має бути закодоване, V – тип поля після кодування. В інтерфейсі присутній метод getEncodingMap, за допомогою якого можна отримати масив, в якому зберігаються значення для перетворення.

Інтерфейс IFilter<T> – це інтерфейс, що відповідає за фільтрацію значень типу T за якоюсь ознакою. Метод getMethod відповідає за отриманя методу, який буде фільтрувати значення, та метод setMethod який встановить цей метод.

Інтерфейс ILambda<T> – це інтерфейс, що відповідає за лямбу, яка буде виконуватися для кожного об’єкту типу T. Метод call відповідає за виклик методу на кожному об’єкту T. Методи getMethod та setMethod відповідають за отримання та встановлення методу відповідно.

Інтерфейс ILog<T, U> – це інтерфейс, що відповідає за логування певних дій в середині пайплайну. Метод log відповідає за логування певної інформації з вхідного об’єкту T у вихідний об’єкт U.

Інтерфейс IParser<T, U> – це інтерфейс, що відповідає за парсинг об’єкту U з об’єкту T. В інтерфейсі присутній метод getParserMethod, який відповідає за отримання методу парсеру.

Коди інтерфейсів та класів підмодуля common зображено в Додатку А.

3 ПРАКТИЧНЕ ВИКОРИСТАННЯ БІБЛІОТЕКИ

3.1 Датасет для аналізу

Для тестування розробленої бібліотеки було взято один з найвідоміших наборів даних у сфері машинного навчання та статистики датасет іриси Фішера [9]. Цей датасет був вперше представлений британським статистиком та біологом Рональдом Фішером у 1936 році. Він містить 150 зразків квіток ірису, розділених на три види: Iris setosa, Iris versicolor і Iris virginica, по 50 зразків кожного виду. Кожен зразок має п’ять характеристик: довжину, ширину чашолистків і пелюсток та клас який відповідає підвиду ірису.

Датасет іриси Фішера використовують для демонстрації основних методів класифікації та аналізу даних. Він є відмінним інструментом для навчання алгоритмів машинного навчання, таких як K-ближчі сусіди (K-NN), логістична регресія, метод опорних векторів (SVM) та дерева рішень. Відмінності між видами ірисів на основі їхніх характеристик допомагають зрозуміти, як працюють ці алгоритми.

Цей датасет є одним з найбільш часто використовуваних у навчанні та дослідженнях завдяки своїй простоті та наочності. Він дозволяє дослідникам і студентам отримати перший досвід роботи з реальними даними та застосування різних методів аналізу і класифікації.

Важливість датасету іриси Фішера в історії машинного навчання не можна переоцінити, оскільки він залишається актуальним інструментом для навчання і досліджень.

Окрім цього, датасет дозволяє проводити візуалізацію даних, що допомагає краще зрозуміти розподіл і взаємозв’язки між різними видами ірисів. Цей набір даних є класичним прикладом, який допомагає розробникам та дослідникам вдосконалювати свої навички у галузі машинного навчання та статистики.

3.2 Тестові задачі

Для перевірки розробленої бібліотеки розглянуто задачу створення пайплайну для попередньої обробки датасету ірисів та виконання кодування ознак. Ця задача включає в себе: парсинг даних з csv файлу з даними, нормалізацію даних, кодування класів ірисів, виведення результатів пайплайну.

Парсинг даних з csv файлу – це перший крок, що включає зчитування даних із файлу та їх завантаження у структуру даних, зручну для подальшої обробки. Цей процес забезпечує перетворення даних з текстового формату у формат, який можна аналізувати за допомогою програмних засобів. Після парсингу дані часто проходять через етапи перевірки та валідації, щоб переконатися у їх коректності та відповідності очікуваній структурі. Це важливо для уникнення помилок на наступних етапах обробки. Парсинг також може включати попередню обробку, таку як видалення зайвих пробілів або конвертацію типів даних. В результаті дані стають придатними для подальшої обробки та аналізу.

Нормалізація даних – це процес, під час якого дані приводяться до певного проміжку для уніфікації їхнього масштабу. Це робиться для зменшення впливу відмінностей у масштабах на результати обчислень. Завдяки нормалізації, кожна ознака отримує однакову вагу при навчанні моделей машинного навчання. Це особливо важливо для алгоритмів, які чутливі до масштабів ознак, таких як методи класифікації та регресії. Нормалізація також допомагає покращити швидкість та стабільність навчання, знижуючи ризик переобучення моделі. В цьому датасеті процес нормалізації даних не є релевантним оскільки дані вже нормалізовані, проте для перевірки роботи бібліотеки дані про чашолистків будуть переведені до цілочисельного типу даних.

