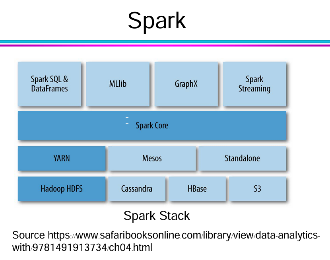
รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, แสดง

Description automatically generatedรูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, แผนภาพ, ออกแบบ

Description automatically generatedรูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, แผนภาพ, ไลน์

Description automatically generated2--Spark • โมเดลการประมวลผลแบบแบทช์ของ MapReduce ไม่เหมาะกับเฟรมเวิร์กบางอย่าง เช่นการทำงานแบบวนซ้ำ การปฏิสัมพันธ์ หรือการคำนวณตามคำขอที่มีบนชุดข้อมูลเดียว{\*}• เพื่อให้มีการประสานงานและความทนทานต่อข้อผิดพลาด โมเดล MapReduce ใช้โมเดลการดำเนินการแบบดึงที่ต้องการให้ผลลัพธ์ชั่วคราวถูกเขียนกลับไปยัง HDFS แล้วจึงถูกอ่านเพื่อการประมวลผลขั้นต่อไป{\*}• การอ่าน/เขียนไปยัง HDFS ใช้เวลามาก ดังนั้น MapReduce อาจเป็นไปได้ช้าสำหรับการคำนวณแบบปฏิสัมพันธ์ตามคำขอที่ทันที[\*]3--• Spark เป็นแนวคิดการคำนวณแบบกระจายที่เป็นแบบทั่วไปและรวดเร็วเป็นครั้งแรก ซึ่งไม่ต้องการให้การทำงานของกระแสข้อมูลถูกแยกออกเป็นชุดของงาน MapReduce{\*}• Spark บรรลุความเร็วสูงผ่านรูปแบบข้อมูลใหม่ที่เรียกว่า resilient distributed datasets (RDDs){\*}• RDDs ถูกเก็บไว้ในหน่วยความจำขณะที่กำลังถูกคำนวณ ซึ่งเป็นการลดการเขียนไปยังดิสก์ของข้อมูลชั่วคราวที่แพงและหายไป เรื่องการแบ่งปันข้อมูลในหน่วยความจำมีความเร็วระหว่างเครือข่ายและดิสก์มากถึง 10 ถึง 100 เท่า[\*]RDDS: Spark เป็นเทคโนโลยีที่เกี่ยวข้องกับการประมวลผลข้อมูลแบบ Real-time (เรียลไทม์) ที่มีความเร็วและประสิทธิภาพสูง ซึ่งมักใช้สำหรับการวิเคราะห์ข้อมูลที่มาจากแหล่งต่าง ๆ เพื่อให้ผลลัพธ์ที่แท้จริงและทันที โดยเฉพาะในสถานการณ์ที่ต้องการการตอบสนองอย่างรวดเร็วต่อข้อมูลที่เข้ามาอย่างต่อเนื่อง ซึ่งส่วนใหญ่นิยมใช้ในงานด้านวิเคราะห์ข้อมูล (Data Analytics) และการประมวลผลข้อมูลที่มีความซับซ้อน{\*}RDDS: Spark จะมีความสามารถในการจัดการกับข้อมูลขนาดใหญ่อย่างมีประสิทธิภาพ และสามารถประมวลผลข้อมูลได้ในรูปแบบแพร่หลายๆ ขั้นตอน (Parallel Processing) เพื่อให้สามารถทำงานได้เร็วขึ้น โดยใช้หลักการของ In-memory Computing ซึ่งทำให้สามารถเก็บข้อมูลในหน่วยความจำของระบบได้เพื่อลดเวลาในการเข้าถึงข้อมูลจากดิสก์{\*}Spark ยังมีโมดูลสำหรับการประมวลผลข้อมูลแบบกราฟ (Graph Processing), การเรียนรู้เชิงลึก (Deep Learning), และการประมวลผลข้อมูลแบบสตรีม (Stream Processing) ที่ช่วยให้สามารถแก้ไขปัญหาที่ซับซ้อนและหลากหลายได้อย่างมีประสิทธิภาพ{\*}Spark เป็นโครงการโอเพนซอร์สที่ได้รับความนิยมและการสนับสนุนจากชุมชนมากมาย โดยมี Apache Spark เป็นโครงการหลักที่ได้รับการพัฒนาและสนับสนุนโดย Apache Software Foundation[\*]4--• Spark มี API สำหรับโปรแกรมเชิงกระจายที่คล้ายกับโมเดล MapReduce แต่ถูกออกแบบให้มีความเร็วสำหรับคำถามแบบโต้ตอบและอัลกอริทึมการวนซ้ำที่สำคัญสำหรับการปรับแต่งและการเรียนรู้เครื่อง{\*}• มันบรรลุเป้าหมายนี้โดยหลักการแคชข้อมูลที่จำเป็นสำหรับการคำนวณในหน่วยความจำของโหนดในคลัสเตอร์[\*]5--• Spark ไม่เพียงรองรับการดำเนินการ map และ reduce เท่านั้น แต่ยังมี API หลายตัวสำหรับการประมวลผลข้อมูลแบบกระจายรวมถึง sample, filter, join และ collect เป็นต้น{\*}• Spark ถูกสร้างขึ้นด้วย Scala แต่ยังมีการให้บริการ API สำหรับโปรแกรมเชิงกระจายใน Scala, Java, R และ Python{\*}• Spark จัดเก็บชุดข้อมูลในหน่วยความจำให้มากที่สุดเท่าที่เป็นไปได้ตลอดการใช้งานของแอปพลิเคชัน เพื่อป้องกันการโหลดข้อมูลใหม่ระหว่างการวนซ้ำ[\*]6--• โปรแกรม Spark สามารถถูกรายละเอียดให้เป็นกราฟการไหลข้อมูลโดยใช้กราฟระบบเชิงตรรกะที่ Directed Acyclic Graph (DAG){\*}• เครื่องดำเนินการของ Spark ทราบล่วงหน้าจาก DAG และจึงกระจายการคำนวณในทั้งคลัสเตอร์และจัดการรายละเอียดของการคำนวณ{\*}• โดยการเพิ่มจำนวนโหนดในคลัสเตอร์และดังนั้นเพิ่มจำนวนหน่วยความจำที่ใช้สำหรับเก็บชุดข้อมูลขนาดใหญ่มาก Spark สามารถประมวลผลข้อมูลอย่างรวดเร็ว และเหมาะสำหรับการรันโปรแกรมแบบโต้ตอบได้[\*]Directed Acyclic Graph (DAG) เป็นโครงสร้างข้อมูลที่ประกอบด้วยโหนด (node) และเส้นเชื่อม (edge) โดยที่เส้นเชื่อมมีทิศทางและไม่มีรูปร่างวงรี นั่นหมายความว่าไม่มีการเชื่อมโหนดในลูกศรโดยที่สามารถเดินทางกลับมายังโหนดเริ่มต้นได้ ทำให้ไม่เกิดการวนวายในการเดินทางของโหนด{\*}การเรียงลำดับของโหนดใน DAG มีความสำคัญโดยมีกฎเบื้องต้นว่า ไม่สามารถมีลูกศรที่ชี้ไปยังโหนดที่มาก่อนหรือเป็นลูกศรกลับ (back edge) ในกรณีที่เส้นเชื่อมชี้ไปยังโหนดที่มาก่อนหน้าหรือเป็นลูกศรกลับ จะทำให้เกิดลูป (cycle) ซึ่ง DAG จะไม่มีลูปนั้นเกิดขึ้น{\*}DAG มักถูกนำมาใช้ในหลายแบบประยุกต์ต่าง ๆ รวมถึง:{\*}1. \*\*การประมวลผลข้อมูล\*\*: DAG ใช้ในการแสดงกระบวนการประมวลผลข้อมูลแบบแพร่หลายขั้นตอน (Pipeline) โดยแต่ละโหนดจะแทนการประมวลผลที่ต้องทำกับข้อมูล เช่น การดึงข้อมูล, การทำความสะอาดข้อมูล, การวิเคราะห์, และการเก็บผลลัพธ์ {\*}2. \*\*การวางแผนงาน\*\*: DAG ใช้ในการแสดงแผนงาน (Workflow) โดยแต่ละโหนดจะแทนกิจกรรมหรืองานที่ต้องทำ และเส้นเชื่อมจะแสดงลำดับการดำเนินงาน {\*}3. \*\*การคำนวณทางวิทยาศาสตร์\*\*: ในการแก้ปัญหาทางวิทยาศาสตร์หลาย ๆ ครั้ง เราต้องแสดงกระบวนการทางคณิตศาสตร์ด้วย DAG เช่น การประมาณค่า, การค้นหาเส้นทางสั้นที่สุด, และอื่น ๆ{\*}DAG เป็นโครงสร้างข้อมูลที่มีความยืดหยุ่นและเป็นประโยชน์มากในการแสดงและจัดการกระบวนการที่มีความซับซ้อนโดยที่ไม่มีการเกิดลูป ทำให้เป็นเครื่องมือที่มีประสิทธิภาพในการจัดการและวางแผนงานที่ต้องการลำดับการทำงานที่ชัดเจน[\*]7--• Spark ให้ความสำคัญกับการคำนวณเท่านั้นและไม่ใช่การจัดเก็บข้อมูล{\*}• สามารถใช้งานในคลัสเตอร์ที่นำเข้าเครื่องมือสำหรับการจัดเก็บข้อมูลและการจัดการคลัสเตอร์ได้ เช่น{\*}1. Hadoop พร้อม HDFS และ Yarn {\*}2. Mesos (ซอฟต์แวร์จัดการคลัสเตอร์ Apache, หน่วยควบคุม Mesos ทำงานบนทุกๆ โหนด สูงสุด 10,000 โหนดหรือมากกว่า และมี API สำหรับการจัดการทรัพยากรและการตั้งเวลาการทำงานข้ามศูนย์ข้อมูลและสภาพแวดล้อมคลาวด์ต่างๆ เช่น Hadoop, Spark, Kafka, Elasticsearch) {\*}3. Standalone (คลัสเตอร์ที่ง่ายดายที่มีโหนดหลักทำการกระจายงานไปยังโหนดทำงานทั้งหมดหรือหนึ่งหรือมากกว่า)[\*]Apache Mesos เป็นเครื่องมือสำหรับการจัดการทรัพยากรในระบบคอมพิวเตอร์แบบมาโมนอลิทิกส์ (Monolithic) ที่มีวัตถุประสงค์เพื่อให้การใช้ทรัพยากรบนเครื่องเซิร์ฟเวอร์มีประสิทธิภาพและมีประสิทธิภาพสูงสุด โดยการจัดการทรัพยากรเหล่านี้ให้กับแอปพลิเคชันและการทำงานที่ต่างกันในระบบเดียวกัน{\*}Mesos ทำงานโดยการแบ่งแยกทรัพยากรของเครื่องเซิร์ฟเวอร์ออกเป็นหน่วยเล็ก ๆ ที่เรียกว่า "กล่องทรัพยากร" (resource container) และมอบให้แต่ละแอปพลิเคชันหรืองานที่ต้องการทรัพยากรดังกล่าว กล่องทรัพยากรสามารถจัดการเครื่องเซิร์ฟเวอร์เป็นประสิทธิภาพสูงโดยใช้เทคโนโลยีการจัดการทรัพยากรแบบเขตแยก (resource isolation) เช่น Linux Containers (LXC) หรือ Docker containers{\*}หนึ่งในคุณสมบัติที่สำคัญของ Apache Mesos คือการทำงานร่วมกับระบบจัดการงานแบบโปรดิวเซอร์ (Scheduler) เพื่อจัดสรรทรัพยากรให้แก่งานต่าง ๆ อย่างมีประสิทธิภาพ ระบบโปรดิวเซอร์ทำหน้าที่เลือกทรัพยากรที่เหมาะสมและแจกจ่ายให้กับงานตามความต้องการ ทำให้ Mesos เป็นโครงสร้างพื้นฐานสำหรับการสร้างและจัดการระบบคอมพิวเตอร์แบบมาโมนอลิทิกส์แบบมากมาย เช่น ระบบคลัสเตอร์ขนาดใหญ่, ระบบที่มีการเปลี่ยนแปลงการใช้งานทรัพยากรบ่อยครั้ง, และระบบแบบคลาวด์ (cloud)[\*] Standalone ในบ context ของเทคโนโลยีและซอฟต์แวร์อาจมีความหมายต่าง ๆ ได้ตามที่ใช้งานกัน แต่ละที่สามารถแปลงเป็นโครงสร้างพื้นฐานที่เป็นอิสระและสามารถทำงานได้ด้วยตนเอง ดังนั้นข้อความข้างล่างนี้จะอธิบายเกี่ยวกับหลายที่ที่ Standalone สามารถมีได้ในวงการเทคโนโลยี:{\*}1. \*\*Standalone Server\*\*: หมายถึงเซิร์ฟเวอร์ที่ทำงานอย่างอิสระและไม่ต้องการเชื่อมต่อกับเครือข่ายหรือระบบอื่น ๆ เพื่อการทำงาน ซึ่งมักจะใช้ในระบบที่ต้องการความเสถียรและประสิทธิภาพ เช่น เซิร์ฟเวอร์เกมส์, เซิร์ฟเวอร์ฐานข้อมูล, หรือเซิร์ฟเวอร์เว็บที่มีการจัดเก็บข้อมูลสำคัญซึ่งไม่ต้องการการแชร์ข้อมูลกับเครือข่ายอื่น {\*}2. \*\*Standalone Application\*\*: หมายถึงแอปพลิเคชันที่ทำงานได้อย่างอิสระและไม่ต้องการโมดูลหรือโปรแกรมเสริมอื่น ๆ เพื่อให้ทำงานได้ สามารถทำงานบนเครื่องคอมพิวเตอร์เพียงเครื่องเดียว โดยไม่ต้องพึ่งพาระบบที่เชื่อมต่อกับเครือข่ายหรืออื่น ๆ{\*}3. \*\*Standalone Software\*\*: หมายถึงซอฟต์แวร์ที่สามารถทำงานอย่างอิสระและไม่ต้องการโปรแกรมเสริมหรืออุปกรณ์อื่น ๆ เพื่อให้ทำงานได้ เช่น ซอฟต์แวร์ที่ทำงานบนเครื่องคอมพิวเตอร์เพียงเครื่องเดียวโดยไม่ต้องพึ่งพาการเชื่อมต่อกับอินเทอร์เน็ตหรืออุปกรณ์เสริมอื่น ๆ{\*}การใช้งาน Standalone ช่วยเพิ่มความเสถียรและประสิทธิภาพในการทำงานของระบบ โดยลดความขึ้นอยู่กับระบบภายนอกลง นอกจากนี้ยังช่วยลดความซับซ้อนในการดูแลรักษาระบบและการเชื่อมต่ออื่น ๆ ที่อาจมีผลกระทบต่อประสิทธิภาพและเสถียรภาพของระบบ[\*]10--• เมื่อ Spark ถูกสร้างขึ้นร่วมกับ Hadoop มันจะใช้ YARN เพื่อจัดสรรและบริหารจัดการทรัพยากรในคลัสเตอร์ เช่น หน่วยประมวลผลและหน่วยความจำ ผ่านผู้จัดการทรัพยากร{\*}• Spark จะสามารถเข้าถึงแหล่งข้อมูลใด ๆ ใน Hadoop ได้ เช่น HDFS, HBASE, Hive หรือ Cassandra (ฐานข้อมูล NoSQL){\*}• Spark มีโมดูลหลักที่เรียกว่า Spark Core ซึ่งให้ฟังก์ชันพื้นฐานและทั่วไปผ่าน API[\*]11--• Spark จึงสร้างขึ้นบนฐานนี้ โดยสร้างไลบรารีเพื่อการใช้งานเฉพาะสำหรับหลายงานด้านวิทยาศาสตร์ข้อมูลที่ติดต่อกับ Hadoop{\*}• ไลบรารีที่ประกอบด้วยไม่ได้รวมอยู่ใน Spark Core ซึ่งช่วยเพิ่มความยืดหยุ่นในการพัฒนาไลบรารีส่วนใหม่บน Spark Core{\*}• ไลบรารีหลักประกอบไปด้วย{\*}• Spark SQL ให้ API สำหรับการติดต่อกับ Spark ผ่าน Spark SQL และผลลัพธ์สามารถส่งกลับมาเป็น Dataset (ชุดข้อมูลที่มีชนิดแบบมีประเภทและเป็นคอลเล็กชันของวัตถุที่เฉพาะของโดเมนซึ่งสามารถแปลงได้ตามคำสั่งทำงานเป็นพาราเลล) หรือ Data Frames (การรวบรวมข้อมูลที่กระจายอยู่ในหลายๆ คอลัมน์)[\*]12--• Spark Streaming ช่วยให้เป็นไปได้ในการประมวลผลและจัดการกับการสตรีมข้อมูลที่ไม่มีขอบเขตในเวลาจริง{\*}• MLlib เป็นไลบรารีของอัลกอริทึมการเรียนรู้ของเครื่องที่พบบ่อยที่ได้รับการนำเข้าและดำเนินการผ่าน Spark บน RDDs เช่น การจำแนกประเภท การถดถอย ฯลฯ{\*}• GraphX เป็นชุดของอัลกอริทึมและเครื่องมือสำหรับการจัดการกราฟและการดำเนินการและการคำนวณกราฟแบบขนาน[\*]13--• ส่วนประกอบเหล่านี้ร่วมกับโมเดลการเขียนโปรแกรมของ Spark มอบวิธีการที่มั่นคงและหลากหลายในการติดต่อกับทรัพยากรของคลัสเตอร์{\*}Resilient Distributed Datasets (RDDs){\*}• RDDs เป็นการนิยามโปรแกรมที่แสดงถึงชุดของวัตถุที่มีคุณสมบัติอ่านเท่านั้น (immutable) ซึ่งถูกแบ่งเป็นชุดข้อมูลต่าง ๆ ที่กระจายอยู่บนเครื่องคอมพิวเตอร์ชุดหนึ่ง)[\*]14--Spark คือแพลตฟอร์มการประมวลผลข้อมูลที่สามารถทำงานกับข้อมูลขนาดใหญ่ได้อย่างรวดเร็วและมีประสิทธิภาพ โดยที่คำว่า RDDs (Resilient Distributed Datasets) คือโครงสร้างข้อมูลหลักที่ Spark ใช้ในการจัดการและประมวลผลข้อมูลขนาดใหญ่ ต่อไปนี้เป็นรายละเอียดและคำอธิบายเกี่ยวกับ RDDs และ Spark APIs:{\*}### RDDs (Resilient Distributed Datasets){\*}- \*\*การสร้าง RDDs ผ่านการทำงานแบบขนาน\*\*:{\*}RDDs สามารถสร้างขึ้นได้โดยการดำเนินงานแบบขนานซึ่งจะโหลดชุดข้อมูลจากที่เก็บข้อมูล เช่น HDFS (Hadoop Distributed File System) หรือ S3 (Amazon Simple Storage Service) จากนั้นจะแบ่งข้อมูลนั้นเป็นกลุ่มๆ ที่สามารถเก็บไว้ในหน่วยความจำของโหนดทำงาน (worker nodes) เพื่อให้สามารถนำมาใช้ซ้ำได้ทันที (โดยใช้คำสั่ง `cache`) โดยปกติ RDDs จะถูกเก็บในหน่วยความจำ แต่สามารถย้ายไปยังดิสก์ได้หากต้องการพื้นที่ในหน่วยความจำสำหรับ RDDs ใหม่ๆ{\*}### Spark APIs{\*}- \*\*การดำเนินการของ Spark APIs\*\*:{\*}Spark APIs คือชุดของการดำเนินการที่สามารถสร้าง แปลง และส่งออก RDDs ในรูปแบบของโครงสร้างโปรแกรมเชิงฟังก์ชัน (functional programming constructs) ซึ่งจะคืนค่าวัตถุข้อมูลใหม่เสมอ (always return new data objects){\*}\*\*สรุป\*\*:{\*}- RDDs ใน Spark ช่วยให้การจัดการและประมวลผลข้อมูลขนาดใหญ่ทำได้อย่างมีประสิทธิภาพ โดยสามารถเก็บข้อมูลไว้ในหน่วยความจำเพื่อการใช้งานที่รวดเร็ว และสามารถย้ายข้อมูลไปยังดิสก์เมื่อจำเป็น{\*}- Spark APIs มอบความสามารถในการสร้างและจัดการ RDDs ด้วยวิธีการเชิงฟังก์ชัน ทำให้การเขียนโปรแกรมและการประมวลผลข้อมูลมีความยืดหยุ่นและมีประสิทธิภาพ{\*}การทำงานของ Spark นี้ช่วยให้การประมวลผลข้อมูลขนาดใหญ่มีประสิทธิภาพมากขึ้น ทั้งในแง่ของความเร็วและการใช้ทรัพยากรระบบอย่างเหมาะสม[\*]15--• มีสองประเภทของการดำเนินการที่สามารถใช้กับ RDDs ได้:{\*}1. Transformations เป็นการดำเนินการที่ใช้กับ RDD ที่มีอยู่เพื่อสร้าง RDD ใหม่ {\*}2. Actions เป็นการดำเนินการที่คืนผลลัพธ์กลับไปยังโปรแกรมขับเคลื่อน Spark - ทำให้เกิดการประสานหรือรวมข้อมูลในทุก ๆ พาร์ติชันใน RDD[\*]16--• ตัวอย่างเช่น map เป็นการแปลงข้อมูล เนื่องจากฟังก์ชัน map ถูกส่งและนำไปใช้กับทุก ๆ วัตถุที่เก็บอยู่ใน RDD และผลลัพธ์ของฟังก์ชันจะถูกแมปไปยัง RDD ใหม่{\*}• การรวมข้อมูลเช่น reduce เป็นการดำเนินการที่เป็น action เนื่องจาก reduce ต้องการให้ RDD ถูกแบ่งแต่งตามคีย์และค่ารวมหรือค่าเฉลี่ยเช่น ผลลัพธ์จะถูกคำนวณและคืนค่ากลับ{\*}• การกระทำส่วนใหญ่ใน Spark ถูกออกแบบขึ้นเพื่อการแสดงผลเท่านั้น - ส่งค่าเดี่ยวหรือรายการค่าเล็ก ๆ หรือเขียนข้อมูลกลับไปยังการจัดเก็บข้อมูลแบบกระจาย[\*]18--• Spark ใช้การแปลงแบบlazilyโดยการตรวจสอบลำดับของการแปลงทั้งหมดและการดำเนินการก่อนที่จะดำเนินการโดยการส่งงานไปยังคลัสเตอร์{\*}• การดำเนินการนี้ทำการดำเนินการอย่างlazilyให้มีการปรับปรุงสำคัญในการจัดเก็บข้อมูลและคำนวณ โดยการตรวจสอบลำดับการแปลงทั้งหมดเพื่อคำนวณเฉพาะข้อมูลที่จำเป็นสำหรับผลลัพธ์ เช่นหากการดำเนินการเป็น first() บน RDD Spark จะหลีกเลี่ยงการอ่านชุดข้อมูลทั้งหมดและคืนเพียงรายการแรกของ RDD เท่านั้น[\*]19--• รหัสในโปรแกรมขับเคลื่อนจะถูกประเมินอย่างlazilyบนเครื่องของไดรเวอร์เมื่อถูกส่งเข้าระบบ และเมื่อมีการดำเนินการ รหัสของไดรเวอร์จะถูกกระจายไปทั่วคลัสเตอร์เพื่อให้โหนดทำงานประมวลผลบนพาร์ติชันของ RDD ของตน{\*}• ผลลัพธ์จากนั้นถูกส่งกลับไปยังไดรเวอร์เพื่อการรวบรวมหรือคอมไพล์เพื่อเก็บรวบรวมผลลัพธ์[\*]20--• ลำดับการไหลของข้อมูลที่สามารถพบในการเขียนโปรแกรม Spark ทั่วไปมีดังนี้:{\*}1. กำหนด RDD(s) หนึ่งหรือมากกว่าหนึ่ง โดยการเข้าถึงข้อมูลที่เก็บบนดิสก์ (เช่น HDFS, Cassandra, HBase หรือ S3), การทำข้อมูลเป็นแบบขนาดเล็ก (การแบ่งชุดข้อมูลเป็นส่วนและส่งแต่ละส่วนของข้อมูลไปยังโหนดทำงานที่จะดำเนินการคำนวณ), การแปลง RDD ที่มีอยู่ หรือการเก็บไว้ในหน่วยความจำ[\*]21--2. เรียกใช้งานการดำเนินการบน RDD โดยการประยุกต์ใช้ closure (ฟังก์ชันที่ไม่ขึ้นกับตัวแปรหรือข้อมูลภายนอก) กับแต่ละองค์ประกอบของ RDD Spark มีตัวดำเนินการระดับสูงมากมายที่เกินกว่าเพียงแผนที่และลด{\*}3. ดำเนินการที่รวมข้อมูล (เช่น นับ, รวบรวม, บันทึก ฯลฯ) บน RDD ผลลัพธ์ การดำเนินการเริ่มต้นการคำนวณของการแปลง (ขั้นตอนที่ 2) และการดำเนินการ (ขั้นตอนที่ 3) บนคลัสเตอร์ (การดำเนินการlazily)[\*]22--• การClosuresสามารถเข้าถึงตัวแปรในระดับตำแหน่งที่อยู่ในบริเวณส่วนตัวของตน เมื่อต้องการใช้ตัวแปรระดับโลกในการแบ่งข้อมูลระหว่างClosuresที่ทำงานบนโหนดทำงานที่แตกต่างกัน มีการสนับสนุนตัวแปรที่ถูกแชร์ใน Spark ที่สามารถเข้าถึงได้โดยโหนดทำงานทั้งหมดในลักษณะที่จำกัด มีสองประเภทของตัวแปรดังนี้:{\*}1. Broadcast variables ตัวแปรการแพร่ส่งข้อมูลไปยังทุกๆ โหนดทำงาน แต่มีลักษณะเป็นตัวอ่านอย่างเดียวและมักจะใช้เป็นตารางค้นหาหรือรายการที่แชร์) [\*]23--2. Accumulators เป็นตัวแปรที่โหนดทำงานสามารถ "เพิ่ม" โดยใช้การดำเนินการที่สมมาตร เช่น x += 1 โดยทั่วไปใช้เป็นตัวนับ [\*]24--• การสนับสนุนการรับมือกับข้อผิดพลาดขั้นพื้นฐานของ Spark คือ:{\*}1. เนื่องจาก Apache Spark RDD เป็นชุดข้อมูลที่ไม่เปลี่ยนแปลง แต่ละ Spark RDD จำlineageของการดำเนินการที่กำหนดไว้บนชุดข้อมูลข้อมูลที่รองรับการรับมือกับข้อผิดพลาดในการสร้างมัน การไหลของlineageของการดำเนินการที่กำหนดไว้นี้เป็นข้อบังคับด้วยเหตุผลทางตรรกะที่เรียกว่า lineage grap{\*}2. หากโหนดทำงานล้มเหลวและมีพาร์ติชันใด ๆ ของ RDD หายไป จากนั้นส่วนนั้นสามารถคำนวณใหม่ (เล่นใหม่) จากชุดข้อมูลของlineageต้นฉบับโดยใช้ the lineage graph of operations[\*]25--3. เพื่อให้มีความทนทานต่อข้อผิดพลาดสำหรับ RDDs ทั้งหมดที่สร้างขึ้นได้ เราสามารถทำให้พวกเขาถาวรบนดิสก์โดยใช้คำสั่ง persistent() ซึ่งจะทำให้พวกเขาสามารถทำซ้ำผ่าน HDFS ระหว่างเอ็กซิคิวเตอร์ของ Spark ได้ในโหนดทำงานทั้งหลายในคลัสเตอร์{\*}• สำหรับชุดข้อมูลที่พอดีกับหน่วยความจำของคลัสเตอร์ Spark จะเร็วพอสมควรที่จะให้ผู้ใช้ทำงานร่วมและสำรวจข้อมูลขนาดใหญ่จาก shell โต้ตอบที่นำมาใช้งานซึ่งนำเสนอ Python REPL (Read Evaluate-Print Loop) ที่เรียกว่า PySpark[\*]เพิ่มเติมเกี่ยวกับการทำงานของ PySpark:{\*}PySpark เป็นเครื่องมือที่มีประสิทธิภาพสำหรับการประมวลผลข้อมูลใน Spark โดยใช้ Python REPL ซึ่งเป็นโหมดการทำงานที่อนุญาตให้ผู้ใช้ป้อนคำสั่ง Python และดูผลลัพธ์ที่ได้ทันที การใช้ PySpark ช่วยให้ผู้ใช้สามารถทำการวิเคราะห์และประมวลผลข้อมูลใน Spark ได้อย่างสะดวกสบายโดยใช้ภาษา Python ซึ่งเป็นภาษาที่นิยมและเข้าใจง่ายที่สุดในวงการวิทยาการข้อมูลและการเรียนรู้เครื่อง นอกจากนี้ PySpark ยังมีความสามารถในการใช้งานร่วมกับไลบรารี Python อื่น ๆ ที่มีอยู่เพื่อการวิเคราะห์ข้อมูลอย่างละเอียดและกว้างขวาง ทำให้มันเป็นเครื่องมือที่มีประสิทธิภาพและหลากหลายที่สำหรับการประมวลผลข้อมูลใน Spark โดยใช้ Python ในระดับขององค์กรหรือผู้ใช้ที่มีความเชี่ยวชาญทางด้าน Python[\*]Code[\*]26--เมื่อ PySpark ถูก execute แล้ว มันจะสร้าง SparkContext โดยอัตโนมัติเพื่อใช้งาน ผู้ใช้สามารถอ้างอิง SparkContext ผ่านตัวแปร sc ได้{\*}```{\*}% pyspark{\*}>>> text = sc.textFile(“shakespeare.txt”){\*}>>> text.collect(){\*}>>> exit(){\*}# หรือใช้ ctrl-d{\*}```{\*}Method textFile ของ sc สามารถใช้สร้าง RDD จากไฟล์ข้อความได้[\*]28--ใช้ PySpark เพื่อแก้ปัญหา Word Count{\*}```{\*}% pyspark{\*}>>> text = sc.textFile("shakespeare.txt"){\*}>>> from operator import add{\*}>>> def tokenize(text):{\*}... return text.split(){\*}>>> words = text.flatMap(tokenize){\*}```[\*]29--• สร้าง RDD ใหม่ที่เรียกว่า words โดยการแปลง RDD ชื่อ text ผ่านการใช้ flatMap operator พร้อมกับ closure ที่กำหนดไว้คือ tokenize ซึ่ง flatMap คล้ายกับ map แต่สามารถส่งค่าศูนย์หรือมากกว่าหนึ่งค่าสำหรับแต่ละองค์ประกอบของ RDD ในขณะที่ map ส่งค่าเพียงหนึ่งค่าสำหรับแต่ละองค์ประกอบของ RDD{\*}wc = words.map(lambda x : (x,1) ){\*}• แปลง RDD ชื่อ words เป็น RDD ใหม่ที่ชื่อ wc โดยใช้ map operator พร้อมกับ anonymous function (ด้วยคำสั่ง lambda ใน Python)[\*]30-->>> counts = wc.reduceByKey(add){\*}counts.saveAsTextFile("wc"){\*}• การดำเนินการ reduceByKey ถูกดำเนินการเพื่อรับจำนวนคำผ่านฟังก์ชัน add เข้าไปใน RDD ชื่อ counts{\*}• RDD ชื่อ Counts จากนั้นถูกบันทึกเป็นไฟล์ข้อความในไดเรกทอรี wc ภายใต้ไดเรกทอรีทำงานปัจจุบัน{\*}• ตรวจสอบไฟล์ข้อความที่มีการนับคำ{\*}% ls wc/{\*}% head wc/part-00000[\*]31--• เขียนโปรแกรมแอปพลิเคชัน Spark ใน Python คล้ายกับการทำงานกับ Spark ในโหมดอินเตอร์แอคทีฟ{\*}• โปรแกรมไดรเวอร์ใน Python อาจมีการประกาศข้อมูลบางส่วน (สำหรับตัวแปรที่แชร์), กำหนดClosuresสำหรับการแปลง RDD และวางแผนการดำเนินการของ RDD ขั้นต่อไปและการรวมข้อมูล แบบขั้นตอนละขั้น[\*]32--นี่คือโปรแกรมตัวอย่าง word\_count.py ที่แปลและอธิบายแต่ละบรรทัด:{\*}```python{\*}# ตัวอย่างโปรแกรม word\_count.py{\*}# เริ่มต้นด้วยการนำเข้าคลาส SparkContext จากไลบรารี pyspark{\*}from pyspark import SparkContext{\*}# นิยามฟังก์ชันหลักของโปรแกรม{\*}def main():{\*}# สร้าง SparkContext และกำหนดชื่อแอพพลิเคชันเป็น 'SparkWordCount'{\*}sc = SparkContext(appName='SparkWordCount'){\*}# อ่านข้อมูลจากไฟล์ที่ตั้งอยู่ที่ '/user/cloudera/wc/input.txt'{\*}input\_file = sc.textFile('/user/cloudera/wc/input.txt'){\*}# ดำเนินการแบบขั้นต่อไปโดยแยกคำจากข้อความแต่ละบรรทัดและกำหนดค่าเริ่มต้นเป็น 1{\*}# จากนั้นรวมคำซ้ำๆ ด้วยการรวมข้อมูลด้วยกัน{\*}counts = input\_file.flatMap(lambda line: line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b ){\*}# ตรวจสอบว่าถ้าสคริปต์ถูกเรียกโดยตรงหรือไม่{\*}if \_\_name\_\_ == "\_\_main\_\_":{\*}# เรียกใช้งานฟังก์ชันหลัก{\*}main(){\*}```{\*}อธิบาย:{\*}1. บรรทัดแรกคือคำอธิบายที่ใช้เรียกใช้งานคำสั่งแสดงผลเมื่อคุณอ่านโค้ด{\*}2. บรรทัดที่สองเป็นการนำเข้าคลาส SparkContext จากไลบรารี pyspark{\*}3. บรรทัดที่สามถึงห้าเป็นการนิยามฟังก์ชันหลักที่ชื่อว่า `main()` ซึ่งเป็นจุดเริ่มต้นของโปรแกรม{\*}4. บรรทัดที่หกสร้าง SparkContext โดยกำหนดชื่อแอพพลิเคชันเป็น 'SparkWordCount'{\*}5. บรรทัดที่เจ็ดอ่านข้อมูลจากไฟล์ที่ตั้งอยู่ที่ '/user/cloudera/wc/input.txt'{\*}6. บรรทัดที่แปดถึงสิบสองทำการแยกคำจากข้อความแต่ละบรรทัด กำหนดค่าเริ่มต้นเป็น 1 และรวมคำซ้ำๆ เข้าด้วยกัน{\*}7. บรรทัดสุดท้ายตรวจสอบว่าสคริปต์ถูกเรียกโดยตรงหรือไม่ และเรียกใช้งานฟังก์ชันหลักเมื่อสคริปต์ถูกเรียกโดยตรง[\*]33--```python{\*}# บันทึกผลลัพธ์ลงในไฟล์ข้อความที่ตั้งอยู่ที่ '/user/cloudera/wc/output'{\*}counts.saveAsTextFile('/user/cloudera/wc/output'){\*}# หยุด SparkContext เมื่อเสร็จสิ้นการใช้งาน{\*}sc.stop(){\*}# ตรวจสอบว่าสคริปต์ถูกเรียกโดยตรงหรือไม่ และเรียกใช้งานฟังก์ชันหลักเมื่อสคริปต์ถูกเรียกโดยตรง{\*}if \_\_name\_\_ == '\_\_main\_\_':{\*}main(){\*}```{\*}อธิบาย:{\*}- บรรทัดแรกใช้Method `saveAsTextFile()` เพื่อบันทึกผลลัพธ์ลงในไฟล์ข้อความที่ระบุตำแหน่งที่ '/user/cloudera/wc/output'{\*}- บรรทัดที่สองใช้Method `stop()` เพื่อหยุด SparkContext เมื่อเสร็จสิ้นการใช้งาน{\*}- บรรทัดสุดท้ายใช้เงื่อนไข `if \_\_name\_\_ == '\_\_main\_\_':` เพื่อตรวจสอบว่าสคริปต์ถูกเรียกโดยตรงหรือไม่ และเรียกใช้งานฟังก์ชันหลัก `main()` ในกรณีที่สคริปต์ถูกเรียกโดยตรง[\*]34--เพื่อ execute โปรแกรม Spark{\*}```{\*}% spark-submit --master local[\*] word\_count.py{\*}```{\*}ตัวเลือก master ที่ใช้งานได้แก่{\*}- local[\*]: รันโปรแกรม Spark บนเครื่องโฮสต์ที่เป็น local ด้วยกระบวนการมากเท่าที่จำเป็น{\*}- yarn: รันโปรแกรม Spark พร้อมกับ YARN เป็นตัวจัดการทรัพยากร{\*}- spark://host:port: รันโปรแกรม Spark บนโหมด standalone โดยมีโฮสต์เป็น master node (พอร์ตเริ่มต้น = 7077){\*}- mesos://host:port: รันโปรแกรม Spark บนคลัสเตอร์ Mesos (พอร์ตเริ่มต้น = 5050)[\*]35--การดำเนินการแปลงข้อมูลใน Spark:{\*}- `map(func)`: สร้าง RDD ใหม่โดยส่งผ่านทุกๆ องค์ประกอบของ RDD ต้นฉบับผ่านฟังก์ชัน `func`{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.map(lambda x: x \* 2){\*}>>> out.collect(){\*}[2, 4, 6, 8]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- จากนั้นเราใช้ `map()` เพื่อสร้าง RDD ใหม่ชื่อ `out` โดยคูณทุกๆ องค์ประกอบของ `rdd` ด้วย 2 ผ่านฟังก์ชัน lambda{\*}- สุดท้ายเราใช้ `collect()` เพื่อเก็บผลลัพธ์ทั้งหมดใน `out` และแสดงผลลัพธ์ในรูปของลิสต์ `[2, 4, 6, 8]`[\*]36--การดำเนินการกรองข้อมูลใน Spark:{\*}- `filter(func)`: สร้าง RDD ใหม่โดยเลือกเฉพาะองค์ประกอบของ RDD ต้นฉบับที่ฟังก์ชัน `func` คืนค่าเป็นจริง{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.filter(lambda x: x % 2 == 0){\*}>>> out.collect(){\*}[2, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- จากนั้นเราใช้ `filter()` เพื่อสร้าง RDD ใหม่ชื่อ `out` โดยเลือกเฉพาะตัวเลขที่หารด้วย 2 ลงตัว ผ่านฟังก์ชัน lambda{\*}- สุดท้ายเราใช้ `collect()` เพื่อเก็บผลลัพธ์ทั้งหมดใน `out` และแสดงผลลัพธ์ในรูปของลิสต์ `[2, 4]`[\*]37--การดำเนินการหาข้อมูลที่ไม่ซ้ำใน Spark:{\*}- `distinct()`: สร้าง RDD ใหม่ที่มีเฉพาะองค์ประกอบที่ไม่ซ้ำกันจาก RDD ต้นฉบับ{\*}```python{\*}>>> rdd1 = sc.parallelize([1, 2, 3, 2, 4, 3]){\*}>>> out = rdd1.distinct(){\*}>>> out.collect(){\*}[1, 2, 3, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd1` จากข้อมูล `[1, 2, 3, 2, 4, 3]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- จากนั้นเราใช้ `distinct()` เพื่อสร้าง RDD ใหม่ชื่อ `out` ที่มีเฉพาะองค์ประกอบที่ไม่ซ้ำกัน{\*}- สุดท้ายเราใช้ `collect()` เพื่อเก็บผลลัพธ์ทั้งหมดใน `out` และแสดงผลลัพธ์ในรูปของลิสต์ `[1, 2, 3, 4]`[\*]38--การดำเนินการ flatMap ใน Spark:{\*}- `flatMap(func)`: คล้ายกับ `map` แต่แต่ละรายการนำเข้าสามารถมีการแมปเป็น 0 หรือมากกว่ารายการผลลัพธ์ (ดังนั้นฟังก์ชันควรคืนค่าเป็นลำดับของรายการ){\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.map(lambda x: [x, x + 5]){\*}>>> out.collect(){\*}[[1, 6], [2, 7], [3, 8], [4, 9]]{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.flatMap(lambda x: [x, x + 5]){\*}>>> out.collect(){\*}[1, 6, 2, 7, 3, 8, 4, 9]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- ใช้ `map()` เพื่อสร้าง RDD ใหม่ชื่อ `out` โดยแต่ละรายการนำเข้าถูกแมปเป็นลิสต์ของค่า x และ x + 5{\*}- ใช้ `flatMap()` เพื่อสร้าง RDD ใหม่ชื่อ `out` โดยแต่ละรายการนำเข้าถูกแมปเป็นค่า x และ x + 5 และรวมรายการผลลัพธ์ทั้งหมดในลิสต์เดียวกัน[\*]39--การดำเนินการ Action ใน Spark:{\*}- `reduce(func)`: รวบรวมองค์ประกอบของชุดข้อมูลใน RDD โดยใช้ฟังก์ชัน `func` ฟังก์ชันนี้รับอาร์กิวเมนต์สองตัวและคืนค่าเพียงหนึ่งค่า และเป็นเชิงอนุมาน(commutative)และเป็นดั้งเดิม(associative)เพื่อให้สามารถคำนวณได้ถูกต้องในโหมดขนาน (ในคลัสเตอร์){\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.reduce(lambda a, b: a \* b){\*}24{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `reduce()` เพื่อรวบรวมองค์ประกอบของ RDD โดยใช้ฟังก์ชัน lambda ที่รับ a และ b เป็นอาร์กิวเมนต์และคืนค่า a \* b{\*}- ผลลัพธ์ที่ได้คือ 24 ซึ่งเป็นผลคูณของทุกองค์ประกอบใน RDD ต้นฉบับ[\*]40--การดำเนินการ Action ใน Spark:{\*}- `take(n)`: คืนค่าอาร์เรย์ที่มีองค์ประกอบ n อันแรกของ RDD{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.take(2){\*}[1, 2]{\*}>>> rdd.take(5){\*}[1, 2, 3, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `take(2)` เพื่อดึงค่าสองตัวแรกของ RDD และ `take(5)` เพื่อดึงค่าห้าตัวแรก ผลลัพธ์ที่ได้คือ `[1, 2]` และ `[1, 2, 3, 4]` ตามลำดับ[\*]41--การดำเนินการ Action ใน Spark:{\*}- `collect()`: คืนค่าทุกๆ องค์ประกอบของ RDD ในรูปของอาร์เรย์{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.collect(){\*}[1, 2, 3, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `collect()` เพื่อรวบรวมทุกๆ องค์ประกอบของ RDD และคืนค่าในรูปของอาร์เรย์ `[1, 2, 3, 4]`[\*]42--การดำเนินการ Action ใน Spark:{\*}- `takeOrdered(n, key=func)`: คืนค่าอาร์เรย์ที่มีองค์ประกอบ n อันแรกของ RDD ในลำดับธรรมชาติของคีย์ที่ระบุโดยฟังก์ชัน `func`{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.takeOrdered(3, lambda x: -x){\*}[4, 3, 2]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `takeOrdered(3, lambda x: -x)` เพื่อดึงค่าสามตัวแรกของ RDD ในลำดับจากมากไปน้อยตามคีย์ที่เป็นผลลัพธ์ของการกระทำ `-x` ผลลัพธ์ที่ได้คือ `[4, 3, 2]`[\*]43--การดำเนินการ Action ใน Spark:{\*}- `reduceByKey(func)`: คืนค่าชุดข้อมูลแบบกระจายใหม่ของคู่ (K, V) ใน RDD ที่มีค่าสำหรับแต่ละคีย์ถูกรวบรวมโดยใช้ฟังก์ชันลดลงที่กำหนดโดย func ((V, V) -> V){\*}```python{\*}>>> rdd = sc.parallelize([(1, 2), (3, 4), (3, 6), (1, 3), (3, 8)]){\*}>>> out = rdd.reduceByKey(lambda a, b: a + b){\*}>>> out.collect(){\*}[(1, 5), (3, 18)]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[(1, 2), (3, 4), (3, 6), (1, 3), (3, 8)]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `reduceByKey()` เพื่อรวบรวมค่าสำหรับแต่ละคีย์ ในกรณีนี้เราใช้ฟังก์ชันลดลงที่เพิ่มค่าเข้าไป{\*}- ผลลัพธ์ที่ได้คือ `[(1, 5), (3, 18)]` คู่คีย์-ค่าที่รวบรวมเท่ากับผลลัพธ์ที่ได้จากการรวมค่าสำหรับแต่ละคีย์[\*]44-การดำเนินการ Action ใน Spark:{\*}- `sortByKey()`: คืนค่า RDD ใหม่ของคู่ (K, V) ที่เรียงลำดับตามคีย์ K ในลำดับจากน้อยไปมาก{\*}```python{\*}>>> rdd = sc.parallelize([(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]){\*}>>> out = rdd.sortByKey(){\*}>>> out.collect(){\*}[(1, 'c'), (1, 'b'), (3, 'd'), (3, 'a'), (3, 'e')]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `sortByKey()` เพื่อเรียงลำดับคู่ (K, V) ตามคีย์ K ในลำดับจากน้อยไปมาก{\*}- ผลลัพธ์ที่ได้คือ `[(1, 'c'), (1, 'b'), (3, 'd'), (3, 'a'), (3, 'e')]` คู่ (K, V) ที่เรียงลำดับตามคีย์ K ในลำดับจากน้อยไปมาก[\*]45--การดำเนินการ Action ใน Spark:{\*}- `groupByKey()`: คืนค่า RDD ใหม่ของคู่ (K, iterable <V>) โดยที่ค่า V ในแต่ละคีย์ K ถูกกลุ่มรวมเข้าด้วยกันใน iterable{\*}```python{\*}>>> rdd = sc.parallelize([(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]){\*}>>> out = rdd.groupByKey(){\*}>>> out.map(lambda x: (x[0], list(x[1]))).collect(){\*}[(1, ['c', 'b']), (3, ['d', 'a', 'e'])]{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `groupByKey()` เพื่อกลุ่มค่า V ตามแต่ละคีย์ K{\*}- เราใช้ `map()` และ `collect()` เพื่อแปลงค่า V ในแต่ละคีย์ K ให้กลายเป็นรายการ (list) และรวบรวมผลลัพธ์ ผลลัพธ์ที่ได้คือ `[(1, ['c', 'b']), (3, ['d', 'a', 'e'])]` คู่ (K, iterable <V>) ที่ค่า V ถูกกลุ่มรวมเข้าด้วยกันใน iterable[\*]46--การดำเนินการ Action ใน Spark:{\*}- `count()`: คืนค่าจำนวนรายการข้อมูลใน RDD{\*}```python{\*}>>> rdd = sc.parallelize([(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]){\*}>>> rdd.count(){\*}5{\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เราใช้ `count()` เพื่อนับจำนวนรายการใน RDD ซึ่งในที่นี้คือ 5 รายการ[\*]47--การดำเนินการ Action ใน Spark:{\*}- `foreach(func)`: ปรับใช้ฟังก์ชัน `func` กับทุกรายการใน RDD ที่กำหนด{\*}```python{\*}>>> def f(x):{\*}... print(x){\*}...