Кодування класів ірисів – це етап, в якому рядкові значення класів перетворюються на чисельні. Цей процес дозволяє алгоритмам машинного навчання легше обробляти категоріальні дані, оскільки вони зазвичай

працюють з числовими значеннями. Наприклад, клас “Iris setosa” може бути закодований як 0, “Iris versicolor” – як 1, а “Iris virginica” – як 2. Кодування значень забезпечує уніфікацію даних, що спрощує їх аналіз та моделювання. Крім того, цей етап дозволяє зберегти інформацію про категорії у зручному для аналізу форматі. Закодовані дані можуть бути легко інтегровані в різні алгоритми для подальшого навчання та прогнозування.

Виведення результатів пайплайну – це кінцева (термінальна операція), в якій виводяться дані після всіх попередніх етапів пайплайну. На цьому етапі оброблені дані можуть бути збережені в різних форматах, таких як CSV або бази даних, для подальшого аналізу та моделювання. Збережені результати можуть бути використані для створення звітів або візуалізацій, що допоможе виявити важливу інформацію про результати роботи пайплайну. Крім того, виведення даних дозволяє перевірити точність та коректність всього процесу обробки. Це важливо для виявлення можливих помилок та забезпечення високої якості даних. Виведені дані можуть бути інтегровані в інший пайплайн для подальшого використання.

3.3 Виконання тестування за допомогою розробленої бібліотеки

Для перевірки розробленої бібліотеки розроблено класи які інкапсулюють дані ірисів та класи частин пайплайну: IrisData, IrisDataValue, IrisParser, IrisDataNormalizedEncoded, IrisDataNormalizedEncodedValue, IrisDataNormalizedEncode, IrisDataNormalize, IrisDataNormalized, IrisDataNormalizedValue, IrisDataNormalizedValueLog, IrisDataNormalizedEncodedValueLog, IrisDataValueLog.

IrisData – це сирий тип даних в якому присутні поля для кожної з характеристик ірису. Цей клас використовується лише як огортка-інкапсулятор для даних ірису та не повинен використовуватися в самому пайплайні.

IrisDataValue – це клас-огортка навколо сирого типу IrisData який імплементує інтерфейс IValue<T>. Всі операції перетворення даних які присутні в пайплайні будуть відбуватися саме з цим класом.

IrisParser – це клас, що імплементує IParser<StringValue, IrisDataValue> який відповідає за парсинг рядків файлу в структуру даних ірису. В класі присутній один метод apply який відповідає за перетворення даних з одного типу до іншого.

IrisDataNormalizedEncoded – це сирий тип даних після нормалізації даних, в якому присутні поля для кожної з ознак ірису як IrisData, проте ознака класу перетворена з строкового значення в цілочисельне під час кодування ознаки.

IrisDataNormalizedEncodedValue – це клас-огортка навколо сирого типу IrisDataEncoded, який імплементує інтерфейс IValue<T>. Всі операції в пайплайні будуть відбуватися з цим типом даних, а не з сирою огорткою.

IrisDataNormalizedEncode – це клас кодувальник нормалізованих даних ірису. Клас імплементує інтерфейс IFeatureEncode<IrisDataNormalizedValue, IrisDataNormalizedEncodedValue, StringValue, IntegerValue>, в якому IrisDataNormalizedValue – це нормалізований тип даних який надходить до цієї пайплайну, IrisDataEncodedValue – це нормалізований та закодований тип даних що виходить з частини пайплайну, StringValue – тип даних класу ірису, який буде закодовано, IntegerValue – це тип даних класу ірису після кодування.

IrisDataNormalized – це сирий тип нормалізованих даних ірису після частини нормалізації в пайплайні. Клас виступає лише огорткою-інкапсулятором даних ірису, тому не має додаткової логіки.

IrisDataNormalize – це клас нормалізатор для класу IrisDataValue який нормалізує деякі поля даних ірису. Клас імплементує інтерфейс кодування IFeatureEncode<IrisDataValue, IrisDataNormalizedValue, FloatValue, IntegerValue>, в якому IrisDataValue – це клас-огортка навколо сирих даних ірису, IrisDataNormalizedValue – це клас з нормалізованими даними ірису, FloatValue – це тип даних до нормалізації, IntegerValue – це тип даних після нормалізації.

IrisDataNormalizedValue – це клас-огортка навколо сирого IrisDataNormalized типу який імплементує IValue<T> інтерфейс. Всі операції перетворення даних на нормалізованих даних відбуваються саме з цим класом.

IrisDataNormalizedValueLog – це клас-логер, що імплементує ILog інтерфейс та який виведе інформацію про нормалізовані дані ірису в стандартний потік програми.

IrisDataNormalizedEncodedValueLog – це клас-логер, який імплементує інтерфейс ILog та виведе інформацію про нормалізовані та закодовані дані ірису в стандартний потік програми.

IrisDataValueLog – це клас-логер який імплементує інтерфейс ILog, що виведе інформацію про дані про ірис на початку пайплайну після парсингу з файлу з даними.

Загальний потік пайплайну та перетворення даних з одного типу в інший наведено на рисунку 3.1, результати та код пайплайну – рис. 3.2 та 3.3.

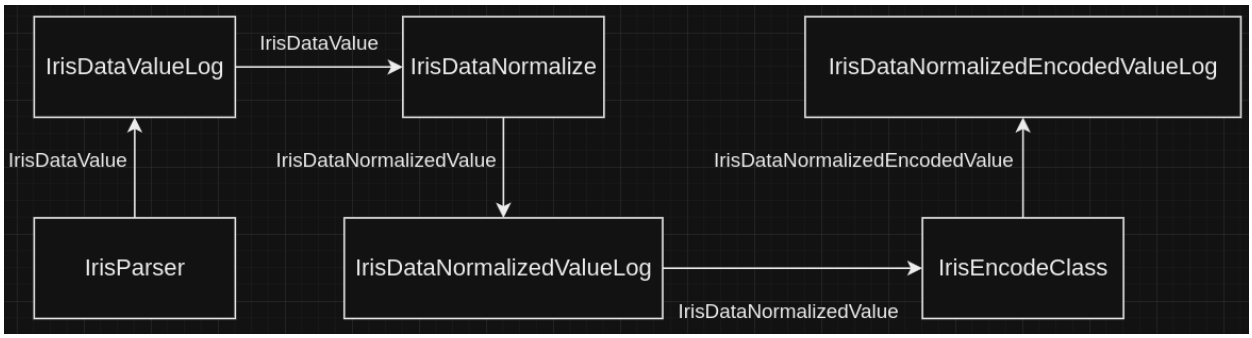


Рисунок 3.1 – Графічне зображення пайплайну

```
TestIris > testIris STANDARD_OUT
    PARSED IRIS:IrisData(sepalLength=7.0, sepalWidth=3.2, petalLength=4.7, petalWidth=1.4, irisClass=Iris-versicolor)
    PARSED IRIS:IrisData(sepalLength=6.3, sepalWidth=3.3, petalLength=6.0, petalWidth=2.5, irisClass=Iris-virginica)
    PARSED IRIS:IrisData(sepalLength=5.1, sepalWidth=3.5, petalLength=1.4, petalWidth=0.2, irisClass=Iris-setosa)
    PARSED NORMALIZED IRIS: IrisDataNormalized(sepalLength=7, sepalWidth=3, petalLength=4, petalWidth=1, irisClass=Iris-versicolor)
    PARSED NORMALIZED IRIS: IrisDataNormalized(sepalLength=6, sepalWidth=3, petalLength=6, petalWidth=2, irisClass=Iris-virginica)
    PARSED NORMALIZED IRIS: IrisDataNormalized(sepalLength=5, sepalWidth=3, petalLength=1, petalWidth=0, irisClass=Iris-setosa)
    PARSED NORMALIZED ENCODED IRIS: IrisDataNormalizedEncoded(sepalLength=7, sepalWidth=3, petalLength=4, petalWidth=1, irisClassEncoded=2)
    PARSED NORMALIZED ENCODED IRIS: IrisDataNormalizedEncoded(sepalLength=6, sepalWidth=3, petalLength=6, petalWidth=2, irisClassEncoded=3)
    PARSED IRIS:IrisData(sepalLength=6.4, sepalWidth=3.2, petalLength=4.5, petalWidth=1.5, irisClass=Iris-versicolor)
    PARSED NORMALIZED ENCODED IRIS: IrisDataNormalizedEncoded(sepalLength=5, sepalWidth=3, petalLength=1, petalWidth=0, irisClassEncoded=1)
    PARSED NORMALIZED IRIS: IrisDataNormalized(sepalLength=6, sepalWidth=3, petalLength=4, petalWidth=1, irisClass=Iris-versicolor)
    PARSED IRIS:IrisData(sepalLength=4.9, sepalWidth=3.0, petalLength=1.4, petalWidth=0.2, irisClass=Iris-setosa)
    PARSED IRIS:IrisData(sepalLength=5.8, sepalWidth=2.7, petalLength=5.1, petalWidth=1.9, irisClass=Iris-virginica)
    PARSED NORMALIZED IRIS: IrisDataNormalized(sepalLength=4, sepalWidth=3, petalLength=1, petalWidth=0, irisClass=Iris-setosa)