{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.foreach(f){\*}```{\*}ในตัวอย่างนี้:{\*}- เริ่มต้นด้วยการสร้าง RDD ชื่อ `rdd` จากข้อมูล `[1, 2, 3, 4]` โดยใช้ `parallelize()` Methodของ `SparkContext` (sc){\*}- เรากำหนดฟังก์ชัน `f(x)` ที่จะพิมพ์ค่า `x`{\*}- เราใช้ `foreach()` เพื่อใช้ฟังก์ชัน `f` กับทุกองค์ประกอบใน RDD และพิมพ์ผลลัพธ์ที่ได้ ผลลัพธ์ที่ได้คือ การพิมพ์แต่ละค่าใน RDD ในบรรทัดแยกกัน[\*]48--ใน Spark, ตัวแปร Broadcast ใช้เพื่อส่งข้อมูลให้กับโปรแกรมของ Spark ที่ทำงานบนคอมพิวเตอร์ต่าง ๆ ในเครือข่าย โดยไม่ต้องส่งข้อมูลซ้ำในแต่ละงาน การส่งข้อมูลด้วย Broadcast สามารถช่วยลดการทำงานของเครือข่ายและทำให้โปรแกรมทำงานได้เร็วขึ้นได้{\*}ด้านล่างเป็นตัวอย่างการใช้งาน Broadcast variables ใน Spark:{\*}ในโปรแกรมของไดรเวอร์:{\*}```python{\*}b = sc.broadcast([1, 2, 3, 4]){\*}```{\*}ในฟังก์ชันปิด:{\*}```python{\*}def f(x):{\*}return b.value[x]{\*}```{\*}จากนั้นเราสามารถใช้ Broadcast variables ในการแผนและประมวลผล RDD ได้:{\*}```python{\*}rdd = sc.parallelize([0, 3]){\*}out = rdd.map(f){\*}print(out.collect()) # ผลลัพธ์ที่คาดหวังคือ [1, 4]{\*}```{\*}ในกรณีนี้:{\*}- เริ่มต้นโดยการสร้าง Broadcast variable `b` ที่ส่งรายการ `[1, 2, 3, 4]`{\*}- แล้วเรากำหนดฟังก์ชัน `f(x)` ซึ่งใช้ Broadcast variable `b` เพื่อเข้าถึงค่าตามดัชนี `x`{\*}- เราสร้าง RDD `rdd` โดยใช้ `parallelize()` เพื่อสร้าง RDD ที่มีค่า `[0, 3]`{\*}- เราใช้ `map()` เพื่อแผนและประมวลผลข้อมูลใน RDD ด้วยฟังก์ชัน `f` และเก็บผลลัพธ์ใน `out`{\*}- สุดท้ายเราใช้ `collect()` เพื่อรวบรวมผลลัพธ์ทั้งหมดที่เกิดขึ้นใน RDD ซึ่งคาดว่าจะได้ผลลัพธ์ `[1, 4]` ดังนั้นเมื่อพิมพ์ `out.collect()` ผลลัพธ์ที่ได้คือ `[1, 4]` ตามที่คาดหวัง[\*]49--ใน Spark, ตัวแปร Accumulator ใช้เพื่อการนับหรือการสะสมค่าของข้อมูลในระหว่างการประมวลผลข้อมูลของ RDD บางอย่างโดยไม่ต้องรวบรวมผลลัพธ์กลับไปที่โปรแกรมหลัก ตัวแปร Accumulator สามารถใช้เพื่อจัดการงานที่สามารถนับหรือสะสมค่าได้ภายในโปรแกรม Spark โดยไม่ต้องรวบรวมผลลัพธ์กลับไปที่โปรแกรมหลัก{\*}ด้านล่างเป็นตัวอย่างการใช้งานตัวแปร Accumulator ใน Spark:{\*}ในโปรแกรมของไดรเวอร์:{\*}```python{\*}rdd = sc.parallelize([1, 2, 3, 4]){\*}c = sc.accumulator(0){\*}```{\*}ในฟังก์ชันปิด:{\*}```python{\*}def f(x):{\*}global c{\*}c += x{\*}```{\*}จากนั้นเราสามารถใช้ Accumulator variable ในการนับหรือสะสมค่าข้อมูลของ RDD ได้:{\*}```python{\*}rdd.foreach(f){\*}print(c.value) # ผลลัพธ์ที่คาดหวังคือ 10{\*}```{\*}ในกรณีนี้:{\*}- เริ่มต้นโดยการสร้าง RDD `rdd` โดยใช้ `parallelize()` เพื่อสร้าง RDD ที่มีค่า `[1, 2, 3, 4]`{\*}- เราสร้าง Accumulator variable `c` ด้วยค่าเริ่มต้นเป็นศูนย์{\*}- ภายในฟังก์ชัน `f(x)` เราใช้ `global c` เพื่อให้สามารถเข้าถึงตัวแปร Accumulator `c` จากภายในฟังก์ชัน และเพิ่มค่า `x` เข้าไปในตัวแปร Accumulator `c`{\*}- เราใช้ `foreach()` เพื่อใช้ฟังก์ชัน `f` กับทุกรายการใน RDD และทำการนับหรือสะสมค่า `x` ลงในตัวแปร Accumulator `c`{\*}- สุดท้ายเราใช้ `c.value` เพื่อเข้าถึงค่าปัจจุบันของ Accumulator `c` ซึ่งคาดว่าจะได้ผลลัพธ์ 10 ที่คาดหวัง ดังนั้นเมื่อพิมพ์ `c.value` ผลลัพธ์ที่ได้คือ 10 ตามที่คาดหวัง[\*]50--ใน Spark, คุณสามารถตั้งระดับการแสดงข้อมูลของบันทึกได้โดยใช้Method `setLogLevel()` ของ SparkContext (sc) เพื่อกำหนดระดับของบันทึกที่คุณต้องการให้แสดงในขณะที่โปรแกรมของคุณทำงาน ค่า level สามารถเป็นได้แก่ "ALL", "TRACE", "DEBUG", "INFO", "WARN", "ERROR", "FATAL", และ "OFF" โดยต้องการส่งระดับของบันทึกที่คุณต้องการเพื่อใช้งานMethodนี้{\*}ตัวอย่างการใช้งาน:{\*}```python{\*}sc.setLogLevel("ALL") # ตั้งระดับการแสดงข้อมูลเป็น "ALL"{\*}sc.setLogLevel("ERROR") # ตั้งระดับการแสดงข้อมูลเป็น "ERROR"{\*}```{\*}ในตัวอย่างนี้:{\*}- เราใช้ `setLogLevel()` เพื่อกำหนดระดับของบันทึกที่ต้องการให้แสดง{\*}- ในตัวอย่างด้านบน เราตั้งระดับการแสดงข้อมูลเป็น "ALL" และ "ERROR" ตามลำดับ ทำให้บันทึกทั้งหมดและเฉพาะข้อผิดพลาดถูกแสดงในขณะที่โปรแกรมทำงาน[\*]SparkLab[\*]การใช้งาน Spark ในการอ่านข้อมูลจากไฟล์ที่เก็บบน Hadoop นั้นเป็นกระบวนการที่สามารถทำได้ง่ายๆ ดังนี้:{\*}1. ใช้โปรแกรมที่เหมาะสมเช่น nano เพื่อสร้างไฟล์ข้อความ (test.txt) และนำข้อความที่ต้องการใส่ลงไปในไฟล์นั้น{\*}```{\*}% nano test.txt{\*}```{\*}หลังจากนั้นให้บันทึกและออกจากโปรแกรม nano{\*}2. สร้างไดเรกทอรีใน Hadoop และทำการอัพโหลดไฟล์ที่สร้างไว้ไปยังไดเรกทอรีนั้น{\*}```{\*}% hadoop fs -mkdir /user/cloudera/sw{\*}% hadoop fs -put test.txt /user/cloudera/sw/test.txt{\*}```{\*}3. เริ่มต้นโปรแกรม PySpark{\*}```{\*}% pyspark{\*}```{\*}4. ใช้คำสั่ง `sc.textFile()` เพื่อโหลดไฟล์ข้อความจาก Hadoop{\*}```python{\*}>>> text = sc.textFile('/user/cloudera/sw/test.txt'){\*}```{\*}5. ใช้ `collect()` เพื่อเก็บข้อมูลทั้งหมดจากไฟล์ในรูปแบบของ RDD{\*}```python{\*}>>> text.collect(){\*}```{\*}6. เมื่อเสร็จสิ้นการใช้งาน PySpark ให้ออกจากโปรแกรม{\*}```{\*}>>> exit(){\*}```{\*}หรือใช้ Ctrl + D เพื่อออกจากโปรแกรม PySpark[\*]ใช้ PySpark เพื่อแก้ปัญหา Word Count ดังนี้:{\*}1. เริ่มต้นโปรแกรม PySpark{\*}```{\*}% pyspark{\*}```{\*}2. ใช้คำสั่ง `sc.textFile()` เพื่อโหลดไฟล์ข้อความจาก Hadoop{\*}```python{\*}>>> text = sc.textFile('/user/cloudera/sw/test.txt'){\*}```{\*}3. import operator และกำหนดฟังก์ชัน tokenize ซึ่งเป็นการแยกข้อความเป็นคำๆ{\*}```python{\*}>>> from operator import add{\*}>>> def tokenize(line):{\*}... return line.split(){\*}```{\*}4. ใช้ `flatMap()` เพื่อแยกแต่ละคำจากข้อความ{\*}```python{\*}>>> words = text.flatMap(tokenize){\*}```{\*}ดังนั้น, ตอนนี้คุณมี RDD ที่ประกอบด้วยคำแยกแยะแล้วพร้อมที่จะนับจำนวนคำต่างๆ ในข้อความของคุณโดยใช้ Word Count Algorithm ต่อไป! [\*]นี่คือขั้นตอนที่เหลือของการแก้ปัญหา Word Count โดยใช้ PySpark:{\*}1. ใช้ `map()` เพื่อทำการแปลงคำใน RDD เป็นคู่ (คำ, 1) เพื่อเตรียมสำหรับการนับจำนวนคำ{\*}```python{\*}>>> wc = words.map(lambda x: (x, 1)){\*}```{\*}2. ใช้ `reduceByKey()` เพื่อรวมจำนวนคำที่เหมือนกันโดยใช้ฟังก์ชัน `add` ซึ่งจะรวมค่าเข้าด้วยกัน{\*}```python{\*}>>> counts = wc.reduceByKey(add){\*}```{\*}3. บันทึกผลลัพธ์เป็นไฟล์ข้อความบน Hadoop{\*}```python{\*}>>> counts.saveAsTextFile('/user/cloudera/sw'){\*}```{\*}หลังจากการบันทึกไฟล์เสร็จสิ้น, คุณสามารถตรวจสอบไฟล์ที่บันทึกไว้ด้วยคำสั่ง Hadoop ดังนี้:{\*}4. ใช้คำสั่ง `hadoop fs -ls` เพื่อดูไฟล์ที่อยู่ในไดเรกทอรี{\*}```{\*}% hadoop fs -ls /user/cloudera/sw{\*}```{\*}5. ใช้คำสั่ง `hadoop fs -cat` เพื่อดูเนื้อหาของไฟล์ที่เก็บ Word Count{\*}```{\*}% hadoop fs -cat /user/cloudera/sw/part-00000{\*}```{\*}นี่คือข้อมูลที่ปรากฏบนไฟล์หลังจากการนับจำนวนคำเสร็จสิ้น โดยไฟล์จะประกอบไปด้วยคำและจำนวนที่ปรากฏของแต่ละคำในข้อความที่นำเข้าไว้ในระบบ Spark และปรากฏบนไดเรกทอรี Hadoop ที่คุณได้ระบุไว้[\*]การทำแปลงด้วยการใช้ `map(func)` ใน Spark มีความหมายคือการสร้าง RDD ใหม่โดยการผ่านทุกๆ สมาชิกของ RDD ต้นฉบับผ่านฟังก์ชัน `func` ซึ่งเป็นฟังก์ชันที่กำหนดโดยผู้ใช้เอง ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.map(lambda x: x \* 2){\*}>>> out.collect(){\*}[2, 4, 6, 8]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 4]`{\*}- เราใช้ `map()` เพื่อทำการแปลงทุกๆ สมาชิกของ RDD ต้นฉบับ โดยการคูณด้วย 2{\*}- ผลลัพธ์ที่ได้คือ RDD `out` ที่มีสมาชิกเป็น `[2, 4, 6, 8]` [\*]การใช้ `filter(func)` ใน Spark มีความหมายคือการสร้างชุดข้อมูลใหม่ใน RDD โดยการเลือกสมาชิกของ RDD ต้นฉบับที่ฟังก์ชัน `func` คืนค่าเป็นจริง (True) ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.filter(lambda x: x % 2 == 0){\*}>>> out.collect(){\*}[2, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 4]`{\*}- เราใช้ `filter()` เพื่อเลือกเฉพาะสมาชิกของ RDD ต้นฉบับที่หารด้วย 2 ลงตัว{\*}- ผลลัพธ์ที่ได้คือ RDD `out` ที่มีสมาชิกเป็น `[2, 4]` เพราะเราเลือกเฉพาะตัวเลขคู่จาก RDD ต้นฉบับ[\*]การใช้ `distinct()` ใน Spark หมายถึงการสร้างชุดข้อมูลใหม่ใน RDD ที่ประกอบด้วยสมาชิกที่แตกต่างกันของชุดข้อมูลต้นฉบับ ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd1 = sc.parallelize([1, 2, 3, 2, 4, 3]){\*}>>> out = rdd1.distinct(){\*}>>> out.collect(){\*}[1, 2, 3, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd1` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 2, 4, 3]`{\*}- เราใช้ `distinct()` เพื่อระบุเฉพาะสมาชิกที่แตกต่างกันใน RDD ต้นฉบับ{\*}- ผลลัพธ์ที่ได้คือ RDD `out` ที่มีสมาชิกเป็น `[1, 2, 3, 4]` เนื่องจากมีการลบสมาชิกที่ซ้ำออกจาก RDD ต้นฉบับ[\*]การใช้ `flatMap(func)` ใน Spark คล้ายกับ `map` แต่แต่ละรายการนำเข้าสามารถถูกแปลงเป็น 0 หรือมากกว่า 0 รายการผลลัพธ์ (ดังนั้นฟังก์ชัน `func` ควรคืนค่าเป็นลำดับของรายการ) ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.map(lambda x: [x, x+5]){\*}>>> out.collect(){\*}[[1, 6], [2, 7], [3, 8], [4, 9]]{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> out = rdd.flatMap(lambda x: [x, x+5]){\*}>>> out.collect(){\*}[1, 6, 2, 7, 3, 8, 4, 9]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 4]`{\*}- เราใช้ `map()` ในตัวอย่างแรกเพื่อแปลงแต่ละสมาชิกใน RDD เป็นลิสต์ของสองตัวเลข (เช่น `[x, x+5]`){\*}- เราใช้ `flatMap()` ในตัวอย่างที่สองเพื่อแปลงแต่ละสมาชิกใน RDD เป็นรายการสองตัวเลขโดยตรง (ไม่ใช่ลิสต์ภายในลิสต์) และรวมรายการผลลัพธ์ทั้งหมดใน RDD และเรียงลำดับให้อยู่ในลำดับเดียวกันที่มีที่สอง.[\*]การใช้ `reduce(func)` ใน Spark คือการรวมสมาชิกในชุดข้อมูลของ RDD โดยใช้ฟังก์ชัน `func` ฟังก์ชันนี้รับอาร์กิวเมนต์สองตัวและคืนค่าเพียงหนึ่งค่า และเป็นการเปลี่ยนที่สามารถทำได้ในลักษณะสมมาตรและเชื่อมโยงเพื่อให้สามารถคำนวณได้อย่างถูกต้องในโหมดขนาน (ในคลัสเตอร์) ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.reduce(lambda a, b: a \* b){\*}24{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 4]`{\*}- เราใช้ `reduce()` เพื่อทำการคูณทุกคู่ของสมาชิกใน RDD เข้าด้วยกัน ซึ่งผลลัพธ์ที่ได้คือ 24 ที่เป็นผลคูณของ 1, 2, 3, และ 4 ทุกตัว[\*]การใช้ `take(n)` ใน Spark คือการดึงข้อมูล n รายการแรกออกจาก RDD และคืนเป็นอาร์เรย์ ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.take(2){\*}[1, 2]{\*}>>> rdd.take(5){\*}[1, 2, 3, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 4]`{\*}- เราใช้ `take(2)` เพื่อดึงสองรายการแรกของ RDD ซึ่งคือ `[1, 2]`{\*}- เราใช้ `take(5)` แต่ละรายการของ RDD โดยไม่ได้กำหนด n ซึ่งจะทำให้มันดึงทุกสมาชิกของ RDD และคืนค่าทั้งหมดในรูปของอาร์เรย์ `[1, 2, 3, 4]` [\*]การใช้ `collect()` ใน Spark คือการรวบรวมข้อมูลทั้งหมดใน RDD และคืนเป็นอาร์เรย์ ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.collect(){\*}[1, 2, 3, 4]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 4]`{\*}- เราใช้ `collect()` เพื่อรวบรวมทุกสมาชิกของ RDD และคืนค่าในรูปของอาร์เรย์ `[1, 2, 3, 4]` [\*]การใช้ `takeOrdered(n, key=func)` ใน Spark คือการดึงอาร์เรย์ของ n องค์ประกอบแรกของ RDD ในลำดับธรรมชาติของคีย์ที่ระบุโดย `func` ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.takeOrdered(3, lambda x: -x){\*}[4, 3, 2]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` จากการใช้ `sc.parallelize()` โดยมีสมาชิกคือ `[1, 2, 3, 4]`{\*}- เราใช้ `takeOrdered(3, lambda x: -x)` เพื่อดึงสามสมาชิกแรกของ RDD ในลำดับที่ถูกเรียงตามคีย์ที่กำหนดโดยฟังก์ชัน lambda `lambda x: -x` ซึ่งจะทำให้เรียงจากมากไปหาน้อยและคืนค่าในรูปของอาร์เรย์ `[4, 3, 2]` [\*]การใช้ `reduceByKey(func)` ใน Spark คือการรวบรวมคู่ (K, V) ใหม่ใน RDD ที่กระจายกัน โดยที่ค่าสำหรับแต่ละคีย์จะถูกรวมกันโดยใช้ฟังก์ชันลดทอนที่กำหนดโดย `func` ( (V, V) -> V ) ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([(1, 2), (3, 4), (3, 6), (1, 3), (3, 8)]){\*}>>> out = rdd.reduceByKey(lambda a, b: a + b){\*}>>> out.collect(){\*}[(1, 5), (3, 18)]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` โดยใช้ `sc.parallelize()` และมีคู่ (K, V) เป็น `[(1, 2), (3, 4), (3, 6), (1, 3), (3, 8)]`{\*}- เราใช้ `reduceByKey(lambda a, b: a + b)` เพื่อรวบรวมค่าของ V สำหรับแต่ละคีย์ โดยการรวมค่าในคู่ (K, V) เดียวกัน ดังนั้น เมื่อเราใช้ฟังก์ชันลดทอน `lambda a, b: a + b` ผลลัพธ์จะเป็นคู่ (K, V) ที่ค่า V คือผลรวมของค่า V ทั้งหมดสำหรับแต่ละคีย์ คืนค่าในรูปของ RDD ที่มีคู่ (K, V) เป็น `[(1, 5), (3, 18)]` [\*]ใน Spark, `sortByKey()` ใช้สำหรับเรียงลำดับคู่ (K, V) ใน RDD ตามคีย์ K ในลำดับเรียงจากน้อยไปหามาก{\*}ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]){\*}>>> out = rdd.sortByKey(){\*}>>> out.collect(){\*}[(1, 'c'), (1, 'b'), (3, 'd'), (3, 'a'), (3, 'e')]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` โดยใช้ `sc.parallelize()` และมีคู่ (K, V) เป็น `[(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]`{\*}- เราใช้ `sortByKey()` เพื่อเรียงลำดับคู่ (K, V) ตามคีย์ K ในลำดับเรียงจากน้อยไปหามาก ผลลัพธ์คืนค่าเป็น RDD ที่มีคู่ (K, V) เรียงลำดับแล้วเป็น `[(1, 'c'), (1, 'b'), (3, 'd'), (3, 'a'), (3, 'e')]` [\*]ใน Spark, `groupByKey()` ใช้สำหรับการจัดกลุ่มค่าของข้อมูลใน RDD โดยให้แต่ละกลุ่มมีคีย์ K เดียวกันและค่า V ที่สอดคล้องกันอยู่ในรูปของ iterable หรือลิสต์ของค่า{\*}ดังตัวอย่างด้านล่าง:{\*}```python{\*}>>> rdd = sc.parallelize([(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]){\*}>>> out = rdd.groupByKey(){\*}>>> out.map(lambda x: (x[0], list(x[1]))).collect(){\*}[(1, ['c', 'b']), (3, ['d', 'a', 'e'])]{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` โดยใช้ `sc.parallelize()` และมีคู่ (K, V) เป็น `[(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]`{\*}- เราใช้ `groupByKey()` เพื่อจัดกลุ่มค่าของ V ตามคีย์ K และคืน RDD ใหม่ที่มีคู่ (K, iterable<V>) ค่าที่กลุ่มรวมกัน{\*}- เราใช้ `map()` เพื่อแปลงค่าใน RDD ใหม่เป็นรูปแบบที่สะดวกต่อการอ่าน โดยแปลง iterable ในแต่ละคีย์ K เป็นลิสต์{\*}- สุดท้ายเราใช้ `collect()` เพื่อเก็บผลลัพธ์ทั้งหมดไว้ในรูปของลิสต์ที่รวบรวมทั้งหมดของคู่ (K, iterable<V>)[\*]ใน Spark, `count()` ใช้สำหรับนับจำนวนรายการข้อมูลใน RDD และคืนจำนวนรายการทั้งหมดใน RDD{\*}ตัวอย่างการใช้งาน:{\*}```python{\*}>>> rdd = sc.parallelize([(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]){\*}>>> rdd.count(){\*}5{\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง RDD `rdd` โดยใช้ `sc.parallelize()` และมีคู่ (K, V) เป็น `[(1, 'c'), (3, 'd'), (3, 'a'), (1, 'b'), (3, 'e')]`{\*}- เราใช้ `count()` เพื่อนับจำนวนรายการทั้งหมดใน RDD ซึ่งคืนค่าเป็นจำนวนรายการทั้งหมดที่มีใน RDD นั้นๆ ซึ่งในที่นี้คือ 5 รายการ[\*]ใน Spark, `foreach(func)` ใช้สำหรับการนำฟังก์ชั่น `func` มาประยุกต์ใช้กับแต่ละรายการใน RDD ที่กำหนด{\*}ตัวอย่างการใช้งาน:{\*}```python{\*}>>> def f(x):{\*}... print(x){\*}...{\*}>>> rdd = sc.parallelize([1, 2, 3, 4]){\*}>>> rdd.foreach(f){\*}```{\*}ในตัวอย่างนี้:{\*}- เรากำหนดฟังก์ชัน `f(x)` ซึ่งเป็นการพิมพ์ค่า `x` ออกทางหน้าจอ{\*}- เราสร้าง RDD `rdd` โดยใช้ `sc.parallelize()` และมีค่าเป็น `[1, 2, 3, 4]`{\*}- เราใช้ `foreach()` เพื่อใช้ `f` กับแต่ละรายการใน RDD `rdd` ซึ่งจะพิมพ์ค่าของแต่ละรายการออกทางหน้าจอ[\*]ใน Spark, ตัวแปร Broadcast ใช้สำหรับกระจายข้อมูลให้ทุกๆ worker node ในการทำงาน โดยที่ข้อมูลในตัวแปร Broadcast จะถูกส่งไปยังทุกๆ worker node และถูกเก็บไว้ในหน่วยความจำของแต่ละ worker node เพื่อให้สามารถเข้าถึงได้โดยตรง โดยไม่จำเป็นต้องส่งข้อมูลซ้ำๆ ในแต่ละครั้งที่มีการใช้งาน{\*}การใช้งาน Broadcast variables สามารถทำได้ดังนี้:{\*}ในโปรแกรมหลัก (driver program):{\*}```python{\*}b = sc.broadcast([1, 2, 3, 4]){\*}```{\*}ในฟังก์ชันหรือโปรแกรมย่อย (closure):{\*}```python{\*}def f(x):{\*}return b.value[x]{\*}rdd = sc.parallelize([0, 3]){\*}out = rdd.map(f){\*}print(out.collect()){\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง Broadcast variable `b` โดยใช้ `sc.broadcast()` และมีค่าเป็น `[1, 2, 3, 4]`{\*}- เรากำหนดฟังก์ชัน `f(x)` ใน closure ซึ่งใช้ Broadcast variable `b` เพื่อดึงค่าออกมาจาก index `x`{\*}- เราสร้าง RDD `rdd` โดยใช้ `sc.parallelize()` และมีค่าเป็น `[0, 3]`{\*}- เราใช้ `map()` เพื่อใช้ฟังก์ชัน `f` กับแต่ละรายการใน RDD `rdd` และใช้ `collect()` เพื่อเก็บผลลัพธ์ทั้งหมดที่ได้ไปยัง driver program และพิมพ์ผลลัพธ์ออกทางหน้าจอ[\*]ตัวแปร Accumulator ใน Spark ใช้สำหรับการนับหรือการสะสมค่าในขณะที่โปรแกรมกำลังทำงาน โดยสามารถเข้าถึงได้โดยง่ายและเป็นอัตโนมัติโดยทุก worker node ในกระบวนการการทำงาน Spark สามารถเพิ่มค่าในตัวแปร Accumulator ได้จากหลายๆ worker node โดยไม่ต้องกังวลเกี่ยวกับปัญหาการแข่งขันในการเขียนค่าที่มีอาจทำให้เกิดการขัดข้องของข้อมูล{\*}ในโปรแกรมหลัก (driver program):{\*}```python{\*}rdd = sc.parallelize([1, 2, 3, 4]){\*}c = sc.accumulator(0){\*}```{\*}ในฟังก์ชันหรือโปรแกรมย่อย (closure):{\*}```python{\*}def f(x):{\*}global c{\*}c += x{\*}rdd.foreach(f){\*}print(c.value){\*}```{\*}ในตัวอย่างนี้:{\*}- เราสร้าง Accumulator variable `c` โดยใช้ `sc.accumulator(0)` เพื่อสร้างตัวแปรที่มีค่าเริ่มต้นเป็น 0{\*}- เรากำหนดฟังก์ชัน `f(x)` ซึ่งเพิ่มค่า `x` เข้ากับตัวแปร Accumulator `c` โดยใช้ `global c` เพื่อให้ฟังก์ชันสามารถเข้าถึงตัวแปรนี้ได้{\*}- เราใช้ `foreach()` เพื่อใช้ฟังก์ชัน `f` กับแต่ละรายการใน RDD `rdd`{\*}- เมื่อกระบวนการการทำงาน Spark สิ้นสุดลง ค่าของตัวแปร Accumulator `c` สามารถเข้าถึงได้ผ่าน `.value` และจะได้ผลลัพธ์เป็น 10 ที่ได้จากการรวมค่าทั้งหมดของรายการใน RDD{\*}การใช้ Accumulator ใน Spark เป็นวิธีที่ดีในการนับหรือสะสมค่าในระหว่างการประมวลผลข้อมูลในสถานการณ์ที่ต้องการการแบ่งปันข้อมูลระหว่าง worker nodes โดยที่ไม่ต้องกังวลเกี่ยวกับการจัดการข้อมูลในการแบ่งแยกหรือการล็อกข้อมูลที่ซ้ำกันที่เกิดจากการประมวลผลแบบพรีเซ็นชั่นการแบ่งแยกแบบทวนสัญญาณที่ไม่เกี่ยวข้องกันใน worker nodes แต่ยังสามารถอ้างอิงถึงค่าผลลัพธ์ของการประมวลผลทั้งหมดได้ในตอนท้ายของการทำงาน นอกจากนี้ยังเป็นเครื่องมือที่มีประสิทธิภาพสำหรับการดูแลรักษาและการเก็บรวบรวมข้อมูลในกระบวนการการทำงานของ Spark อย่างมีประสิทธิภาพและที่มีประสิทธิภาพสูง ทำให้เขียนโปรแกรม Spark ได้อย่างสะดวกและเป็นประสบการณ์ที่ดีมากขึ้น[\*]สำหรับโปรแกรม Python "wordcount.py" นี้ เป็นโปรแกรมที่ใช้ Spark เพื่อนับจำนวนคำในไฟล์ข้อความที่ระบุ โดยใช้ RDDs ใน Spark ดังนี้:{\*}```python{\*}#!/usr/bin/env python{\*}# นำเข้า SparkContext จากไลบรารี pyspark{\*}from pyspark import SparkContext{\*}# สร้าง SparkContext ด้วยชื่อแอปพลิเคชัน "WordCount"{\*}sc =SparkContext(appName='WordCount'){\*}# โหลดไฟล์ข้อความจากเส้นทางที่ระบุ{\*}input\_file = sc.textFile('/user/cloudera/sw/test.txt'){\*}# แบ่งแยกคำโดยใช้ช่องว่างเป็นตัวแยก{\*}tokens = input\_file.flatMap(lambda line: line.split()){\*}# สร้างคู่ (คำ, 1) สำหรับแต่ละคำ{\*}words = tokens.map(lambda word: (word, 1)){\*}# รวมจำนวนคำทั้งหมดโดยรวมกันตามคำเดียวกัน{\*}wc = words.reduceByKey(lambda a, b: a + b){\*}# บันทึกผลลัพธ์เป็นไฟล์ข้อความ{\*}wc.saveAsTextFile('/user/cloudera/sw/output'){\*}# หยุด SparkContext{\*}sc.stop(){\*}```{\*}คำอธิบาย:{\*}- `SparkContext(appName='WordCount')` สร้าง SparkContext สำหรับการประมวลผลข้อมูล{\*}- `sc.textFile('/user/cloudera/sw/test.txt')` โหลดไฟล์ข้อความจากเส้นทางที่ระบุ{\*}- `flatMap(lambda line: line.split())` แยกแยะคำจากข้อความโดยใช้ช่องว่างเป็นตัวแยก{\*}- `map(lambda word: (word, 1))` สร้างคู่ (คำ, 1) สำหรับแต่ละคำ{\*}- `reduceByKey(lambda a, b: a + b)` รวมจำนวนคำทั้งหมดโดยรวมกันตามคำเดียวกัน{\*}- `saveAsTextFile('/user/cloudera/sw/output')` บันทึกผลลัพธ์เป็นไฟล์ข้อความในเส้นทางที่ระบุ{\*}- `sc.stop()` หยุด SparkContext เมื่อเสร็จสิ้นการทำงาน[\*]เพื่อรันโปรแกรม Python บน Spark จากไดเรกทอรีทำงาน คุณสามารถทำตามขั้นตอนดังนี้:{\*}1. เปิดหน้าเทอร์มินัล (Terminal) หรือโปรแกรมที่ใช้สั่งรันคำสั่ง{\*}2. เข้าสู่ไดเรกทอรีที่มีไฟล์ "wordcount.py" ตั้งอยู่{\*}3. ใช้คำสั่งต่อไปนี้เพื่อรันโปรแกรมบน Spark:{\*}```bash{\*}spark-submit --master local[\*] wordcount.py{\*}```{\*}โปรแกรมจะทำการประมวลผลและบันทึกผลลัพธ์เป็นไฟล์ข้อความในเส้นทางที่ระบุ หลังจากนั้นคุณสามารถตรวจสอบไฟล์ผลลัพธ์โดยใช้คำสั่งต่อไปนี้:{\*}```bash{\*}hadoop fs -ls /user/cloudera/sw/output{\*}hadoop fs -cat /user/cloudera/sw/output/part-00000{\*}```{\*}คำสั่งแรกจะแสดงรายการไฟล์ที่อยู่ในเส้นทาง "/user/cloudera/sw/output" และคำสั่งที่สองจะแสดงเนื้อหาของไฟล์ผลลัพธ์ part-00000 ที่อยู่ในเส้นทางนั้น ซึ่งเป็นผลการนับจำนวนคำในไฟล์ข้อความต้นฉบับที่ระบุในโปรแกรม "wordcount.py"[\*]SparkSQL[\*]DataFrame ใน Spark เป็นคอลเลกชันของแถวที่กระจายอยู่ในหลายๆ โหนดของข้อมูลภายใต้คอลัมน์ที่มีชื่อ มันเหมือนกับตารางในฐานข้อมูลระบบสัมพันธ์หรือชีท Excel ที่มีหัวข้อคอลัมน์{\*}มันยังมีลักษณะเดียวกับ RDD ด้วย:{\*}1. ลักษณะ Immutable: DataFrame / RDD เมื่อสร้างแล้วจะไม่สามารถเปลี่ยนแปลงได้ DataFrame / RDD สามารถแปลงเป็น DataFrame / RDD อื่นๆ หลังจากนำไปใช้งานโดยการใช้การแปลง{\*}2. Lazy Evaluations: หมายถึงงานจะไม่ถูกดำเนินการจนกว่าจะมีการกระทำที่เกิดขึ้น{\*}3. กระจายอยู่ในโหนดของข้อมูลทั้งหมด{\*}ดังนั้น DataFrame เป็นโครงสร้างข้อมูลที่สำคัญใน Spark ที่ใช้ในการประมวลผลข้อมูลแบบโครงสร้าง และมีประสิทธิภาพสำหรับการทำงานกับข้อมูลที่มีโครงสร้างในแบบที่มีคอลัมน์และแถวที่ถูกกำหนดชัดเจน[\*]Spark Data Frames ถูกออกแบบมาเพื่อประมวลผลชุดข้อมูลที่มีโครงสร้างหรือมีโครงสร้างบางส่วน{\*}• Spark Data Frames ถูกจัดเก็บในหน่วยความจำโดยใช้การจัดการหน่วยความจำที่กำหนดเอง (ต่างจาก RDD) ซึ่งช่วยให้มีการปรับปรุงการใช้งานต่างๆ ตัวอย่างเช่น ข้อมูลถูกจัดเก็บในหน่วยความจำแบบ off-heap ในรูปแบบที่เป็นไบนารี เนื่องจากสกีมาของหน่วยความจำเป็นที่รู้จัก ดังนั้นการเก็บรวบรวมขยะของ Java heap สำหรับข้อมูลถูกหลีกเลี่ยงและการเซียนระบบของ Java ที่ใช้สำหรับการกระจายข้อมูลที่เครือข่ายก็ถูกหลีกเลี่ยงเช่นกัน{\*}• ข้อมูลใน Spark Data Frames จะถูกจัดระเบียบภายใต้คอลัมน์ที่มีชื่อ ซึ่งช่วยให้ Spark เข้าใจโครงสร้างของ Data Frames นี้ ซึ่งช่วยให้ Spark ปรับปรุงแผนการประมวลผลในคำสั่งคิวรี่ และลดเวลาการดำเนินการคำสั่งคิวรี่{\*}• Data Frame สามารถเข้าถึงได้ผ่าน Spark SQL ซึ่งมีการสนับสนุน API สำหรับภาษาต่างๆ เช่น Python, R, Scala, Java.[\*]Data Frame เป็นโครงสร้างข้อมูลที่สำคัญใน Apache Spark ที่ช่วยในการจัดการข้อมูลในรูปแบบของตาราง (tabular data) โดยมีลักษณะคล้ายกับตารางในฐานข้อมูลแบบ relational database ซึ่งมีแถวและคอลัมน์ โดย Data Frame ใน Spark สามารถจัดการกับข้อมูลที่มีโครงสร้าง (structured data) หรือไม่มีโครงสร้าง (unstructured data) ได้ตามที่ต้องการ{\*}การเริ่มต้นใช้งาน SQLContext ใน Python เพื่อใช้งาน Data Frame ใน Spark สามารถทำได้ดังนี้:{\*}1. Import modules:{\*}```python{\*}from pyspark import SparkContext{\*}from pyspark.sql import SQLContext{\*}```{\*}2. สร้าง SparkContext:{\*}```python{\*}sc = SparkContext(){\*}```{\*}3. สร้าง SQLContext จาก SparkContext ที่สร้างขึ้น:{\*}```python{\*}sqlContext = SQLContext(sc){\*}```{\*}ดังนั้น SQLContext จะเป็นอินสแตนซ์ที่ใช้ในการเรียกใช้งาน Data Frame ใน Apache Spark ผ่านภาษา Python โดยอาจใช้งาน SQL หรือ Python API เพื่อประมวลผลข้อมูลในรูปแบบของ Data Frame ได้ตามต้องการในโปรแกรม Python ของคุณ[\*]การสร้าง DataFrame จาก RDD{\*}```python{\*}# สร้าง RDD จากข้อมูลของบุคคล{\*}person = [('Anna', 25, 'CA'), ('Jack', 22, 'TX'), ('Tom', 20, 'FL'), ('Bob', 26, 'NY'), ('Frank', 29, 'CA')]{\*}rdd = sc.parallelize(person){\*}# สร้าง DataFrame โดยระบุชื่อคอลัมน์{\*}df = sqlContext.createDataFrame(rdd, ['name', 'age', 'state']){\*}# ตรวจสอบประเภทของ DataFrame ที่สร้างและแสดงเนื้อหาของ DataFrame{\*}print(type(df)){\*}df.show(){\*}```{\*}ผลลัพธ์ที่คาดหวังคือการแสดง DataFrame แบบตารางที่มีชื่อ อายุ และรัฐของบุคคลทั้งหมดที่มีอยู่ใน RDD[\*]การสร้าง schema ใน PySpark ใช้คำสั่ง StructType และ StructField ซึ่งมีลักษณะดังนี้:{\*}1. \*\*StructType\*\*: สร้าง schema ที่ประกอบด้วย fields ต่างๆ โดยรับพารามิเตอร์เป็นลิสต์ของ StructField ที่ระบุ fields แต่ละตัวที่ต้องการใน schema.{\*}2. \*\*StructField\*\*: ใช้สำหรับกำหนดแต่ละ field ใน schema โดยระบุชื่อของ field, data type ของ field, และค่า boolean ที่ระบุว่า field นั้นสามารถมีค่า null ได้หรือไม่.{\*}ตัวอย่างการสร้าง schema ด้วยคำสั่ง StructType และ StructField ใน PySpark จะเป็นดังนี้:{\*}```python{\*}from pyspark.sql.types import \*{\*}# สร้าง schema ที่ประกอบด้วย fields ต่างๆ{\*}schema = StructType([{\*}# กำหนด field "ID" โดยมี data type เป็น IntegerType() และไม่สามารถมีค่า null ได้{\*}StructField("ID", IntegerType(), False),{\*}# กำหนด field "Name" โดยมี data type เป็น StringType() และสามารถมีค่า null ได้{\*}StructField("Name", StringType(), True),{\*}# กำหนด field "Age" โดยมี data type เป็น FloatType() และสามารถมีค่า null ได้{\*}StructField("Age", FloatType(), True){\*}]){\*}```{\*}ดังนั้นเราได้สร้าง schema ที่มี fields คือ "ID", "Name", และ "Age" โดยกำหนด data type และค่า boolean ที่ระบุว่า field นั้นสามารถมีค่า null ได้หรือไม่ ตามลำดับ[\*]ต่อไปนี้คือการสร้าง Data Frame จากไฟล์ข้อความโดยใช้ PySpark:{\*}```python{\*}from pyspark.sql.types import \*{\*}# อ่านข้อมูลจากไฟล์ข้อความและแปลงเป็น RDD{\*}rdd = sc.textFile('/user/cloudera/test.data') .map(lambda line: line.split(",")) .map(lambda line: [int(line[0]), int(line[1]), float(line[2])]){\*}# กำหนด schema สำหรับ Data Frame{\*}schema = StructType([{\*}StructField("x", IntegerType(), True),{\*}StructField("y", IntegerType(), True),{\*}StructField("z", FloatType(), True){\*}]){\*}# สร้าง Data Frame จาก RDD และ schema{\*}df = sqlContext.createDataFrame(rdd, schema){\*}# แสดง schema ของ Data Frame{\*}df.printSchema(){\*}```{\*}ในส่วนของโค้ดด้านบนมีขั้นตอนการสร้าง Data Frame จากไฟล์ข้อความที่ชื่อว่า 'test.data':{\*}1. อ่านข้อมูลจากไฟล์ข้อความและแปลงเป็น RDD โดยใช้ `sc.textFile()` และ `map()` เพื่อแยกข้อมูลในแต่ละบรรทัดและแยกตามเครื่องหมาย ','{\*}2. กำหนด schema ของ Data Frame โดยใช้ `StructType` และ `StructField` เพื่อระบุชื่อและประเภทของแต่ละ column และระบุว่าสามารถมีค่า null ได้หรือไม่{\*}3. สร้าง Data Frame จาก RDD และ schema โดยใช้ `sqlContext.createDataFrame()`{\*}4. แสดง schema ของ Data Frame โดยใช้ `printSchema()`[\*]ด้านล่างนี้คือตัวอย่างของการดำเนินการพื้นฐานกับ DataFrame ใน PySpark:{\*}1. \*\*show(n)\*\*: ใช้เพื่อแสดงข้อมูลของ DataFrame โดยจะแสดงแถวแรกถึงแถวที่ n ของ DataFrame{\*}```python{\*}df.show(5){\*}```{\*}2. \*\*count()\*\*: ใช้เพื่อนับจำนวนแถวใน DataFrame{\*}```python{\*}df.count(){\*}```{\*}3. \*\*select(cols)\*\*: ใช้เพื่อเลือกเฉพาะคอลัมน์ที่ต้องการจาก DataFrame เพื่อสร้าง DataFrame ใหม่{\*}```python{\*}df.select(['name', 'age']).show(){\*}```{\*}ตัวอย่างเหล่านี้สามารถใช้เพื่อทำงานกับ DataFrame และดำเนินการดูข้อมูลและการวิเคราะห์ข้อมูลพื้นฐานอื่นๆ ได้อย่างมีประสิทธิภาพใน PySpark.[\*]นี่คือตัวอย่างของการใช้งาน DataFrame Basic Operations ใน PySpark:{\*}1. \*\*orderBy(cols, ascending)\*\*: ใช้สร้าง DataFrame ใหม่ที่เรียงลำดับข้อมูลตามคอลัมน์ที่ระบุใน cols โดย ascending เป็นอาร์กิวเมนต์ที่ระบุลำดับการเรียงข้อมูลโดยปริยายเป็น ascending (True) หรือ descending (False){\*}```python{\*}df.orderBy(['age'], ascending=False).show(){\*}```{\*}การใช้งานดังกล่าวจะสร้าง DataFrame ใหม่ที่มีข้อมูลเรียงลำดับตามคอลัมน์ 'age' ในลำดับจากมากไปน้อยและแสดงผลลัพธ์ในรูปแบบของ DataFrame ที่เรียงลำดับแล้วโดยมีค่า 'age' ที่มากที่สุดอยู่ด้านบนของตารางรูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ไลน์, ตัวอักษร

Description automatically generated[\*]นี่คือตัวอย่างของการใช้งาน DataFrame Basic Operations ใน PySpark:{\*}1. \*\*groupBy(cols)\*\*: ใช้สร้าง DataFrame ใหม่ที่มีข้อมูลที่กลุ่มตามคอลัมน์ที่ระบุใน cols{\*}```python{\*}df.groupBy(['state']).avg('age').show(){\*}```{\*}ในตัวอย่างด้านบน เรากำลังกลุ่มข้อมูลใน DataFrame ตามคอลัมน์ 'state' และคำนวณค่าเฉลี่ยของคอลัมน์ 'age' สำหรับแต่ละกลุ่ม และแสดงผลลัพธ์ในรูปแบบของ DataFrame ที่มีคอลัมน์ 'state' และ 'avg(age)' ซึ่งเป็นค่าเฉลี่ยของอายุสำหรับแต่ละสถานะที่แตกต่างกัน[\*]นี่คือตัวอย่างของการใช้งาน DataFrame Basic Operations ใน PySpark:{\*}1. \*\*filter(cond)\*\*: ใช้เพื่อนำเงื่อนไขที่ระบุ (cond) มาใช้งานกับ DataFrame และสร้าง DataFrame ที่ผลลัพธ์มีเฉพาะข้อมูลที่ผ่านเงื่อนไขนั้น{\*}```python{\*}df.filter('age > 25').show(){\*}```{\*}ในตัวอย่างข้างบน เรากำลังกรองข้อมูลใน DataFrame โดยเลือกเฉพาะแถวที่มีค่าในคอลัมน์ 'age' มากกว่า 25 และแสดงผลลัพธ์ในรูปแบบของ DataFrame ที่มีข้อมูลที่ผ่านเงื่อนไขนี้เท่านั้น[\*]สำหรับชุดคำสั่งด้านล่างนี้:{\*}1. \*\*registerAsTable(table\_name)\*\*: ใช้เพื่อลงทะเบียน DataFrame ที่กำหนดให้เป็นตารางใน Spark SQL ด้วยชื่อตารางที่กำหนด{\*}```python{\*}df.registerAsTable('person\_table'){\*}```{\*}2. หลังจากที่ลงทะเบียน DataFrame แล้ว เราสามารถใช้ Spark SQL เพื่อประมวลผลข้อมูลในตารางได้ โดยใช้คำสั่ง `sqlContext.sql()` เพื่อรันคำสั่ง SQL และ `show()` เพื่อแสดงผลลัพธ์{\*}```python{\*}sqlContext.sql('select name,state from person\_table').show(5){\*}```{\*}การทำแบบนี้จะทำให้เราสามารถใช้งานข้อมูลใน DataFrame ด้วยภาษา SQL ได้ โดยใช้ชื่อตารางที่ลงทะเบียนไว้และประมวลผลข้อมูลในรูปแบบของ DataFrame ใหม่ที่ได้จากการรันคำสั่ง SQL นั้นๆ โดยใช้ฟังก์ชัน `sqlContext.sql()` ใน PySpark[\*]Spark and Data Frame Lab[\*]การสร้าง DataFrame จาก RDD เป็นกระบวนการที่ใช้ใน PySpark เพื่อสร้าง DataFrame จาก RDD ที่มีข้อมูลอยู่แล้ว ดังตัวอย่างด้านล่างนี้:{\*}```python{\*}# สร้าง RDD จากข้อมูลบุคคล{\*}person = [('Anna', 25, 'CA'), ('Jack', 22, 'TX'), ('Tom', 20, 'FL'), ('Bob', 26, 'NY'), ('Frank', 29, 'CA')]{\*}rdd = sc.parallelize(person){\*}# สร้าง DataFrame จาก RDD โดยระบุชื่อคอลัมน์{\*}df = sqlContext.createDataFrame(rdd, ['name', 'age', 'state']){\*}# ตรวจสอบประเภทของ DataFrame ที่สร้างและแสดงเนื้อหาของ DataFrame และสกีมาของมัน{\*}print(type(df)){\*}df.show(){\*}df.printSchema(){\*}```{\*}ในขั้นตอนดังกล่าว:{\*}- เริ่มต้นโดยการสร้าง RDD จากข้อมูลบุคคลที่กำหนดไว้ล่วงหน้า{\*}- จากนั้นใช้ `parallelize` เพื่อสร้าง RDD จากข้อมูลบุคคลที่กำหนดไว้{\*}- ใช้ `createDataFrame` เพื่อสร้าง DataFrame จาก RDD ที่ได้ โดยระบุชื่อคอลัมน์ที่ต้องการ{\*}- ทำการตรวจสอบประเภทของ DataFrame ที่สร้างขึ้นและแสดงเนื้อหาของ DataFrame และสกีมาของมันโดยใช้ `show()` และ `printSchema()` ตามลำดับ[\*]การดำเนินการพื้นฐานของ DataFrame ใน PySpark มีดังนี้:{\*}1. \*\*show(n)\*\*: ใช้เพื่อพิมพ์แถวแรก n ของ DataFrame{\*}```python{\*}df.show(5){\*}```{\*}ตัวอย่างนี้จะแสดงแถวแรก 5 ของ DataFrame{\*}2. \*\*count()\*\*: ใช้เพื่อคืนค่าจำนวนแถวใน DataFrame{\*}```python{\*}df.count(){\*}```{\*}จะคืนค่าจำนวนแถวทั้งหมดใน DataFrame{\*}3. \*\*select(cols)\*\*: ใช้สร้าง DataFrame ใหม่จากลิสต์ของคอลัมน์ที่ระบุใน cols{\*}```python{\*}df.select(['name', 'age']).show(){\*}```{\*}จะคืน DataFrame ที่มีเฉพาะคอลัมน์ 'name' และ 'age' และแสดงข้อมูลทั้งหมดในคอลัมน์ดังกล่าว{\*}คำสั่งด้านบนเป็นตัวอย่างของการใช้งาน DataFrame Basic Operations ใน PySpark:{\*}1. \*\*show(n)\*\*: การใช้คำสั่ง `show(5)` จะแสดงข้อมูลใน DataFrame โดยแสดงแถวแรก 5 แถว{\*}2. \*\*count()\*\*: คำสั่ง `count()` จะนับจำนวนแถวทั้งหมดใน DataFrame และคืนค่านั้นออกมา{\*}3. \*\*select(cols)\*\*: การใช้คำสั่ง `select(['name', 'age'])` จะสร้าง DataFrame ใหม่ที่มีเฉพาะคอลัมน์ 'name' และ 'age' และแสดงข้อมูลทั้งหมดในคอลัมน์ดังกล่าว[\*]คำสั่งนี้ใช้สร้าง DataFrame ใหม่ที่เรียงลำดับข้อมูลตามคอลัมน์ที่ระบุใน `cols` โดยค่า `ascending` เป็นอาร์กิวเมนต์ที่ระบุว่าจะเรียงลำดับให้เกิดขึ้นในลำดับที่เพิ่มขึ้นหรือลดลง (เรียงจากน้อยไปมากหรือจากมากไปน้อย) โดยค่าเริ่มต้นคือเรียงลำดับจากน้อยไปมาก (ascending=True){\*}การใช้งานด้วย PySpark จะดูประมาณนี้:{\*}```python{\*}df.orderBy(['age'], ascending=False).show(){\*}```{\*}คำสั่งนี้จะสร้าง DataFrame ใหม่ที่เรียงลำดับข้อมูลตามคอลัมน์ 'age' ในลำดับที่มากไปน้อยและแสดงผลลัพธ์ที่ได้ในรูปแบบของตาราง[\*]คำสั่ง `groupBy(cols)` ใช้สร้าง DataFrame ใหม่ที่มีการจัดกลุ่มข้อมูลตามคอลัมน์ที่ระบุใน `cols` โดยทั่วไปจะตามด้วยการดำเนินการรวมข้อมูล (aggregate operation) เช่น `count()` หรือ `avg()` หรือ `sum()` เป็นต้น เพื่อทำการประมวลผลบนกลุ่มข้อมูลที่เกิดขึ้น{\*}สำหรับตัวอย่างนี้:{\*}```python{\*}df.groupBy(['state']).avg('age').show(){\*}```{\*}การเรียกใช้คำสั่งนี้จะสร้าง DataFrame ใหม่ที่กลุ่มข้อมูลตามคอลัมน์ 'state' และคำนวณค่าเฉลี่ยของคอลัมน์ 'age' สำหรับแต่ละกลุ่ม และแสดงผลลัพธ์ที่ได้ในรูปแบบของตาราง[\*]คำสั่ง `filter(cond)` ใช้สร้าง DataFrame ใหม่โดยใช้เงื่อนไขที่กำหนด (cond) เพื่อกรองข้อมูลใน DataFrame ที่กำหนด และคืนค่า DataFrame ที่ผลลัพธ์จากการกรองนั้น{\*}ตัวอย่างการใช้งาน:{\*}```python{\*}df.filter('age > 25').show(){\*}```{\*}คำสั่งนี้จะกรองข้อมูลใน DataFrame `df` โดยเลือกเฉพาะแถวที่มีค่าในคอลัมน์ 'age' มากกว่า 25 และแสดงผลลัพธ์ในรูปแบบของตาราง[\*]คำสั่ง `registerAsTable(table\_name)` ใช้ในการลงทะเบียน DataFrame ที่กำหนดให้เป็นตารางด้วยชื่อตารางที่กำหนด ดังนั้นเราสามารถนำ SQL queries มาใช้กับตารางนั้นได้{\*}ตัวอย่างการใช้งาน:{\*}```python{\*}df.registerAsTable('person\_table'){\*}sqlContext.sql('select name,state from person\_table').show(5){\*}```{\*}คำสั่งนี้จะลงทะเบียน DataFrame `df` เป็นตารางที่ชื่อว่า 'person\_table' และจะดึงข้อมูลชื่อและรัฐจากตาราง 'person\_table' และแสดงผลลัพธ์ 5 แถวแรกของข้อมูลที่ได้จากการ query นี้ในรูปแบบของตาราง[\*]ในขั้นตอนแรกคือการดาวน์โหลดไฟล์ข้อความ test.data จาก MS Teams ไปยังโฟลเดอร์ที่แชร์และคัดลอกไฟล์ไปยังไดเร็กทอรีทำงาน หลังจากนั้นคุณสามารถทำต่อได้ตามคำสั่งดังต่อไปนี้:{\*}1. ใช้คำสั่ง `hadoop fs –put test.data /user/cloudera` เพื่ออัปโหลดไฟล์ test.data ไปยังโฟลเดอร์ /user/cloudera ใน Hadoop Distributed File System (HDFS){\*}2. เรียกใช้ pyspark โดยพิมพ์คำสั่ง `pyspark` ใน terminal เพื่อเริ่มต้น Spark shell{\*}เมื่อคุณเรียกใช้ Spark shell สามารถดำเนินการสร้าง DataFrame จากไฟล์ test.data ได้ดังนี้:{\*}```python{\*}from pyspark.sql import SparkSession{\*}# สร้าง SparkSession{\*}spark = SparkSession.builder .appName("Create DataFrame from text file") .getOrCreate(){\*}# อ่านไฟล์ข้อความและสร้าง DataFrame{\*}df = spark.read.text('/user/cloudera/test.data'){\*}# แสดง Schema และข้อมูลของ DataFrame{\*}df.printSchema(){\*}df.show(){\*}```{\*}ดังนั้นคุณจะสามารถสร้าง DataFrame จากไฟล์ test.data และแสดงข้อมูลที่อ่านเข้ามาได้ใน Spark shell ของคุณ[\*]การสร้าง DataFrame จากไฟล์ข้อความใน Spark สามารถทำได้ด้วยการใช้คำสั่ง `textFile` เพื่ออ่านข้อมูลจากไฟล์ข้อความ และหลังจากนั้นใช้ `map` เพื่อแยกข้อมูลและแปลงเป็นรูปแบบที่ต้องการ จากนั้นกำหนด schema ของ DataFrame ด้วย `StructType` และ `StructField` และสร้าง DataFrame ด้วย `createDataFrame` ดังนี้:{\*}```python{\*}from pyspark.sql.types import \*{\*}# อ่านข้อมูลจากไฟล์ข้อความ{\*}rdd = sc.textFile('/user/cloudera/test.data'){\*}# แยกข้อมูลและแปลงเป็นรูปแบบที่ต้องการ{\*}rdd = rdd.map(lambda line: line.split(",")).map(lambda line: [int(line[0]), int(line[1]), float(line[2])]){\*}# กำหนด schema ของ DataFrame{\*}schema = StructType([{\*}StructField("x", IntegerType(), True),{\*}StructField("y", IntegerType(), True),{\*}StructField("z", FloatType(), True){\*}]){\*}# สร้าง DataFrame ด้วย schema{\*}df = sqlContext.createDataFrame(rdd, schema){\*}# แสดง DataFrame{\*}df.show(){\*}```{\*}ดังนั้น DataFrame `df` จะถูกสร้างขึ้นโดยใช้ข้อมูลจากไฟล์ข้อความและ schema ที่กำหนด และข้อมูลใน DataFrame จะถูกแสดงผลด้วย `show()` ซึ่งแสดงค่าเริ่มต้นของ DataFrame{\*}โดยการใช้ `sc.textFile()` เราสามารถอ่านไฟล์ข้อความและแยกข้อมูลด้วย `,` และแปลงเป็นชนิดข้อมูลที่เหมาะสม เช่น IntegerType หรือ FloatType ได้แล้ว จากนั้นเรากำหนด schema ของ DataFrame ด้วย `StructType` และ `StructField` และสร้าง DataFrame ด้วย `createDataFrame` โดยใช้ schema ที่กำหนดไว้ และใช้ `show()` เพื่อแสดงข้อมูลแรกๆ ใน DataFrame ที่ถูกสร้างขึ้นแล้ว ซึ่งทำให้เราสามารถตรวจสอบข้อมูลได้ง่ายและตรวจสอบว่าข้อมูลถูกอ่านและแปลงได้อย่างถูกต้องหรือไม่{\*}การใช้คำสั่ง `sc.textFile()` ใน Spark ทำให้เราสามารถอ่านไฟล์ข้อความได้ และเราใช้ `map()` เพื่อแยกข้อมูลในแต่ละบรรทัดและแปลงเป็นรูปแบบที่เหมาะสม ในที่นี้คือแยกแต่ละบรรทัดตามเครื่องหมาย `,` และแปลงข้อมูลให้อยู่ในรูปแบบของ IntegerType หรือ FloatType ตามความเหมาะสม{\*}เมื่อเรามี RDD ที่มีข้อมูลที่ถูกแยกและแปลงแล้ว เราสามารถกำหนด schema ของ DataFrame ด้วย `StructType` และ `StructField` โดยระบุชื่อของฟิลด์และชนิดของข้อมูล และสร้าง DataFrame โดยใช้ `createDataFrame` กับ RDD และ schema ที่กำหนดไว้{\*}เมื่อ DataFrame ถูกสร้างขึ้นแล้ว เราใช้ `show()` เพื่อแสดงข้อมูลแรกๆ ใน DataFrame เพื่อตรวจสอบว่าข้อมูลถูกอ่านและแปลงได้อย่างถูกต้องหรือไม่[\*]Machine Learning[\*]Machine Learning - คือการศึกษาหรือการเรียนรู้ของเครื่อง ซึ่งเป็นกระบวนการที่ใช้ในการค้นหาแนวโน้มและความสัมพันธ์ระหว่างตัวแปรหรือข้อมูลภายในชุดข้อมูล{\*}- ผลลัพธ์จากการเรียนรู้ของเครื่องสามารถเป็นโมเดลที่สามารถจับต้องแนวโน้มหรือความสัมพันธ์ที่มีอยู่ในข้อมูลได้{\*}- สำหรับการเรียนรู้ของเครื่องแบบ Supervised (การเรียนรู้ที่มีผู้สอน) โมเดลจะอธิบายความสัมพันธ์ระหว่างตัวแปรต้นและตัวแปรตาม และสามารถใช้ในการทำนายค่าของตัวแปรตามโดยอิงจากค่าของตัวแปรต้นที่กำหนดไว้ให้[\*]สำหรับ Machine Learning แบบไม่มีการกำหนดค่าที่สอดคล้องกับความรู้สึก เราใช้โมเดลเพื่อแสดงรูปแบบหรือโครงสร้างที่ซ่อนอยู่ภายในชุดข้อมูล เช่น การรวมกลุ่มข้อมูลหรือกฎการเชื่อมโยง โมเดลดังกล่าวมักจะช่วยให้เราเข้าใจข้อมูลอย่างลึกซึ้งและค้นพบความสัมพันธ์หรือโครงสร้างที่ไม่เป็นทางการที่อาจไม่เห็นได้ง่าย{\*}ลักษณะหรือตัวแปรของการสังเกตการณ์สามารถเป็นแบบหมวดหมู่ (กลุ่ม, ไม่กำหนดหรือกำหนดลำดับ) หรือเป็นตัวเลข (หมวดหมู่หรือต่อเนื่อง){\*}สำหรับ Machine Learning แบบกำหนดค่าที่สอดคล้องกับความรู้สึก ชุดข้อมูลที่กำหนดให้ต้องถูกแบ่งเป็นสองชุดย่อย ดังนี้{\*}1. ชุดข้อมูลการฝึกถูกใช้เพื่อฝึกโมเดลให้สามารถทำนายคลาสหรือค่าตัวแปรตามได้{\*}2. ชุดข้อมูลการทดสอบถูกใช้เพื่อตรวจสอบหรือประเมินประสิทธิภาพในการทำนายของโมเดลที่ฝึกแล้ว โดยการใช้ข้อมูลทดสอบให้ผลการทำนายที่เป็นอิสระจากข้อมูลฝึกช่วยให้เราทราบถึงประสิทธิภาพทั้งหมดของโมเดลที่ได้จากการฝึก[\*]การประยุกต์ใช้ Machine Learning สามารถทำได้ในหลายแง่มุมต่าง ๆ ต่อไปนี้คือการประยุกต์ใช้ที่พบบ่อย:{\*}1. \*\*Classification (การจำแนกประเภท)\*\*: การสร้างโมเดลเพื่อทำนายป้ายกำกับคลาสของการสังเกตการณ์ที่กำหนดให้ เช่น การจำแนกอีเมลเป็นอีเมลขยะหรือไม่ขยะ การจำแนกภาพเป็นแต่ละประเภทของวัตถุ เป็นต้น{\*}2. \*\*Regression (การถดถอย)\*\*: การสร้างโมเดลเพื่อทำนายค่าตัวเลขหรือตัวแปรที่ขึ้นอยู่กับค่าอื่น ๆ โดยการใช้ข้อมูลที่มีอยู่ เช่น การทำนายราคาของบ้านโดยใช้ข้อมูลเกี่ยวกับพื้นที่ จำนวนห้องนอน และสิ่งอำนวยความสะดวกอื่น ๆ{\*}3. \*\*Clustering (การจัดกลุ่ม)\*\*: การแบ่งชุดข้อมูลเป็นกลุ่มหรือส่วนที่กำหนดไว้ล่วงหน้า โดยแต่ละกลุ่มมีการสังเกตการณ์ที่คล้ายคลึงกันมากกว่ากับกลุ่มอื่น ๆ เช่น การจัดกลุ่มลูกค้าตามพฤติกรรมการซื้อสินค้า การจัดกลุ่มข่าวในสื่อออนไลน์ตามหัวข้อที่เกี่ยวข้อง หรือการจัดกลุ่มการศึกษาในวิชาที่มีความเหมือนกัน[\*]Machine Learning (การเรียนรู้ของเครื่อง){\*}1. การตรวจจับความผิดปกติ (Anomaly Detection): การตรวจจับสถานการณ์ที่ผิดปกติหรือข้อมูลที่ออกนอกเกณฑ์จากชุดข้อมูลที่กำหนด{\*}2. การแนะนำ (Recommendation):{\*}- การกรองร่วม (Collaborative Filtering): การพยากรณ์คะแนนที่ไม่ทราบด้วยการใช้คะแนนที่ทราบแล้ว แล้วแนะนำผลิตภัณฑ์ที่มีคะแนนที่พยากรณ์ได้สูง{\*}- การกรองเนื้อหา (Content-based): การใช้คุณสมบัติที่ระบุโดยชัดเจนเพื่อค้นหาผลิตภัณฑ์ที่คล้ายกันและจากนั้นทำการแนะนำสิ่งที่เหมาะสม[\*]Machine Learning (การเรียนรู้ของเครื่อง){\*}3. การลดขนาดของมิติ (Dimension Reduction): การลดจำนวนของคุณลักษณะหรือคุณสมบัติที่มีความสำคัญน้อยหรือไม่จำเป็นออกจากชุดข้อมูล หรือแปลงชุดข้อมูลใหม่ให้มีมิติน้อยลง โดยที่ยังคงคุณสมบัติหลักของชุดข้อมูลเดิมไว้สำหรับการเรียนรู้ของเครื่อง[\*]\*\*การเรียนรู้เครื่องจักรด้วย Spark\*\*{\*}Spark มีสองไลบรารีสำหรับการเรียนรู้เครื่องจักรดังนี้:{\*}1. \*\*MLlib (คลาสสิกมากขึ้น)\*\*: MLlib เป็นไลบรารีที่เก่าแก่กว่าและมีความสมบูรณ์มากกว่า มันใช้ RDD (Resilient Distributed Dataset) เพื่อเก็บและประมวลผลข้อมูลในหน่วยความจำในรูปแบบคลัสเตอร์ที่มีหลายๆ โหนด{\*}2. \*\*Spark ML (หรือเรียกอีกอย่างว่า Pipeline API)\*\*: Spark ML เป็นไลบรารีที่ใหม่ของ Spark ซึ่งเน้นไปที่การใช้งานโครงสร้างข้อมูล Data frames เพื่อเก็บและประมวลผลข้อมูลในรูปแบบคลัสเตอร์ที่มีหลายๆ โหนด โดย Data frame เป็นโครงสร้างข้อมูลที่นำเสนอเพื่อเก็บชุดข้อมูลที่มีโครงสร้างหรือครึ่งโครงสร้างในรูปแบบคอลัมน์{\*}ทั้ง MLlib และ Spark ML เป็นไลบรารีที่สามารถทำงานกับชุดข้อมูลขนาดใหญ่ได้ โดยสามารถทำงานร่วมกับการประมวลผลข้อมูลแบบการกระจายในรูปแบบคลัสเตอร์ที่มีหลายๆ โหนด ซึ่งทำให้เหมาะสำหรับการประมวลผลข้อมูลขนาดใหญ่ที่ต้องการประสิทธิภาพและการขยายขนาดได้แม่นยำ[\*]\*\*MLlib\*\*{\*}MLlib มี API ระดับสูงสำหรับการเรียนรู้เครื่องจักรและการวิเคราะห์ทางสถิติ (เช่น การวิเคราะห์สถิติสรุป, การความสัมพันธ์, การสุ่มตัวอย่างแบบต่างๆ, การทดสอบสมมติฐาน, การสร้างข้อมูลสุ่ม, การประเมินความหนาแน่นของ kernel) API มีให้ใช้ใน Scala, Java, Python และ R{\*}มันสามารถใช้สำหรับงานการเรียนรู้เครื่องจักรทั่วไป เช่น{\*}1. การทำนายและการจำแนกประเภท เช่น การถดถอยเชิงเส้น, การถดถอยโลจิสติกส์, เครื่องรู้สนับสนุนเวกเตอร์, นายพายฉนวน, ต้นไม้การตัดสินใจ, ป่าสุ่ม, การถดถอยแบบเพิ่มเติม, การถดถอยแบบไอโซโทนิก[\*]MLlib{\*}2. การแยกข้อมูลและการแปลง{\*}- TF-IDF{\*}- Word2Vec{\*}- การปรับค่ามาตรฐาน (Standard Scaler){\*}- Normalizer{\*}- การเลือกคุณลักษณะด้วยความถี่ของคำ (Chi-Squared Feature Selection){\*}- ผลคูณแบบอิสระต่อช่องทาง (Elementwise Product)[\*]MLlib{\*}3. การทำค้นหารูปแบบที่เป็นที่นิยม{\*}- FP-growth{\*}- กฎการเชื่อมโยง (Association Rules){\*}- การทำ Prefix Span (Sequential Pattern Mining){\*}4. การแนะนำ{\*}- การกรองแบบร่วมมือโดยใช้ ALS (Collaborative Filtering using ALS){\*}5. การจัดกลุ่ม{\*}- K means[\*]MLlib ประเภทข้อมูล{\*}- Vector: เวกเตอร์ที่ใช้เก็บข้อมูลตัวเลข{\*}- Labeled Point: จุดที่มีป้ายชื่อซึ่งเป็นคู่ระหว่างค่าตัวแปรอิสระและค่าตัวแปรตาม{\*}- Rating: ข้อมูลการจัดอันดับที่ใช้ในการประเมินความชอบที่ผู้ใช้ให้กับรายการที่แนะนำ[\*]MLlib ประเภทเวกเตอร์{\*}ประเภทเวกเตอร์ใน MLlib แทนการเก็บค่าแบบ double precision floating point ที่มีลำดับที่ตั้งเริ่มที่ 0 ของประเภท int (จำนวนเต็ม){\*}เวกเตอร์สามารถแทนจุดในพื้นที่ n มิติได้{\*}MLlib รองรับเวกเตอร์สองประเภท: dense (หนาแน่น) และ sparse (หนาแสดง)[\*]MLlib ประเภทเวกเตอร์{\*}• DenseVector - อินสแตนซ์ของคลาส DenseVector เก็บค่าประเภท double ที่ทุกตำแหน่งดัชนี{\*}• เวกเตอร์หนาแน่นถูกใช้โดยทั่วไปถ้าชุดข้อมูลไม่มีค่าศูนย์มากเกินไป{\*}```scala{\*}import org.apache.spark.mllib.linalg.\_{\*}val denseVector = Vectors.dense(1.0, 0.0, 3.0){\*}```{\*}วิธี dense สร้างอินสแตนซ์ของคลาส DenseVector จากค่าที่ให้เป็นอาร์กิวเมนต์ (ดังนั้นเวกเตอร์จะไม่ยาวมาก)[\*] MLlib ประเภทของเวกเตอร์:{\*}- DenseVector (เวกเตอร์หนาแน่น): เป็นเวอร์ชันหนึ่งของเมทอด dense ที่รับอาร์เรย์ของชนิด Double เป็นอาร์กิวเมนต์และคืนค่าเป็นอินสแตนซ์ของคลาส DenseVector{\*}- SparseVector (เวกเตอร์โดยส่วนใหญ่): เป็นเวกเตอร์ที่มีค่าศูนย์มาก ซึ่งจัดเก็บค่าเฉพาะที่ไม่เท่ากับศูนย์เท่านั้น เป็นชนิดข้อมูลที่มีประสิทธิภาพในการจัดเก็บชุดข้อมูลขนาดใหญ่ที่มีค่าศูนย์มากมาย{\*}- อินสแตนซ์ของคลาส SparseVector มีการรองรับโดยอาร์เรย์สองชุด คือ อาร์เรย์หนึ่งเก็บดัชนีของค่าที่ไม่เท่ากับศูนย์ และอีกหนึ่งเก็บค่าที่ไม่เท่ากับศูนย์ในลำดับเดียวกัน[\*]ข้อมูลเวกเตอร์ของ MLlib{\*}สามารถสร้าง SparseVector ได้โดยใช้เมทอด sparse โดยระบุขนาดของเวกเตอร์ และลิสต์ของคู่ลำดับ (index, value) ที่ไม่เป็นศูนย์{\*}ใน Scala:{\*}```scala{\*}import org.apache.spark.mllib.linalg.\_{\*}val sparseVector = Vectors.sparse(10, Array(3, 6), Array(100.0, 200.0)){\*}```{\*}หรือใช้วิธีอื่นได้ดังนี้{\*}```scala{\*}import org.apache.spark.mllib.linalg.\_{\*}val sparseVector = Vectors.sparse(10, Seq((3, 100.0), (6, 200.0))){\*}```{\*}โดยที่ Seq คือลิสต์ของคู่ลำดับ (index, value) ที่ไม่เป็นศูนย์[\*]สำหรับ PySpark สามารถสร้างเวกเตอร์ได้ดังนี้:{\*}```python{\*}from pyspark.mllib.linalg import Vectors{\*}# สร้าง DenseVector{\*}denseVec = Vectors.dense([1.0, 2.0, 3.0]){\*}# สร้าง SparseVector{\*}sparseVec = Vectors.sparse(4, {1: 1.0, 3: 5.5}){\*}```{\*}ในที่นี้ `4` คือขนาดของเวกเตอร์ และ `{1: 1.0, 3: 5.5}` คือคู่ลำดับ (index, value) ที่ไม่เป็นศูนย์สำหรับค่าที่ไม่เป็นศูนย์ใน SparseVector ที่แทนด้วย `1.0` ที่ตำแหน่งที่ 1 และ `5.5` ที่ตำแหน่งที่ 3 ตามลำดับโดยมีขนาด 4 ของ SparseVector[\*]MLlib ประเภทของ LabeledPoint:{\*}- ประเภท LabeledPoint แทนข้อมูลตัวอย่างในชุดข้อมูลที่มีป้ายชื่อ{\*}- มันประกอบด้วยทั้งป้ายชื่อ (ตัวแปรตาม) และคุณลักษณะ (ตัวแปรอิสระ) ของข้อมูลตัวอย่าง{\*}- ป้ายชื่อถูกเก็บเป็นค่าชนิด Double และคุณลักษณะถูกเก็บเป็นประเภท Vector[\*]ชนิดข้อมูล LabeledPoint ใน MLlib ประกอบด้วยค่า label ซึ่งเป็นค่าแบบ double ซึ่งสามารถแทนค่าทั้งตัวเลขและแบบจำลองได้:{\*}- สำหรับขั้นตอนการ Regression ค่า label เก็บค่าตัวเลข{\*}- สำหรับการแยกประเภทแบบ Binary Classification ค่า label จะต้องเป็น 0.0 (แทนป้ายชื่อลบ) หรือ 1.0 (แทนป้ายชื่อบวก){\*}- สำหรับการแยกประเภทแบบ Multi-class Classification ค่า label จะเก็บดัชนีคลาสตั้งแต่ศูนย์ขึ้นไปของการสังเกต (เช่น 0.0, 1.0, 2.0, ...)[\*]ในการสร้าง LabeledPoint ใน MLlib คุณต้องนำเข้าแพ็กเกจ `org.apache.spark.mllib.linalg.Vectors` และ `org.apache.spark.mllib.regression.LabeledPoint` ก่อน{\*}ต่อมาคุณสร้าง LabeledPoint ด้วยการใช้Method `LabeledPoint` โดยระบุค่า label และ features ของข้อมูล ซึ่ง features สามารถเป็นแบบ Dense หรือ Sparse ได้{\*}ตัวอย่างการสร้าง LabeledPoint:{\*}```scala{\*}import org.apache.spark.mllib.linalg.Vectors{\*}import org.apache.spark.mllib.regression.LabeledPoint{\*}val positive = LabeledPoint(1.0, Vectors.dense(10.0, 30.0, 20.0)){\*}val negative = LabeledPoint(0.0, Vectors.sparse(3, Array(0, 2), Array(200.0, 300.0))){\*}```{\*}แทนค่า label 1.0 และ 0.0 สำหรับแบบจำลองแบบ Binary Classification และใช้ `Vectors.dense` หรือ `Vectors.sparse` เพื่อสร้าง features ของข้อมูลที่ต้องการนำเข้า[\*]สำหรับ Python (PySpark) คุณสามารถสร้าง LabeledPoint ได้โดยใช้คลาส `LabeledPoint` จากแพ็กเกจ `pyspark.mllib.regression` และ `SparseVector` จากแพ็กเกจ `pyspark.mllib.linalg`{\*}ตัวอย่างการสร้าง LabeledPoint:{\*}```python{\*}from pyspark.mllib.linalg import SparseVector{\*}from pyspark.mllib.regression import LabeledPoint{\*}# สร้าง LabeledPoint ที่มี label เป็นบวก และ features เป็น dense vector{\*}pos = LabeledPoint(1.0, [1.0, 0.0, 3.0]){\*}# สร้าง LabeledPoint ที่มี label เป็นลบ และ features เป็น sparse vector{\*}neg = LabeledPoint(0.0, SparseVector(3, [0, 2], [1.0, 3.0])){\*}```{\*}ในการสร้าง sparse vector คุณต้องระบุขนาดของ vector พร้อมกับรายการของ indices และ values ที่ไม่ใช่ศูนย์ของ vector โดยใช้คลาส `SparseVector`[\*]MLlib ประเภทของ Rating:{\*}- ประเภท Rating ใช้กับอัลกอริทึมการแนะนำ{\*}- Rating ประกอบด้วยฟิลด์สามอย่าง:{\*}1. ฟิลด์แรกชื่อว่า user (ไอดีของผู้ใช้) ซึ่งมีประเภทเป็น Int{\*}2. ฟิลด์ที่สองชื่อว่า product (ไอเท็มไอดี) ซึ่งมีประเภทเป็น Int{\*}3. ฟิลด์ที่สามชื่อว่า rating ซึ่งมีประเภทเป็น Double[\*]สำหรับ MLlib ใน Scala คุณสามารถใช้คลาส `Rating` ในแพ็กเกจ `org.apache.spark.mllib.recommendation` เพื่อสร้าง Rating ที่ใช้กับอัลกอริทึมที่เกี่ยวข้องกับการแนะนำ{\*}ตัวอย่างการสร้าง Rating:{\*}```scala{\*}import org.apache.spark.mllib.recommendation.Rating{\*}// สร้าง Rating โดยกำหนด user id, item id, และ rating{\*}val rating = Rating(123, 456, 4.5){\*}```{\*}สำหรับ PySpark คุณสามารถใช้ Rating ได้โดยใช้คลาส `Rating` ในแพ็กเกจ `pyspark.mllib.recommendation`{\*}ตัวอย่างการสร้าง Rating ด้วย PySpark:{\*}```python{\*}from pyspark.mllib.recommendation import Rating{\*}# สร้าง Rating โดยกำหนด user id, item id, และ rating{\*}rating = Rating(123, 456, 4.5){\*}```{\*}คุณสามารถสร้าง Rating โดยระบุ user id, item id, และ rating ของผู้ใช้และรายการสินค้าที่ต้องการให้คะแนน`[\*]ใน Scala ของ MLlib คุณสามารถสร้าง Rating ด้วยการ import `Rating` จากแพ็กเกจ `org.apache.spark.mllib.recommendation` ดังนี้:{\*}```scala{\*}import org.apache.spark.mllib.recommendation.\_{\*}val rating = Rating(100, 10, 3.0){\*}```{\*}ใน PySpark คุณสามารถสร้าง Rating ด้วยการใช้คลาส `Rating` ในแพ็กเกจ `pyspark.mllib.recommendation` ดังนี้:{\*}```python{\*}from pyspark.mllib.recommendation import Rating{\*}rating = Rating(100, 10, 3.0){\*}```{\*}ในทั้งสองกรณีนี้คุณสร้าง Rating โดยระบุ user id เป็น 100, item id เป็น 10, และ rating เป็น 3.0 และค่าเหล่านี้ถูกใช้ในการสร้าง Rating object ใหม่`[\*]23--ใน MLlib มีอัลกอริทึม รูปภาพประกอบด้วย ข้อความ, ตัวอักษร, ภาพหน้าจอ, ไลน์