    PARSED NORMALIZED ENCODED IRIS: IrisDataNormalizedEncoded(sepalLength=6, sepalWidth=3, petalLength=4, petalWidth=1, irisClassEncoded=2)
    PARSED NORMALIZED IRIS: IrisDataNormalized(sepalLength=5, sepalWidth=2, petalLength=5, petalWidth=1, irisClass=Iris-virginica)
    PARSED NORMALIZED ENCODED IRIS: IrisDataNormalizedEncoded(sepalLength=5, sepalWidth=2, petalLength=5, petalWidth=1, irisClassEncoded=3)
    PARSED IRIS:IrisData(sepalLength=7.1, sepalWidth=3.0, petalLength=5.9, petalWidth=2.1, irisClass=Iris-virginica)
    PARSED NORMALIZED IRIS: IrisDataNormalized(sepalLength=7, sepalWidth=3, petalLength=5, petalWidth=2, irisClass=Iris-virginica)
    PARSED NORMALIZED ENCODED IRIS: IrisDataNormalizedEncoded(sepalLength=7, sepalWidth=3, petalLength=5, petalWidth=2, irisClassEncoded=3)
```

Рисунок 3.2 – Результат роботи пайплайну

```
irisData.map(( String el) -> {
    return irisParserBroadcast.getValue().call(new StringValue(el));  // parse string to IrisDataValue
  }) JavaRDD<IrisDataValue>
  // log IrisDataValue
  .map(irisDataValueLogBroadcast.getValue())
  // turn IrisDataValue into IrisDataNormalizedValue
  .map(irisDataNormalizeBroadcast.getValue()) JavaRDD<IrisDataNormalizedValue>
  // log IrisDataNormalizedValue
  .map(irisDataNormalizedLogBroadcast.getValue())
  // turn IrisDataNormalizedValue into IrisDataNormalizedEncodedValue
  .map(irisDataNormalizedEncodeBroadcast.getValue()) JavaRDD<IrisDataNormalizedEncodedValue>
  // log IrisDataNormalizedEncodedValue
  .map(irisDataNormalizedEncodedValueLogBroadcast.getValue())
  // end operation
  .foreach(( IrisDataNormalizedEncodedValue el) -> {
  });
```

Рисунок 3.3 – Код пайплайну

ВИСНОВКИ

Сучасні задачі аналізу даних є доволі складними та громіздкими. Дані, які потрібно проаналізувати, можуть мати будь-який вигляд, починаючи від звичайних текстових форматів, до більш складних як медичні дані, чи просторові дані. Кількість даних, яку треба обчислити, інколи перевищує можливості обробки деяких стаціонарних комп’ютерів.

Для пришвидшення та уможливлення обробки великих даних використовують двигуни обробки даних. Одним з таких двигунів є Apache Spark, спеціально розроблений паралельний двигун обробки великих масивів даних. Двигун написано на Scala, проте має багато API для багатьох мов програмування як: Python, R, Java, C#. Apache Spark підтримує кластерну модель для паралельного обчислення.

Для ефективного використання двигунів обробки даних, потрібно створювати пайплайни обробки даних. У кваліфікаційній роботі розроблено бібліотеку, яка пришвидшує розробку пайплайну завдяки заміні деяких частин готовими рішеннями. Окрім цього бібліотека надає зручний API для створення своїх імплементацій певних частин пайплайну.

ПЕРЕЛІК ПОСИЛАНЬ

1. Contributors to Wikimedia projects. JSON – wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/JSON (дата звернення: 05.10.2024).