Description automatically generatedregression หลายตัวที่มีอยู่ในรูปแบบของคลาสและอ็อบเจ็กต์สิงเกิล (singleton objects) โดยเช่น{\*}1. \*\*Linear Regression with SGD\*\*{\*}- SGD แทน stochastic gradient descent (ใช้สำหรับการเรียนรู้ค่า coefficients ของสมการเชิงเส้นด้วยชุดข้อมูลการฝึกใหญ่){\*}- ใน Scala คุณสามารถใช้ SGD ใน LinearRegressionWithSGD object ดังนี้:{\*}```scala{\*}import org.apache.spark.mllib.regression.LinearRegressionWithSGD{\*}val model = LinearRegressionWithSGD.train(data, iterations, stepSize){\*}```{\*}- ใน PySpark คุณสามารถใช้ SGD ใน LinearRegressionWithSGD object ดังนี้:{\*}```python{\*}from pyspark.mllib.regression import LinearRegressionWithSGD{\*}model = LinearRegressionWithSGD.train(data, iterations, stepSize){\*}```{\*}- ที่นี่ `data` คือ RDD ของ LabeledPoint ที่ใช้ในการฝึกโมเดล, `iterations` คือจำนวนรอบการฝึก, และ `stepSize` คือขนาดของขั้นตอนการเรียนรู้{\*}- โมเดลที่ได้จะเป็น LinearRegressionModel ซึ่งสามารถใช้ในการทำนายผลลัพธ์ของข้อมูลใหม่ได้`[\*]26--Regression Algorithms{\*}- ใน linear regression, overfitting เกิดขึ้นเมื่อโมเดลมีความ "ซับซ้อนเกินไป"{\*}- สิ่งนี้มักเกิดขึ้นเมื่อมีจำนวนตัวแปรอิสระมากเกินไปเมื่อเปรียบเทียบกับจำนวนการสังเกต และ/หรือมี multi-collinearity ระหว่างตัวแปรเหล่านี้{\*}- โมเดลเช่นนี้จะไม่สามารถทำนายข้อมูลใหม่ได้ดี เนื่องจากจะมีประสิทธิภาพต่อข้อมูลการฝึก แต่จะมีประสิทธิภาพต่ำต่อข้อมูลทดสอบ[\*]27--2. การสอดแนมการเรียนรู้โดยใช้การจำกัด (regularization) เพื่อป้องกันการ overfitting โดยใช้เทคนิคเช่น Ridge Regression ซึ่งใช้ regularization แบบ L2 เพื่อป้องกัน overfitting โดยการเพิ่มผลรวมของพจน์กำลังสองของค่าพารามิเตอร์ในวัตถุประสงค์ของการปรับเพื่อลดความสัมพันธ์กับตัวแปรที่มีส่วนร่วมน้อยๆ ในโมเดลเพื่อให้ค่าพารามิเตอร์เหล่านั้นมีค่าเข้าใกล้ศูนย์มากขึ้น[\*]29--การใช้ Lasso with SGD นั้นใช้ Regularization แบบ L1 เพื่อป้องกันการ overfitting โดยการเพิ่มผลรวมของค่าสัมบูรณ์ของค่าพารามิเตอร์ไปยังวัตถุประสงค์ของการปรับและดังนั้นจึงทำให้บังคับให้บางส่วนของค่าพารามิเตอร์ที่มีส่วนร่วมน้อยๆ ในโมเดลเป็นค่าศูนย์ วัตถุประสงค์ของการใช้ L1 regularization นั้นเป็นไปตามสมการดังนี้:{\*}โดยที่ \(w\_i\) คือค่าพารามิเตอร์ของโมเดลที่ต้องปรับ, \(n\) คือรูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, ไลน์

Description automatically generatedจำนวนค่าพารามิเตอร์, และ \(\lambda\) คือพารามิเตอร์ regularization ที่กำหนดโดยผู้ใช้ ซึ่งมีค่ามากขึ้นเมื่อเราต้องการลดการ overfitting โดยที่ \(|w\_i|\) แทนค่าสัมบูรณ์ของ \(w\_i\). การใช้ L1 regularization ทำให้บางค่า \(w\_i\) ที่มีความสำคัญน้อยถูกตั้งค่าเป็น 0 โดยอัตโนมัติ ซึ่งช่วยลดจำนวนค่าพารามิเตอร์ที่ใช้จริงในโมเดลได้ในขณะที่ยังคงความแม่นยำของการทำนายไว้ใจได้.[\*]30--ElasticNet Regression ใช้ทั้ง L1 และ L2 regularization เพื่อป้องกันการ overfitting โดยวัตถุประสงค์ของการใช้ L1 และ L2 regularization นั้นเป็นไปตามสมการดังนี้:{\*}โดยที่ \(w\_i\) คือค่าพารามิเตอร์ของโมเดลที่ต้องปรับ, \(n\) คือจำนวนค่าพารามิเตอร์, และ \(\lambda\_1\) และ \(\lambda\_2\) คือพารามิเตอร์ regularization ที่กำหนดโดยผู้ใช้ ในกรณีนี้ \(\lambda\_1\) แทนค่าของพารามิเตอร์ regularization สำหรับ L1 regularization และ \(\lambda\_2\) แทนค่าของพารามิเตอร์ regularization สำหรับ L2 regularization โดยที่ \(|w\_i|\) แทนค่าสัมบูรณ์ของ \(w\_i\). การใช้ ElasticNet Regression ช่วยลดการ overfitting โดยลดจำนวนค่าพารามิเตอร์ที่มีผลน้อยต่อโมเดล พร้อมกับการลดค่าของพารามิเตอร์ที่มีผลน้อยจากทั้งสอง regularization terms ที่ใช้เข้าร่วมในการคำนวณผลรวมของค่าของวัตถุประสงค์ ซึ่งช่วยให้โมเดลมีความสามารถในการทำนายที่ดีกับข้อมูลที่ไม่เคยเห็นมาก่อนโดยไม่พบปัญหาการ overfitting ได้มากขึ้น{\*}โดยทั่วไปแล้วการใช้ ElasticNet Regression จะมีประสิทธิภาพดีในกรณีที่มีข้อมูลมีขนาดใหญ่และมีจำนวนค่าพารามิเตอร์มาก เนื่องจากมันช่วยลดการ overfitting ได้ดีและยังช่วยให้โมเดลมีความเสถียรมากขึ้นในบางกรณีที่ข้อมูลมีความเป็น sparse ด้วย แต่การใช้ ElasticNet Regression อาจจะมีความซับซ้อนมากขึ้นในการปรับค่าพารามิเตอร์เนื่องจากต้องกำหนดค่าของทั้งสองพารามิเตอร์ regularization ที่ใช้ร่วมกันในการคำนวณ และการเลือกค่าของพารามิเตอร์เหล่านี้อาจมีผลต่อประสิทธิภาพของโมเดลที่ได้ด้วย เพราะฉะนั้นการใช้ ElasticNet Regression ต้องคำนึงถึงการปรับค่าพารามิเตอร์ regularization อย่างรอบคอบเพื่อให้ได้ผลลัพธ์ที่ดีที่สุดจากโมเดลที่สร้างขึ้น[\*]31--Isotonic Regression เป็นอัลกอริทึมหนึ่งที่ใช้ในการทำ Regression และเป็นเครื่องมือที่มีประสิทธิภาพในกรณีที่ข้อมูลมีความสัมพันธ์ที่เป็นอะไรตรงๆ หรือเรียกอีกอย่างว่าเป็นความสัมพันธ์แบบไม่เส้นเรียกกันว่า Monotonic Regression หรือ Non-decreasing Regression{\*}ใน Isotonic Regression ความสัมพันธ์ระหว่างตัวแปรต้น (independent variable) และตัวแปรตาม (dependent variable) จะถูกปรับให้เป็นอะไรตรงตามแนวนอนโดยเฉพาะ หมายความว่าเมื่อตัวแปรต้นเพิ่มขึ้น ตัวแปรตามจะเพิ่มขึ้นเสมอ หรือไม่มีการลดลง เช่นเดียวกับเมื่อตัวแปรต้นลดลง ตัวแปรตามจะลดลงเสมอ โดยไม่มีการเกิดการเปลี่ยนแปลงที่รัดกุม (constraint) ในโมเดลทางสถิติ{\*}อัลกอริทึม Isotonic Regression มักถูกใช้ในงานที่เกี่ยวข้องกับลักษณะของเวลาหรือความสัมพันธ์ที่ต้องการเสถียรภาพตามเวลา หรือในกรณีที่ต้องการระบุความสัมพันธ์ที่ไม่แปรผันระหว่างตัวแปรต้นและตัวแปรตาม ตัวอย่างเช่นการวิเคราะห์ข้อมูลที่เกี่ยวกับราคาสินค้าตามเวลา โดยที่ต้องการหาแนวโน้มการเปลี่ยนแปลงของราคาสินค้าตามเวลา หรือรูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, จำนวน