2. Contributors to Wikimedia projects. XML – wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Xml (дата звернення: 17.10.2024).

3. Contributors to Wikimedia projects. Apache spark – wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Apache_Spark (дата звернення: 20.08.2024).

4. Jayvardhan Reddy. Deep-dive into Spark internals and architecture. 2019. *FreeCodeCamp.org*. URL: https://www.freecodecamp.org/news/deep-dive-into-spark-internals-and-architecture-f6e32045393b (дата звернення: 12.10.2024).

5. Zaharia M. Spark SQL and dataframes – spark 3.5.3 documentation. *Apache Spark™ – Unified Engine for large-scale data analytics.* URL: https://spark.apache.org/docs/latest/sql-programming-guide.html (дата звернення: 08.10.2024).

6. Zaharia M. Spark streaming – spark 3.5.3 documentation. *Apache Spark™ – Unified Engine for large-scale data analytics.* URL: https://spark.apache.org/docs/latest/streaming-programming-guide.html (дата звернення: 04.10.2024).

7. Zaharia M. MLlib: main guide – spark 3.5.3 documentation. *Apache Spark™ – Unified Engine for large-scale data analytics.* URL: https://spark.apache.org/docs/latest/ml-guide.html (дата звернення: 04.08.2024).

8. Zaharia M. GraphX – spark 3.5.3 documentation. *Apache Spark™ – Unified Engine for large-scale data analytics.* URL: https://spark.apache.org/docs/latest/graphx-programming-guide.html (дата звернення: 10.07.2024).

9. Fisher R. UCI machine learning repository. *UCI Machine Learning Repository*. URL: https://archive.ics.uci.edu/dataset/53/iris (дата звернення: 13.09.2024).

ДОДАТОК А

Коди інтерфейсів та класів підмодуля common

А.1 Код інтерфейсу IFeatureEncode

```
package org.nonchasd.sparkutil.common.contract;

import org.apache.spark.api.java.function.Function;

import java.io.Serializable;
import java.util.Map;

public interface IFeatureEncode<I, O, K, V> extends Serializable, Function<I, O>
{
        Map<K, V> getEncodingMap();
}
```

А.2 Код інтерфейсу IFilter

```
package org.nonchasd.sparkutil.common.contract;

import org.apache.spark.api.java.function.Function;

import java.io.Serializable;

public interface IFilter<T> extends Serializable, Function<T, Boolean> {
   java.util.function.Function<T, Boolean> getMethod();
```

```
    void setMethod(java.util.function.Function<T, Boolean> method);
}
```

A.3 Код інтерфейсу ILambda

```
package org.nonchasd.sparkutil.common.contract;

import org.apache.spark.api.java.function.Function;

import java.io.Serializable;

public interface ILambda<T, U> extends Serializable, Function<T, U> {
  java.util.function.Function<T, U> getMethod();

  void setMethod(java.util.function.Function<T, U> method);
}
```

A.4 Код інтерфейсу ILog

```
package org.nonchasd.sparkutil.common.contract;

import org.apache.spark.api.java.function.Function;

import java.io.Serializable;

public interface ILog<T, U> extends Serializable, Function<T, U> {
}
```

А.5 Код інтерфейсу IParser

```
package org.nonchasd.sparkutil.common.contract;

import org.apache.spark.api.java.function.Function;

import java.io.Serializable;

public interface IParser<T, U> extends Function<T, U>, Serializable {
    java.util.function.Function<T, U> getParserMethod();
}
```

А.6 Код інтерфейсу IValue

```
package org.nonchasd.sparkutil.common.contract;

import java.io.Serializable;

public interface IValue<T> extends Serializable {
  T getValue();

  void setValue(T value);

  boolean isNull();
}
```

А.7 Код класу ParseToEntity

```
package org.nonchasd.sparkutil.common.parser;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.nonchasd.sparkutil.common.contract.IParser;
import org.nonchasd.sparkutil.common.value.StringValue;
import java.io.Serializable;
import java.util.function.Function;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class ParseToEntity<T> implements IParser<StringValue, T>, Serializable
{

  private java.util.function.Function<StringValue, T> method;

  @Override
  public T call(StringValue rawData) throws Exception {
    return method.apply(rawData);
  }

  @Override
  public Function<StringValue, T> getParserMethod() {
    return method;
  }
}
```