Description automatically generatedการสำรวจความคิดเห็นของลูกค้าต่อสินค้าหรือบริการที่เปลี่ยนแปลงตามเวลา โดย Isotonic Regression จะช่วยในการระบุแนวโน้มของการเปลี่ยนแปลงได้อย่างตรงไปตรงมา โดยที่ไม่ต้องปรับค่าพารามิเตอร์เพิ่มเติมในโมเดล ซึ่งทำให้เป็นวิธีที่สะดวกและมีประสิทธิภาพในการวิเคราะห์ข้อมูลที่มีลักษณะเช่นนี้[\*]32--ในการวิเคราะห์แบบแยกประเภท (Classification) และการวิเคราะห์แบบเชิงตัวเลข (Regression) นอกจากจะมีวิธีการแบบแบ่งตัดอย่างง่ายแบบหนึ่งเช่นการใช้เส้นตรงหรือโค้งโพลินอมิแอลแบบเดียวกัน เรายังสามารถใช้อัลกอริทึมอื่นๆ เพื่อทำนายผลลัพธ์ได้อีกหลายวิธี ดังนี้{\*}1. \*\*Decision Tree (ต้นไม้ตัดสินใจ)\*\*: ต้นไม้ตัดสินใจเป็นอัลกอริทึมที่ใช้แบ่งชุดข้อมูลออกเป็นชุดย่อยๆ โดยเริ่มต้นที่โหนดหลัก (root node) และแยกแยะข้อมูลโดยใช้เงื่อนไขเชิงคุณลักษณะที่ต่างกัน โดยวัดความไม่แน่นอน (impurity) ของข้อมูลในแต่ละโหนด หนึ่งในวิธีการวัดความไม่แน่นอนที่ใช้บ่อยคือค่าความแปรปรวน (variance) ของข้อมูล การตัดสินใจในแต่ละโหนดจะทำให้เกิดสัญญาณที่สอดคล้องกับข้อมูลมากที่สุดที่เป็นไปได้{\*}2. \*\*Gradient Boosted Trees (ต้นไม้ที่ถูกเพิ่มพลังด้วยการสร้างโมเดลใหม่)\*\*: Gradient Boosted Trees เป็นวิธีการที่ใช้กระบวนการการเรียนรู้เชิงตัวแบบ Ensemble เพื่อสร้างต้นไม้ตัดสินใจหลายต้นขึ้นมา โดยการเรียนรู้จากข้อผิดพลาดของโมเดลก่อนหน้าและพยายามแก้ไขในการสร้างโมเดลใหม่ เรียกว่ากระบวนการ Boosting ซึ่งทำให้โมเดลมีความแม่นยำมากขึ้น{\*}3. \*\*Random Forest (ป่าไม้สุ่ม)\*\*: Random Forest เป็นอัลกอริทึมอีนเซมเบิลที่ใช้แบ่งชุดข้อมูลออกเป็นชุดย่อยๆ โดยการสร้างหลายโมเดลต้นไม้ตัดสินใจแยกย่อย และจากนั้นรวมผลลัพธ์ที่ได้จากแต่ละโมเดล การนำเสนอผลลัพธ์ที่ถูกทำซ้ำของโมเดลต้นไม้ช่วยลดความเสี่ยงที่เกิดจากการเรียนรู้แบบเจาะจงของแต่ละโมเดล นั่นทำให้ Random Forest เป็นอัลกอริทึมที่มีประสิทธิภ{\*}าพและมั่นคงกับข้อมูลใหญ่ที่มีความซับซ้อนได้ดี[\*]33--สำหรับอัลกอริทึมที่เกี่ยวกับการทำนายเชิงเส้นเช่น Linear Regression with SGD หรือ Ridge Regression ใน MLlib คุณสามารถใช้วิธี train เพื่อสร้างโมเดลได้ ตัวอย่างเช่นด้านล่างนี้แสดงการใช้งานของ Linear Regression with SGD:{\*}```scala{\*}import org.apache.spark.mllib.linalg.Vectors{\*}import org.apache.spark.mllib.regression.LabeledPoint{\*}import รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, จำนวน

Description automatically generatedorg.apache.spark.mllib.regression.LinearRegressionWithSGD{\*}// สร้างข้อมูลตัวอย่าง{\*}val data = sc.textFile("data/mllib/ridge-data/lpsa.data"){\*}val parsedData = data.map { line =>{\*}val parts = line.split(','){\*}LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(\_.toDouble))){\*}}{\*}// กำหนดพารามิเตอร์สำหรับโมเดล{\*}val numIterations = 100{\*}val model = LinearRegressionWithSGD.train(parsedData, numIterations){\*}```{\*}ในกรณีที่ต้องการใช้อัลกอริทึมอื่น ๆ เช่น Ridge Regression หรืออัลกอริทึมอื่น ๆ คุณสามารถเปลี่ยน `LinearRegressionWithSGD` เป็นอัลกอริทึมที่คุณต้องการใช้แทน ตามตัวอย่างด้านล่าง:{\*}```scala{\*}import org.apache.spark.mllib.regression.RidgeRegressionWithSGD{\*}val model = RidgeRegressionWithSGD.train(parsedData, numIterations){\*}```{\*}คุณสามารถปรับตัวแปร `numIterations` เพื่อปรับจำนวนรอบการฝึกสอนตามที่ต้องการได้ตามความเหมาะสมของข้อมูลของคุณ[\*]34--สำหรับโค้ดด้านล่างนี้เป็นตัวอย่างของการใช้งาน Linear Regression with SGD ใน Spark MLlib:{\*}```scala{\*}import org.apache.spark.mllib.regression.LabeledPoint{\*}import org.apache.spark.mllib.linalg.Vectors{\*}import org.apache.spark.mllib.regression.LinearRegressionWithSGD{\*}// อ่านข้อมูลจากไฟล์{\*}val lines = sc.textFile("/user/cloudera/lpsa.data"){\*}// แปลงข้อมูลเป็น LabeledPoint{\*}val labeledPoints = lines.map { line =>{\*}val Array(rawLabel, rawFeatures) = line.split(','){\*}val features = rawFeatures.split(' ').map(\_.toDouble){\*}LabeledPoint(rawLabel.toDouble, Vectors.dense(features)){\*}}{\*}// ทำการแคชข้อมูล{\*}labeledPoints.cache(){\*}// กำหนดจำนวนรอบการฝึกสอน{\*}val numIterations = 100{\*}// สร้างและฝึกโมเดล Linear Regression with SGD{\*}val lrModel = LinearRegressionWithSGD.train(labeledPoints, numIterations){\*}// ดึงค่า intercept และ weights จากโมเดล{\*}val intercept = lrModel.intercept{\*}val weights = lrModel.weights{\*}```{\*}โค้ดด้านบนทำการอ่านข้อมูลจากไฟล์และแปลงข้อมูลให้อยู่ในรูปแบบของ LabeledPoint หลังจากนั้นทำการแคชข้อมูลเพื่อประสิทธิภาพในการรูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, ไลน์

Description automatically generatedทำงาน จากนั้นกำหนดจำนวนรอบการฝึกสอนและสร้างและฝึกโมเดล Linear Regression with SGD ก่อนที่จะดึงค่า intercept และ weights จากโมเดลที่ได้กลับมาใช้ในการวิเคราะห์ต่อไป โค้ดดังกล่าวเป็นตัวอย่างเพียงอย่างเดียว การปรับแต่งและประยุกต์ใช้กับข้อมูลของคุณอาจต้องการการปรับแต่งเพิ่มเติมและการทดลองกับพารามิเตอร์ต่าง ๆ ของอัลกอริทึม เพื่อให้ได้ผลลัพธ์ที่ดีที่สุดสำหรับงานที่คุณต้องการทำ[\*]35--โค้ดต่อไปนี้เป็นตัวอย่างของวิธี TrainRegressor ใน Spark MLlib ที่ใช้สร้างและฝึกโมเดล Regression ที่ไม่เชิงเส้น เช่น DecisionTree และ RandomForest:{\*}```scala{\*}import org.apache.spark.mllib.tree.DecisionTree{\*}import org.apache.spark.mllib.tree.RandomForest{\*}import org.apache.spark.mllib.tree.model.DecisionTreeModel{\*}import org.apache.spark.mllib.tree.model.RandomForestModel{\*}import org.apache.spark.mllib.util.MLUtils{\*}// โหลดและแปลงข้อมูลเป็น LabeledPoint{\*}val data = MLUtils.loadLibSVMFile(sc, "data/mllib/sample\_libsvm\_data.txt"){\*}val splits = data.randomSplit(Array(0.7, 0.3)){\*}val (trainingData, testData) = (splits(0), splits(1)){\*}// กำหนดพารามิเตอร์สำหรับการฝึกสอน{\*}val categoricalFeaturesInfo = Map[Int, Int]() // รายการคู่ค่า key-value, โดยที่ key เป็น id ของตัวแปรแบบหมวดหมู่ และ value เป็นจำนวนของค่าที่เป็นแบบหมวดหมู่สำหรับตัวแปรนั้นๆ ในที่นี้เป็นรายการว่างหมายถึงไม่มีตัวแปรแบบหมวดหมู่{\*}val numTrees = 3 // จำนวนต้นไม้{\*}val featureSubsetStrategy = "auto" // จำนวนคุณลักษณะที่จะพิจารณาสำหรับการแบ่งข้อมูลที่แต่ละโหนด{\*}val impurity = "variance" // ค่าของค่าไม่สำคัญสำหรับการแบ่งข้อมูลในต้นไม้{\*}val maxDepth = 4 // ความลึกสูงสุดของต้นไม้{\*}val maxBins = 32 // จำนวนสีสูงสุดสำหรับการหยิบสีของตัวแปรตัวเลข{\*}// Train a DecisionTree model{\*}val dtModel = DecisionTree.trainRegressor(trainingData, categoricalFeaturesInfo, impurity,{\*}maxDepth, maxBins){\*}// Train a RandomForest model{\*}val rfModel = RandomForest.trainRegressor(trainingData, categoricalFeaturesInfo, numTrees,{\*}featureSubsetStrategy, impurity, maxDepth, maxBins){\*}// Evaluate model on test instances and compute รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, แผนภาพ

Description automatically generatedtest error{\*}val dtPredictions = dtModel.predict(testData.map(\_.features)){\*}val rfPredictions = rfModel.predict(testData.map(\_.features)){\*}val dtMSE = dtPredictions.zip(testData.map(\_.label)).map{ case (prediction, label) => math.pow(prediction - label, 2)}.mean(){\*}val rfMSE = rfPredictions.zip(testData.map(\_.label)).map{ case (prediction, label) => math.pow(prediction - label, 2)}.mean(){\*}println(s"DecisionTree Mean Squared Error = $dtMSE"){\*}println(s"RandomForest Mean Squared Error = $rfMSE"){\*}```{\*}ในโค้ดด้านบน เรากำหนดพารามิเตอร์สำหรับการฝึกสอนและสร้างโมเดล DecisionTree และ RandomForest โดยใช้ข้อมูลการฝึกสอน (training data) จากการแบ่งชุดข้อมูลออกเป็นชุดฝึกสอนและชุดทดสอบ (test data) จากนั้นทำการประเมินประสิทธิภาพของโมเดลดังกล่าวบนชุดทดสอบและคำนวณค่า Mean Squared Error ของทั้ง DecisionTree และ RandomForest สุดท้ายทำการแสดงผลลัพธ์ที่ได้ด้วยคำสั่ง println()[\*]ในโค้ดด้านบนเราใช้วิธี `trainRegressor` เพื่อฝึกโมเดล RandomForest ใน MLlib โดยให้ข้อมูลของเราเป็น `labeledPoints` ที่เป็นชุดข้อมูลที่มีป้ายชื่อและคุณลักษณะ (labelled points) และกำหนดพารามิเตอร์ต่างๆดังนี้:{\*}- `labeledPoints`: ชุดข้อมูลที่มีป้ายชื่อและคุณลักษณะ{\*}- `categoricalFeaturesInfo`: รายการคู่ค่า key-value ที่ระบุข้อมูลเกี่ยวกับตัวแปรแบบหมวดหมู่ ซึ่งในที่นี้เราใช้ค่าว่างแทน{\*}- `NumTrees`: จำนวนของต้นไม้ใน RandomForest{\*}- `FeatureSubsetStrategy`: กลยุทธ์ที่ใช้ในการเลือก subset ของ features ที่จะพิจารณาเพื่อใช้ในการสร้างต้นไม้{\*}- `impurity`: ค่าที่ใช้ในการคำนวณ impurity ของ node ในต้นไม้{\*}- `MaxDepth`: ความลึกสูงสุดของต้นไม้{\*}- `maxBins`: จำนวนสีสูงสุดที่ใช้ในการหยิบแบ่งข้อมูลสำหรับตัวแปรตัวเลข{\*}โค้ดดังกล่าวจะสร้างและฝึกโมเดล RandomForest จากชุดข้อมูลที่กำหนด และเก็บโมเดลที่สร้างไว้ในตัวแปร `rfModel` โดยโมเดลนี้จะถูกใช้ในการทำนายค่าของตัวแปรตามในแหล่งข้อมูลใหม่ที่ไม่รู้จักล่วงหน้า หลังจากฝึกโมเดลเสร็จสิ้นแล้ว โมเดลสามารถใช้ในการทำนายค่าของตัวแปรตามโดยใช้ข้อมูลใหม่ที่ไม่เคยเห็นมาก่อนโดยใช้Method `predict` ที่มีอยู่ใน `RandomForestModel` ที่สร้างขึ้นในระหว่างการฝึกสอนโมเดล โดยในการทำนายเราจะใช้ข้อมูลทดสอบที่เราไม่เคยใช้ในการฝึกสอนโมเดล และคำนวณค่า MSE เพื่อประเมินประสิทธิภาพของโมเดล RandomForest ที่เราได้ฝึกสอนไว้ โดยคำนวณผลต่างระหว่างค่าที่ทำนายได้และค่าจริงของแต่ละข้อมูลในชุดข้อมูลทดสอบและคำนวณค่าเฉลี่ยของค่า MSE ทั้งหมดที่ได้จากชุดข้อมูลทดสอบทั้งหมด[\*]36--การสร้างโมเดล Regression ด้วย RandomForest ใน Spark MLlib มีรูปแบบดังนี้:{\*}```scala{\*}val rfModel = RandomForest.trainRegressor({\*}labeledPoints// ชุดข้อมูล LabeledPoint{\*}categoricalFeaturesInfo// ข้อมูลทางสถิติเกี่ยวกับคุณลักษณะทางสถิติแบบอะไรก็ได้ที่มีคุณลักษณะแบบจำกัด{\*}numTrees,// จำนวนของต้นไม้ในโมเดล{\*}featureSubsetStrategy // กลยุทธ์ในการเลือกคุณลักษณะในการสร้างต้นไม้แต่ละต้น{\*}impurity,// วิธีการคำนวณความบริสุทธิ์ในการแบ่งสาขาของต้นไม้{\*}maxDepth,// ความลึกสูงสุดของต้นไม้{\*}maxBins // จำนวนสีสูงสุดที่ใช้ในการแบ่งตัดข้อมูล{\*}){\*}```{\*}โดยที่:{\*}- `labeledPoints`: ชุดข้อมูล LabeledPoint ซึ่งประกอบด้วยค่า label และ features{\*}- `categoricalFeaturesInfo`: ข้อมูลทางสถิติเกี่ยวกับคุณลักษณะทางสถิติแบบอะไรก็ได้ที่มีคุณลักษณะแบบจำกัด (อาจจะใช้ในการแยกประเภท){\*}- `numTrees`: จำนวนของต้นไม้ในโมเดล Random Forest{\*}- `featureSubsetStrategy`: กลยุทธ์ในการเลือกคุณลักษณะในการสร้างต้นไม้แต่ละต้น (เช่น "auto", "sqrt", "log2", "onethird"){\*}- `impurity`: วิธีการคำนวณความบริสุทธิ์ในการแบ่งสาขาของต้นไม้ (เช่น "variance" สำหรับ Regression){\*}- `maxDepth`: ความลึกสูงสุดของต้นไม้{\*}- `maxBins`: จำนวนสีสูงสุดที่ใช้ในการแบ่งตัดข้อมูล[\*]37--อัลกอริทึมการถดถอย (Regression Algorithms){\*}• predict – เป็นเมทอดที่ใช้ทำนายตัวเลขบนป้ายชื่อ (label) สำหรับชุดคุณลักษณะที่กำหนด{\*}• มันจะรับชุดคุณลักษณะเข้ามาในรูปแบบของเวกเตอร์และคืนค่าป้ายชื่อตัวเลขของชนิด Double{\*}```scala{\*}val observedAndPredictedLabels ={\*}labeledPoints.map { observation =>{\*}val predictedLabel = lrModel.predict(observation.features){\*}(observation.label, predictedLabel){\*}}{\*}```{\*}โค้ดนี้มีการใช้งานตัวแปร `labeledPoints` ซึ่งรูปภาพประกอบด้วย ข้อความ, แผนภาพ, ไลน์, พล็อต

Description automatically generatedอาจเป็นชุดข้อมูลที่มีป้ายชื่อและคุณลักษณะที่เกี่ยวข้อง โดยการใช้เมทอด `map` เพื่อผ่านทุก ๆ ข้อมูลใน `labeledPoints` และทำการคำนวณค่าที่ทำนายได้ (`predictedLabel`) โดยใช้โมเดลการถดถอย (`lrModel`) กับคุณลักษณะ (`observation.features`) ของแต่ละข้อมูล และคืนค่าเป็นคู่ป้ายชื่อที่สังเกตและค่าที่ทำนายได้ในรูปแบบที่เป็น Tuple ใน `observedAndPredictedLabels` ซึ่งประกอบด้วยค่าป้ายชื่อจริง (observed label) และค่าที่ทำนายได้ (predicted label) สำหรับแต่ละข้อมูลในชุดข้อมูล `labeledPoints` โดยอ่านเป็นภาษาสคาล่าได้ว่า "ทำการ map ทุก observation ใน labeledPoints แล้วทำการทำนายค่าของ lrModel ด้วย features ของ observation นั้น ๆ และคืนค่าเป็นคู่ของ label ที่สังเกตและ label ที่ทำนายได้"[\*]38--อัลกอริทึมการจำแนกประเภท (Classification Algorithms){\*}• MLlib มีการให้บริการอัลกอริทึมการจำแนกประเภทหลายตัวเช่น - โมเดลเชิงเส้น เช่น LogisticRegressionWithSGD และ LogisticRegressionWithLBFGS (Broyden–Fletcher Goldfarb–Shanno (BFGS) algorithm, เป็นวิธีการทำซ้ำสำหรับการแก้ปัญหาการปรับเปลี่ยนอ้อมแบบไม่มีข้อจำกัดที่ไม่เป็นเชิงเส้น){\*}SVMwithSGD - NaiveBayes - โมเดลเชิงต้นไม้ (ที่ไม่เชิงเส้น) เช่น DecisionTree GradientBoostedTrees RandomForest[\*]39--การสร้างโมเดลการจำแนกประเภท (Building Classification Model){\*}• เมทอด Train ของอ็อบเจกต์การจำแนกประเภทสร้างโมเดลการจำแนกประเภทขึ้นอย่างขึ้นอยู่กับชุดข้อมูลการฝึกที่กำหนด มันรับ RDD ของ LabeledPoint เป็นอาร์กิวเมนต์และคืนโมเดลการจำแนกประเภทที่เฉพาะเจาะจงตามอัลกอริทึม{\*}```scala{\*}val Array(trainingData, validationData, testData) = labeledPoints.randomSplit(Array(0.6, 0.2, 0.2)){\*}trainingData.cache(){\*}val numIterations = 100{\*}val svmModel = SVMwithSGD.train(trainingData, numIterations){\*}```{\*}โค้ดนี้ทำการแบ่งชุดข้อมูล `labeledPoints` ออกเป็นชุดข้อมูลการฝึก (trainingData), ชุดข้อมูลที่ใช้ตรวจสอบ (validationData), และชุดข้อมูลทดสอบ (testData) โดยใช้เมทอด `randomSplit` และกำหนดอัตราส่วนของแต่ละชุดข้อมูลในอาร์เรย์ `[0.6, 0.2, 0.2]` ซึ่งแต่ละอาร์เรย์แทนสัดส่วนของการแบ่งชุดข้อมูลแต่ละชุด หลังจากนั้นจะทำการ cache ชุดข้อมูลการฝึกด้วยเมทอด `cache()` เพื่อเพิ่มประสิทธิภาพในการเข้าถึงข้อมูล สุดท้ายจะทำการฝึกโมเดล SVM โดยใช้ข้อมูลการฝึกและจำนวนการทำซ้ำเท่ากับ 100 ด้วยเมทอด `SVMwithSGD.train()` และเก็บโมเดลที่ได้ในตัวแปร `svmModel`[\*]40--การสร้างโมเดลการจำแนกประเภท (Building Classification Model){\*}• เมทอด trainClassifier สร้างโมเดลการจำแนกประเภทขึ้นตามอัลกอริทึมที่ใช้เช่น DecisionTree และ RandomForest{\*}• มันรับ RDD ของ LabelPoints และพารามิเตอร์พิเศษของอัลกอริทึมเป็นอาร์กิวเมนต์{\*}• มันคืนโมเดลที่เฉพาะเจาะจงตามอัลกอริทึม[\*]41--การสร้างโมเดลการจำแนกประเภท (Building Classification Model){\*}• เมทอด predict คืนค่าคลาสหรือป้ายชื่อหมวดหมู่ของชนิด Double สำหรับชุดคุณลักษณะที่กำหนดในรูปแบบเวกเตอร์{\*}```scala{\*}val predictedLabels = testData.map { observation =>{\*}val predictedLabel = svmModel.predict(observation.features){\*}(predictedLabel, observation.label){\*}}{\*}```{\*}โค้ดนี้ทำการทำนายค่าของชุดทดสอบ `testData` โดยใช้โมเดล SVM (`svmModel`) และเมทอด `predict` โดยผ่านคุณลักษณะของแต่ละ observation ในรูปแบบของเวกเตอร์ และคืนค่าเป็นคู่ของค่าที่ทำนายได้ (`predictedLabel`) และป้ายชื่อจริง (`observation.label`) สำหรับแต่ละ observation ในชุดข้อมูลทดสอบ `testData`[\*]42--การสร้างโมเดลการจำแนกประเภท (Building Classification Model){\*}• เมทอด save บันทึกโมเดลที่ฝึกไว้ลงในดิสก์ มันรับ SparkContext และเส้นทาง (path) เป็นอาร์กิวเมนต์และบันทึกโมเดลต้นฉบับไว้ในเส้นทางที่กำหนด โมเดลที่บันทึกไว้สามารถอ่านเข้ามาในภายหลังได้ด้วยเมทอด load{\*}```scala{\*}svmModel.save(sc, "models/svm-model"){\*}```{\*}• เมทอด load ถูกกำหนดไว้ในอ็อบเจกต์โมเดลที่สามารถนำเข้าได้{\*}```scala{\*}import org.apache.spark.mllib.classification.SVMModel{\*}val savedSVMModel = SVMModel.load(sc, "models/svm-model"){\*}```{\*}โค้ดนี้แสดงการบันทึกโมเดล SVM ที่ฝึกไว้ลงในดิสก์ในเส้นทาง "models/svm-model" โดยใช้เมทอด `save` และสามารถโหลดโมเดลที่บันทึกไว้นี้เข้ามาในภายหลังได้โดยใช้เมทอด `load` โดยนำเข้า `org.apache.spark.mllib.classification.SVMModel` และใช้ `SVMModel.load()` ด้วย SparkContext (`sc`) และเส้นทางของโมเดลที่บันทึกไว้[\*]43--การจัดกลุ่ม (Clustering){\*}• MLlib มีการให้บริการอัลกอริทึมการจัดกลุ่มหลายอย่าง เช่น - Kmeans (การทำขนาดพร้อมกันของวิธี k means++) - StreamingKmeans (ใช้สำหรับการจัดกลุ่มข้อมูลที่มีการเข้ามาเป็นสตรีมในแบบแช่ง) - GuassianMixture (สามารถจัดกลุ่มข้อมูลที่มีรูปแบบและขนาดที่เป็นรูปวงรีโลกได้โดยใช้โมเดลความน่าจะเป็น) - LDA (การจัดกลุ่มแบบหัวข้อที่ใช้สำหรับการจัดกลุ่มข้อมูลเอกสาร) - Power Iteration Clustering (ใช้สำหรับการจัดกลุ่มข้อมูลกราฟ)[\*]44--การจัดกลุ่ม (Clustering){\*}• เมทอด train ถูกโดยอ็อบเจกต์ที่เกี่ยวข้องกับการจัดกลุ่มให้ความสำคัญ มันรับ RDD ของ Vector และพารามิเตอร์ที่เฉพาะเจาะจงตามอัลกอริทึมการจัดกลุ่มเป็นอาร์กิวเมนต์ และคืนโมเดลที่เฉพาะเจาะจงตามอัลกอริทึม{\*}• พารามิเตอร์และประเภทของโมเดลที่คืนขึ้นอยู่กับอัลกอริทึมการจัดกลุ่ม{\*}• ตัวอย่าง - พารามิเตอร์ที่รับได้โดยเมทอด train สำหรับ Kmeans รวมถึงจำนวนกลุ่ม จำนวนการทำซ้ำสูงสุดในแต่ละรอบ จำนวนการรันขนาดพร้อมกัน โหมดการเริ่มต้น และค่าเริ่มต้นของตัวเลขสุ่มสำหรับเริ่มต้น[\*]45--การจัดกลุ่ม (Clustering){\*}```scala{\*}import org.apache.spark.mllib.clustering.KMeans{\*}import org.apache.spark.mllib.linalg.Vectors{\*}```{\*}ในบรรทัดแรกนี้ เราทำการ import คลาส KMeans และ Vectors จากไลบรารี org.apache.spark.mllib.clustering และ org.apache.spark.mllib.linalg ตามลำดับ{\*}```scala{\*}val lines = sc.textFile("data/mllib/Kmeans\_data.txt"){\*}```{\*}บรรทัดที่สองนี้เป็นการโหลดข้อมูลจากไฟล์ "data/mllib/Kmeans\_data.txt" โดยใช้ SparkContext (sc) และเก็บข้อมูลในตัวแปร lines{\*}```scala{\*}val arraysofDoubles = lines.map { line => line.split(' ').map(\_.toDouble)}{\*}```{\*}บรรทัดที่สามนี้ทำการแยกข้อมูลในแต่ละบรรทัดเป็นค่าที่แยกกันด้วยช่องว่าง แล้วแปลงเป็น Double โดยใช้เมทอด map และ split{\*}```scala{\*}val vectors = arraysofDoubles.map { a => Vectors.dense(a) }.cache(){\*}```{\*}บรรทัดที่สี่นี้ทำการแปลงข้อมูลที่แยกไว้ในรูปของอาร์เรย์ของ Double เป็น Vector โดยใช้ Vectors.dense และเก็บข้อมูลในแคชเพื่อเพิ่มประสิทธิภาพในการเข้าถึงข้อมูล{\*}```scala{\*}val numClusters = 2{\*}val numIterations = 20{\*}```{\*}ในบรรทัดที่ห้าและหก เรากำหนดจำนวน cluster และจำนวนรอบการทำซ้ำสูงสุดในแต่ละ cluster{\*}```scala{\*}val KMeansModel = KMeans.train(vectors, numClusters, numIterations){\*}```{\*}สุดท้าย เราใช้เมทอด train ของคลาส KMeans เพื่อฝึกโมเดล KMeans โดยใช้ vectors, numClusters และ numIterations เป็นพารามิเตอร์ และเก็บโมเดลที่ได้ในตัวแปร KMeansModel[\*]46--การจัดกลุ่ม (Clustering){\*}```scala{\*}val numClusters = 2{\*}val numIterations = 20{\*}```{\*}ในบรรทัดแรกถึงสอง เรากำหนดจำนวนของ cluster และจำนวนรอบการทำซ้ำสูงสุดในแต่ละ cluster{\*}```scala{\*}val kmeans = new KMeans().setMaxIterations(numIterations).setK(numClusters){\*}```{\*}บรรทัดที่สามนี้สร้างอ็อบเจกต์ของคลาส KMeans และกำหนดจำนวนรอบการทำซ้ำและจำนวนของ cluster ด้วยเมทอด setMaxIterations และ setK ตามลำดับ{\*}```scala{\*}val KMeansModel = kmeans.run(vectors){\*}```{\*}ในบรรทัดสี่ เราใช้เมทอด run ของอ็อบเจกต์ KMeans เพื่อฝึกโมเดล KMeans โดยใช้ข้อมูลเข้า (vectors) เราสามารถเก็บโมเดลที่ได้ในตัวแปร KMeansModel ได้[\*]47--ฟังก์ชัน Predict ในการจัดกลุ่ม (Clustering) มีหน้าที่คืนค่าดัชนีของกลุ่ม (cluster index) สำหรับตัวอย่างข้อมูลที่กำหนด เมื่อรับพารามิเตอร์เป็นเวกเตอร์ (Vector) และคืนค่าเป็นชนิดข้อมูล Int (จำนวนเต็ม){\*}หลักการทำงานของฟังก์ชัน Predict นั้นมักใช้ร่วมกับวิธีการจัดกลุ่ม (clustering algorithm) เช่น K-means, Hierarchical clustering, หรือ DBSCAN เพื่อให้แยกข้อมูลออกเป็นกลุ่มตามลักษณะหรือลักษณะของข้อมูลที่มีอยู่ แต่ละกลุ่มจะถูกกำหนดด้วยดัชนีหรือตัวเลขที่แทนความสัมพันธ์หรือความคล้ายคลึงกันของข้อมูลภายในกลุ่มนั้นๆ{\*}ในกรณีที่ใช้งานฟังก์ชัน Predict นี้ คุณจะส่งเวกเตอร์ของข้อมูลที่ต้องการจัดกลุ่มเข้าไป และฟังก์ชันจะคืนค่าดัชนีของกลุ่มที่ข้อมูลนั้นถูกจัดไว้ เพื่อใช้งานต่อในการวิเคราะห์หรือการประมวลผลข้อมูลต่อไป การคืนค่าดัชนีของกลุ่มจะช่วยให้สามารถระบุว่าข้อมูลนั้นอยู่ในกลุ่มใดในขั้นตอนถัดไปได้และนำไปใช้ประโยชน์ในการตัดสินใจหรือการวิเคราะห์ข้อมูลต่อไปได้ให้มีประสิทธิภาพมากยิ่งขึ้น{\*}การจัดกลุ่ม (Clustering){\*}```scala{\*}val obs = Vectors.dense(0.0, 0.0, 0.0){\*}val clusterIndex = KMeansModel.predict(obs){\*}```{\*}บรรทัดแรกนี้ใช้เมทอด predict เพื่อทำนายหมวดหมู่ (cluster) สำหรับตัวอย่างที่กำหนด โดยให้ obs เป็น Vector และเก็บค่าหมายเลขของ cluster ที่ได้ในตัวแปร clusterIndex{\*}```scala{\*}val WSSSE = KMeansModel.computeCost(vectors){\*}```{\*}บรรทัดที่สองใช้เมทอด computeCost เพื่อคำนวณค่า WSSSE (Within Set Sum of Squared Errors) ซึ่งเป็นผลรวมของระยะห่างที่ถูกยกกำลังสองของตัวอย่างจากจุดศูนย์กลุ่มที่ใกล้ที่สุดของพวกเขา ซึ่งเป็นวิธีในการประเมินโมเดล KMeans ได้ลองใช้งานโดยให้ vectors เป็นอาร์กิวเมนต์ และเก็บผลลัพธ์ในตัวแปร WSSSE[\*]48--เมธอด Save ใช้ในการบันทึก (persist) โมเดลที่ฝึก (trained model) จากการจัดกลุ่มลงบนดิสก์ โดยมีการรับ SparkContext และเส้นทาง (path) เป็นพารามิเตอร์ โมเดลต้นฉบับจะถูกบันทึกไว้ในเส้นทางที่ระบุ โมเดลที่ถูกบันทึกนี้สามารถอ่านกลับมาใช้งานในภายหลังด้วยเมธอด load โดยเมธอดนี้ช่วยให้สามารถบันทึกและนำโมเดลไปใช้งานในระยะยาวได้โดยมีความสะดวกและเป็นเอกสิทธิ์{\*}การจัดกลุ่ม (Clustering){\*}```scala{\*}KMeansModel.save(sc, "models/kmeans"){\*}```{\*}บรรทัดแรกนี้ใช้เมทอด save เพื่อบันทึกโมเดลการจัดกลุ่มที่ฝึกไว้ลงในดิสก์ โดยรับ SparkContext (sc) และเส้นทาง (path) เป็นอาร์กิวเมนต์ และบันทึกโมเดลต้นฉบับไว้ในเส้นทางที่กำหนด{\*}```scala{\*}import org.apache.spark.mllib.clustering.KMeansModel{\*}val savedKMeansModel = KMeansModel.load(sc, "models/kmeans"){\*}```{\*}บรรทัดที่สองนี้ใช้เมทอด load เพื่อโหลดโมเดลที่บันทึกไว้จากเส้นทาง "models/kmeans" โดยรับ SparkContext (sc) และเส้นทางของโมเดลที่บันทึกไว้เป็นอาร์กิวเมนต์ และเก็บโมเดลที่โหลดไว้ในตัวแปร savedKMeansModel[\*]50--การกรองร่วม (Collaborative Filtering) เป็นเทคนิคที่ใช้ในการสร้างข้อเสนอ (recommendations) โดยอิงจากพฤติกรรมในอดีต ความชอบ หรือความคล้ายคลึงกับสิ่งที่รู้จัก เช่น ผู้ใช้หรือสิ่งของต่างๆ สร้างข้อเสนอโดยการพิจารณาคะแนนความชอบ (preference score) หรือการจัดอันดับของผู้ใช้ต่อสิ่งของต่างๆ ในรูปแบบของเมทริกซ์ผู้ใช้-สิ่งของ (User-Item Matrix) โดยมีรายละเอียดดังนี้:{\*}- item1, item2, item3: คือสิ่งของที่ต้องการทำนายหรือทำข้อเสนอเกี่ยวกับ{\*}- user1, user2, ...: คือผู้ใช้หรือส่วนที่ต้องการทำนายสิ่งของให้กับ{\*}- ri,j: คือคะแนนความชอบหรือการจัดอันดับของผู้ใช้ i ต่อสิ่งของ j ซึ่งอาจมีค่าอยู่ในช่วง 1 ถึง 5 เช่น{\*}- 1: ไม่ชอบ{\*}- 2: นิดหน่อย{\*}- 3: ปานกลาง{\*}- 4: ชอบ{\*}- 5: ชอบมาก{\*}User-Item Matrix เป็นโครงสร้างข้อมูลที่ใช้เก็บคะแนนหรือความชอบของผู้ใช้ต่อสิ่งของที่ต้องการทำนาย โดยแนวคิดหลักคือการจัดเก็บข้อมูลในรูปแบบตารางที่แถวแทนผู้ใช้และคอลัมน์แทนสิ่งของ ดังนั้นคุณสามารถใช้ข้อมูลใน User-Item Matrix เพื่อสร้างโมเดลและทำนายความชอบหรือข้อเสนอสำหรับผู้ใช้ในอนาคตโดยอิงจากข้อมูลของผู้ใช้ก่อนหน้านี้และความคล้ายคลึงกับผู้ใช้อื่นๆ ที่มีพฤติกรรมที่คล้ายกัน[\*]51--การกรองร่วม (Collaborative Filtering) มักพบว่าส่วนใหญ่ของข้อมูลในเมทริกซ์ผู้ใช้-สิ่งของ (User-Item Matrix) นั้นเป็นค่าที่ไม่ทราบ (unknown) ซึ่งอาจเป็นเพราะผู้ใช้ยังไม่ได้ให้คะแนนหรือประเมินสิ่งของเหล่านั้น ดังนั้นการกรองร่วมจะพยายามทำนายค่า ri,j ที่ไม่ทราบสำหรับผู้ใช้ i ต่อสิ่งของ j โดยใช้ข้อมูลที่มีอยู่ในเมทริกซ์{\*}เมื่อค่าที่ทำนายได้ถูกคำนวณแล้ว สิ่งของ j จะถูกแนะนำให้กับผู้ใช้ i โดยขึ้นอยู่กับค่าที่ทำนายได้ (เช่น ถ้าคะแนนที่ทำนายได้สูง ก็จะแนะนำสิ่งของนั้นให้ผู้ใช้){\*}สำหรับข้อมูลที่มีการตอบรับโดยตรง (explicit feedback) ค่า ri,j ที่ทราบจะถูกให้โดยผู้ใช้ i โดยตรง ผู้ใช้สามารถให้คะแนนหรือประเมินสิ่งของเหล่านั้นให้กับระบบอีคอมเมิร์ซ โดยการถามผู้ใช้ให้ให้คะแนนสำหรับบางรายการสินค้าที่ต้องการประเมินในระบบของอีคอมเมิร์ซ[\*]52--สำหรับข้อมูลที่มีการตอบรับแบบอ้อม (implicit feedback) ค่า ri,j ที่ทราบจะถูกสร้างขึ้นโดยระบบโดยอัตโนมัติ โดยการตรวจสอบการกระทำต่างๆ ของผู้ใช้ เช่น ประวัติการซื้อของ, ประวัติการเดินทาง, เวลาที่ใช้บนหน้าเว็บบางหน้า, ลิงก์ที่ผู้ใช้ได้กดเข้าไป เป็นต้น ข้อมูลแบบอ้อมช่วยลดภาระของผู้ใช้ในการให้ข้อดีของตน แต่อาจจะมีความแม่นยำที่น้อยลง{\*}การใช้ข้อมูลแบบอ้อมสามารถเพิ่มประสิทธิภาพในการทำนายและการแนะนำสิ่งของได้ โดยไม่ต้องขึ้นอยู่กับผู้ใช้ในการให้ข้อมูลแบบแสดงความคิดเห็น แต่ความแม่นยำอาจจะลดลงเนื่องจากข้อมูลอาจไม่เป็นที่แม่นยำเท่าที่ควร หรืออาจขาดการเปรียบเทียบระหว่างการกระทำของผู้ใช้และความชอบที่แท้จริงของพวกเขา[\*]53--ในกระบวนการของการกรองร่วม (Collaborative Filtering) เริ่มต้นด้วยการสร้างฐานข้อมูลในรูปแบบของเมทริกซ์ผู้ใช้-สิ่งของ (User-Item Matrix) เพื่อเก็บข้อมูลเกี่ยวกับความชอบหรือคะแนนของผู้ใช้ต่อสิ่งของ{\*}ค่าที่ไม่ทราบเกี่ยวกับความชอบหรือคะแนน ri,j สามารถได้มาจากเทคนิคต่างๆ ซึ่งรวมถึง:{\*}1. เทคนิคเก็บความจำ (Memory techniques): ในการทำนายค่า ri,j รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร

Description automatically generatedนั้น จะใช้วิธีการหาค่าเฉลี่ยของความชอบหรือคะแนนของผู้ใช้ที่อยู่ใกล้เคียง (neighbors) ของผู้ใช้ i หรือสิ่งของ j โดยใช้วิธีการ K-nearest Neighbors (KNN) เพื่อค้นหาผู้ใช้หรือสิ่งของที่อยู่ใกล้เคียงกัน จากนั้นคำนวณค่าเฉลี่ยของความชอบหรือคะแนนของผู้ใช้เหล่านั้นเพื่อใช้เป็นค่าทำนายสำหรับ ri,j[\*]54--ในการกรองร่วมแบบพื้นฐานของผู้ใช้ (User-based Collaborative Filtering) มีวิธีการคำนวณค่า ri,j อย่างน้อยสองวิธี ดังนี้:{\*}1. \*\*คำนวณความคล้ายคลึงระหว่างผู้ใช้ (User Similarity)\*\*: วิธีนี้เริ่มต้นด้วยการคำนวณค่าความคล้ายคลึงระหว่างผู้ใช้ โดยใช้คะแนนหรือความชอบของผู้ใช้ต่อสิ่งของเหล่านั้น โดยมักใช้วิธีการคล้ายคลึงเชิงความคล้ายคลึงระหว่างเวกเตอร์ของคะแนนหรือความชอบของผู้ใช้ เมื่อเราคำนวณได้ค่าความคล้ายคลึงระหว่างผู้ใช้ทั้งหมด เราสามารถเลือกผู้ใช้ K คนที่มีความคล้ายคลึงที่สุดกับผู้ใช้ i (K-nearest neighbors) เพื่อนำไปใช้ในการคำนวณค่า ri,j โดยการใช้ค่าเฉลี่ยของคะแนนหรือความชอบของผู้ใช้เหล่านั้นเป็นน้ำหนักในการคำนวณ{\*}2. \*\*คำนวณค่า ri,j ด้วยค่าเฉลี่ยเพื่อนบ้าน (Weighted Average of Neighbors)\*\*: เมื่อเราเลือกผู้ใช้ K คนที่มีความคล้ายคลึงที่สุด ในขั้นตอนนี้เราคำนวณค่า ri,j โดยใช้ค่าเฉลี่ยของคะแนนหรือความชอบของผู้ใช้เหล่านั้น โดยให้ค่าแต่ละคะแนนหรือความชอบของผู้ใช้เป็นน้ำหนักในการคำนวณ นั่นคือ ค่า ri,j จะเป็นผลคูณระหว่างคะแนนหรือความชอบของแต่ละเพื่อนบ้านและค่าความคล้ายคลึงระหว่างผู้ใช้ i กับเพื่อนบ้านนั้นๆ[\*]56--ในการกรองร่วม (Collaborative Filtering) มีวิธีการวัดความคล้ายคลึงระหว่างผู้ใช้ที่นิยมที่สุดสองวิธี คือการใช้ค่าความสัมพันธ์ทางสถิติ (Correlation-based) เช่น ค่าสหสัมพันธ์ของ Pearson (Pearson’s correlation coefficient){\*}ค่าสหสัมพันธ์ของ Pearson เป็นวิธีการวัดความสัมพันธ์ระหว่างค่าของตัวแปรสองตัว ในกรณีของการกรองร่วม นั่นคือความสัมพันธ์ระหว่างการจัดอันดับหรือคะแนนของผู้ใช้ต่อสิ่งของเหล่านั้น วิธีการนี้มักถูกใช้เนื่องจากความง่ายในการเข้าใจและการใช้งาน โดยค่าสหสัมพันธ์ของ Pearson จะมีค่าอยู่ในช่วง -1 ถึง 1:{\*}- ค่าที่มีความสัมพันธ์เป็นบวกแสดงถึงความสัมพันธ์ที่เชิงบวก (positive correlation) คือเมื่อค่าหนึ่งเพิ่มขึ้น ค่าอีกตัวก็จะเพิ่มขึ้นด้วย{\*}- ค่าที่มีความสัมพันธ์เป็นลบแสดงถึงความสัมพันธ์ที่เชิงลบ (negative correlation) คือเมื่อค่าหนึ่งเพิ่มขึ้น ค่าอีกตัวก็จะลดลงด้วย{\*}- ค่าที่เข้ารูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, ไลน์

Description automatically generatedใกล้ 0 แสดงถึงความสัมพันธ์ที่น้อยหรือไม่มีความสัมพันธ์เลย (no correlation){\*}การใช้ค่าสหสัมพันธ์ของ Pearson ในการวัดความคล้ายคลึงระหว่างผู้ใช้ทำให้เราสามารถหาผู้ใช้ที่มีพฤติกรรมในการให้คะแนนหรือจัดอันดับสิ่งของที่คล้ายคลึงกับผู้ใช้ i ได้และนำมาใช้ในการกรองร่วมได้อย่างมีประสิทธิภาพ[\*]57--การกรองการทำงานร่วมกัน{\*}• ความคล้ายคลึงกันของโคไซน์ ขึ้นอยู่กับปริภูมิเวกเตอร์{\*}โมเดลที่มองแต่ละอินสแตนซ์ข้อมูลเป็น{\*}เวกเตอร์ มันวัดความคล้ายคลึงกันระหว่างสอง{\*}อินสแตนซ์ข้อมูลขึ้นอยู่กับค่าโคไซน์ของ{\*}มุมระหว่างเวกเตอร์สองตัวที่แสดงถึง{\*}สองอินสแตนซ์ข้อมูล [\*]58--การกรองร่วมแบบข้อมูลสิ่งของ (Item-based Collaborative Filtering) เป็นเทคนิคที่คำนวณค่า ri,j โดยใช้ค่าความคล้ายคลึงระหว่างสิ่งของแทนการใช้ค่าความคล้ายคลึงระหว่างผู้ใช้ ค่าที่ทำนายได้สามารถคำนวณได้โดยการใช้ค่าเฉลี่ยของคะแนนระหว่างเพื่อนบ้านที่เป็นสิ่งของ k รายการ{\*}การใช้การกรองร่วมแบบข้อมูลสิ่งของเป็นทางเลือกที่พบบ่อยสำหรับบริการที่มีภาระการทำงานสูง (high load services) ซึ่งมีจำนวนผู้ใช้และสิ่งของมาก สาเหตุหลักคือว่าชุดข้อมูลของสิ่งของมักมีการเปลี่ยนแปลงช้ากว่าชุดข้อมูลของผู้ใช้{\*}นั่นคือ ชุดข้อมูลเพื่อนบ้านของสิ่งของมักมีการเปลี่ยนแปลงช้าและมีความเสถียรมากกว่าชุดข้อมูลเพื่อนบ้านของผู้ใช้ ซึ่งทำให้การกรองร่วมแบบข้อมูลสิ่งของมีประสิทธิภาพสูงในการจัดอันดับและแนะนำสิ่งของในสถานการณ์ที่มีจำนวนผู้ใช้และสิ่งของมาก[\*]59--การกรองร่วมแบบมีโมเดล (Model-based Collaborative Filtering) เป็นเทคนิคที่ใช้โมเดลทางสถิติหรือแบบจำลองเพื่อการทำนายคะแนนหรือความชอบของผู้ใช้ โดยที่ไม่ต้องพึ่งพาข้อมูลของเพื่อนบ้านหรือความคล้ายคลึงระหว่างผู้ใช้{\*}MLlib เป็นไลบรารีสำหรับการประมวลผลข้อมูลขนาดใหญ่ใน Spark ซึ่งเน้นการแนะนำของผู้ใช้โดยใช้เทคนิค Matrix Completion โดยเทคนิคที่นิยมที่สุดคือ Alternating Least Squares (ALS){\*}เมื่อเรามีเมทริกซ์ของผู้ใช้-สิ่งของ M ขนาด m x n โดยส่วนใหญ่ของรายการใน M จะไม่ทราบค่า งานของการทำการสำเร็จแล้ว คือการประมาณค่าของรายการในเมทริกซ์โดยใช้เทคนิค Matrix Completion[\*]60--ในแบบจำลอง Collaborative Filtering ที่ใช้เทคนิค Matrix Completion โดยใช้สมการ M = UVT รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร

Description automatically generatedโดยที่:{\*}- M เป็นเมทริกซ์ของผู้ใช้-สิ่งของ ขนาด m x n{\*}- U เป็นเมทริกซ์ของผู้ใช้ ขนาด m x k{\*}- V เป็นเมทริกซ์ของสิ่งของ ขนาด n x k{\*}- k เป็นระดับ (rank) ของโมเดล ซึ่งมักเป็นค่าที่น้อยกว่าหรือเท่ากับค่าน้อยที่สุดระหว่าง m และ n{\*}ในเทคนิคนี้ เราจะทำการแยกประเภทหรือรูปร่างของข้อมูลเพื่อให้ได้โมเดลที่มีความแท้จริง (model) มากที่สุด ซึ่งสามารถประมาณค่าของรายการที่ไม่ทราบในเมทริกซ์ M ได้โดยใช้เพียง k ค่าน้อยที่สุดที่จำเป็นในการอธิบายข้อมูลใน M ด้วยประสิทธิภาพสูงสุด[\*]62--วิธีการ ALS (Alternating Least Squares) ใช้ใน Collaborative Filtering มีขั้นตอนการทำงานดังนี้:{\*}ขั้นตอนที่ 1: กำหนดค่าเริ่มต้นของเมทริกซ์ V โดยกำหนดค่าเฉลี่ยของคะแนนสำหรับแถวแรกและกำหนดค่าสุ่มเล็กน้อยสำหรับรายการที่เหลือ{\*}ขั้นตอนที่ 2: ระบุ V และแก้ปัญหา U โดยการลดค่าฟังก์ชันขาดทุน (RMSE) โดยใช้วิธีเลียสต์สแควร์เชิงเส้นที่มีการปรับปรุงเพื่อลดการ overfitting{\*}ขั้นตอนที่ 3: ระบุ U และแก้ปัญหา V โดยการลดค่าฟังก์ชันขาดทุน (RMSE) โดยใช้วิธีเลียสต์สแควร์เชิงเส้นที่มีการปรับปรุงเพื่อลดการ overfitting{\*}ขั้นตอนที่ 4: ทำซ้ำขั้นตอนที่ 2 และ 3 จนกระทั่งการลู่เข้าถึงค่าที่ต้องการ หรือตามเงื่อนไขการหยุดที่กำหนด{\*}ในขั้นตอนที่ 2 และ 3 การแก้ปัญหาจะใช้วิธีการแบบเลียร์สแควร์เชิงเส้นที่ปรับปรุงมาเพื่อลดการ overfitting ซึ่งการลดการ overfitting จะช่วยลดความเสี่ยงที่โมเดลจะทำนายผลลัพธ์ที่ไม่แม่นยำสำหรับข้อมูลใหม่โดยการควบคุมความซับซ้อนของโมเดลและปรับพารามิเตอร์ที่เหมาะสม[\*]63--การใช้เมทอด train ของออบเจ็กต์ ALS จะทำการฝึกหรือจับคู่กับโมเดล MatrixFactorizationModel ด้วย RDD ของการจัดอันดับ (Rating) โดยมันจะรับ RDD ของการจัดอันดับ (Rating) และพารามิเตอร์ที่เฉพาะเจาะจงตาม ALS และส่งคืนอินสแตนซ์ของคลาส รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, ไลน์

Description automatically generatedmatrixFactorizationModelClass{\*}พารามิเตอร์รวมถึงจำนวนคุณลักษณะที่ซ่อนอยู่ (latent features) หรือค่า rank, จำนวนการทำซ้ำ, ตัวปรับ regularization, ระดับของการขนาน, และ random seed โดยพารามิเตอร์สองตัวสุดท้ายเป็นไม่บังคับใช้ โดยทั่วไปเมื่อต้องการสร้างโมเดล ALS นักพัฒนาจะกำหนดค่าพารามิเตอร์เหล่านี้เพื่อปรับการทำงานของโมเดลให้เหมาะสมกับข้อมูลและการทำงานที่ต้องการ[\*]64--โค้ดด้านล่างเป็นตัวอย่างการใช้งาน ALS ใน Spark สำหรับการสร้างโมเดลการแนะนำโดยใช้ข้อมูลจากไฟล์ข้อความ:{\*}```scala{\*}import org.apache.spark.mllib.recommendation.ALS{\*}import org.apache.spark.mllib.recommendation.Rating{\*}// อ่านข้อมูลจากไฟล์ข้อความ{\*}val lines = sc.textFile("data/mllib/als/test.data"){\*}// แปลงข้อมูลในแต่ละบรรทัดเป็น Rating object{\*}val ratings = lines.map { line =>{\*}val Array(user, item, rate) = line.split(','){\*}Rating(user.toInt, item.toInt, rate.toDouble){\*}}{\*}// กำหนดค่า rank และจำนวนการทำซ้ำ{\*}val rank = 10{\*}val numIterations = 10{\*}// ฝึกโมเดล ALS ด้วยข้อมูลการจัดอันดับและพารามิเตอร์ที่กำหนด{\*}val mfModel = ALS.train(ratings, rank, numIterations, 0.01){\*}```{\*}ในโค้ดนี้:{\*}- รับการนำเข้าของคลาส ALS และ Rating{\*}- โหลดข้อมูลจากไฟล์ข้อความโดยใช้ `textFile` และแปลงแต่ละบรรทัดให้อยู่ในรูปแบบของ Rating object{\*}- กำหนดค่า rank และจำนวนการทำซ้ำ{\*}- ใช้เมทอด `train` ของ ALS เพื่อฝึกโมเดล ALS ด้วยรูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, จำนวน

Description automatically generatedข้อมูลการจัดอันดับและพารามิเตอร์ที่กำหนด{\*}- สร้างโมเดลและเก็บไว้ในตัวแปร `mfModel`[\*]65--เมธอด `trainImplicit` ใช้เมื่อมีข้อมูลเฉพาะเรื่องการตอบรับของผู้ใช้ต่อสินค้าที่ใช้เป็นข้อมูลอิมพลิชิต เมื่อใช้เมธอดนี้ ค่าตอบรับที่ได้ \( r\_{ui} \) (เช่น จำนวนการดูหน้าเว็บของสินค้า \( i \) สำหรับผู้ใช้ \( u \)) ควรถูกปรับให้อยู่ในช่วงระหว่าง 0 ถึง 1 แทนที่จะเป็นค่าอื่น ๆ แทนคะแนน การใช้ `trainImplicit` มุ่งหวังที่จะทำนายความชอบ \( p\_{ui} \) พร้อมค่าความมั่นใจ \( c\_{ui} \){\*}นอกจากนี้ ฟังก์ชันเป้าหมายสำหรับกระบวนการเรียนรู้ของเครื่องรวมถึงพารามิเตอร์การปรับปรุง \( \lambda \) ซึ่งช่วยป้องกันการฝึกเกินและควบคุมความซับซ้อนของโมเดลได้[\*]66--เมธอด `trainImplicit` รับพารามิเตอร์เป็น feedback RDD และพารามิเตอร์ที่เฉพาะเจาะจงตาม ALS และคืนค่าอินสแตนซ์ของคลาส MatrixFactorizationModel ซึ่งอ้างถึงวิจัยโดย Yifan Hu, Yehuda Koren, Chris Volinsky ในหัวข้อ "Collaborative Filtering for Implicit Feedback" โดยทั่วไปแล้ว การใช้ `trainImplicit` นั้นมักจะเหมาะสำหรับข้อมูลที่มีเฉพาะการตอบรับที่อิมพลิชิตและไม่ได้ให้คะแนนชัดเจนโดยผู้ใช้ เช่น การกดคลิกหน้าเว็บหรือการเข้าชมสินค้าโดยไม่มีการให้คะแนนโดยตรง และมักจะให้ความสำคัญกับความแน่นอนในการทำนายความชอบของผู้ใช้ในสินค้านั้น ๆ ที่มีการตอบรับอิมพลิชิตและเชื่อมั่นสูงขึ้น[\*]67—โค้ดด้านล่างเป็นตัวอย่างการใช้เมธอด `trainImplicit` ใน Spark ALS สำหรับการสร้างโมเดลการแนะนำโดยใช้ feedback RDD และการกำหนดพารามิเตอร์:{\*}```scala{\*}val rank = 10{\*}val numIterations = 10{\*}val alpha = 0.01 // พารามิเตอร์สำหรับแปลงค่าตอบรับเป็นค่าความเชื่อมั่น{\*}val lambda = 0.01 // พารามิเตอร์การปรับปรุงเพื่อการละเมิดทางปกติ{\*}// ใช้เมธอด trainImplicit เพื่อฝึกโมเดล ALS ด้วย feedback RDD และพารามิเตอร์ที่กำหนด{\*}val mfModel = ALS.trainImplicit(feedback, rank, numIterations, lambda, alpha){\*}```{\*}ในโค้ดนี้:{\*}- จำนวนคุณลักษณะที่ซ่อนอยู่ (rank) ถูกกำหนดเป็น 10{\*}- จำนวนการทำซ้ำถูกกำหนดเป็น 10{\*}- ค่าพารามิเตอร์ alpha ถูกกำหนดเป็น 0.01 ซึ่งเป็นพารามิเตอร์สำหรับแปลงค่าตอบรับให้อยู่ในรูปของค่าความมั่นในการทำนาย{\*}- ค่าพารามิเตอร์ lambda ถูกกำหนดเป็น 0.01 ซึ่งเป็นพารามิเตอร์การปรับปรุงเพื่อละเมิดทางปกติสำหรับการควบคุมความซับซ้อนของโมเดล{\*}- `trainImplicit` ถูกเรียกใช้เพื่อฝึกโมเดล ALS โดยใช้ feedback RDD และพารามิเตอร์ที่กำหนด และเก็บโมเดลที่ได้ในตัวแปร `mfModel`[\*]68--เมธอด predict ในคลาส MatrixFactorizationModel จะคืนค่าการให้คะแนนสำหรับผู้ใช้และสินค้าที่ระบุ{\*}การใช้งานเมธอด รูปภาพประกอบด้วย ข้อความ, ภาพหน้าจอ, ตัวอักษร, จำนวน

Description automatically generatedpredict มีรูปแบบดังนี้:{\*}- รับพารามิเตอร์ user id และ product id ซึ่งมีประเภทเป็น Int{\*}- คืนค่าการให้คะแนนที่คาดการณ์ได้ซึ่งมีประเภทเป็น Double{\*}ตัวอย่างการใช้งานเมธอด predict ด้วยโมเดลที่ชื่อ mfModel ได้แสดงด้านล่าง:{\*}```scala{\*}val userId = 1{\*}val prodId = 1{\*}val predictedRating = mfModel.predict(userId, prodId){\*}```{\*}ในตัวอย่างนี้:{\*}- userId และ prodId ถูกกำหนดค่าเป็น 1{\*}- เมื่อเรียกใช้งานเมธอด predict กับ userId และ prodId ดังกล่าว โมเดลจะคำนวณและคืนค่าการให้คะแนนที่คาดการณ์ได้สำหรับผู้ใช้และสินค้าที่ระบุ[\*]69--นี่คือตัวอย่างของการใช้ `predict` ที่ยอมรับ RDD ของคู่ user id และ product id และคืนค่าเป็น RDD ของ Rating:{\*}```scala{\*}// แปลง Rating ใน RDD ให้เป็นคู่ของ user id และ product id{\*}val usersProducts = ratings.map { case Rating(user, product, rate) => (user, product) }{\*}// เรียกใช้เมธอด predict โดยใช้ RDD ของคู่ user id และ product id{\*}val predictions = mfModel.predict(usersProducts){\*}// เรียกใช้เมธอด take(5) เพื่อรับรายการ Rating 5 รายการแรก{\*}val firstFivePredictions = predictions.take(5){\*}```{\*}การทำงานของโค้ดนี้คือ:{\*}- ระบบจะแปลงค่า Rating ใน RDD ให้เป็นคู่ของ user id และ product id โดยใช้เมธอด map{\*}- เมื่อเรียกใช้เมธอด predict โดยใช้ RDD ของคู่ user id และ product id ระบบจะคำนวณและคืนค่า Rating สำหรับทุกคู่ user id และ product id{\*}- เมธอด take(5) ถูกระบุเพื่อรับรายการ Rating 5 รายการแรกที่คำนวณได้จากโมเดล[\*]70--เมธอด recommendProducts ในคลาส MatrixFactorizationModel ใช้ในการแนะนำจำนวนสินค้าที่ระบุสำหรับผู้ใช้ที่กำหนด โดยเมธอดนี้จะคืนค่าเป็นอาร์เรย์ของอ็อบเจกต์ Rating โดยที่แต่ละ Rating จะระบุ user id, product id และคะแนนการให้คะแนนที่คาดการณ์ได้สำหรับสินค้านั้น ๆ{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}val userId = 1{\*}val numProducts = 3{\*}val recommendedProducts = mfModel.recommendProducts(userId, numProducts){\*}```{\*}ในตัวอย่างนี้:{\*}- userId คือ ID ของผู้ใช้ที่ต้องการทำการแนะนำสินค้า{\*}- numProducts คือจำนวนสินค้าที่ต้องการทำการแนะนำ{\*}ผลลัพธ์ที่คืนจะเป็นอาร์เรย์ของ Rating โดยที่สินค้าที่แนะนำจะถูกเรียงตามคะแนนการให้คะแนนที่คาดการณ์ได้ลงมาจากสูงสุดไปยังต่ำสุดในอาร์เรย์ที่คืนค่า[\*]71--เมธอด recommendProductsForUsers ในคลาส MatrixFactorizationModel ใช้ในการแนะนำจำนวนสินค้าที่ระบุสำหรับทุกผู้ใช้ โดยเมธอดนี้จะคืนค่าเป็น RDD ที่ประกอบด้วยข้อมูลของผู้ใช้แต่ละคนและสินค้าที่แนะนำสำหรับแต่ละผู้ใช้{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}val numProducts = 2{\*}val recommendedProductsForAllUsers = mfModel.recommendProductsForUsers(numProducts){\*}```{\*}ในตัวอย่างนี้:{\*}- numProducts คือจำนวนสินค้าที่ต้องการทำการแนะนำ{\*}ผลลัพธ์ที่คืนคือ RDD ที่ประกอบด้วยข้อมูลของผู้ใช้แต่ละคนและสินค้าที่แนะนำสำหรับแต่ละผู้ใช้ โดยในแต่ละรายการจะมี user id และอาร์เรย์ของ Rating ซึ่งแต่ละ Rating จะระบุ user id, product id และคะแนนการให้คะแนนที่คาดการณ์ได้สำหรับสินค้านั้น ๆ สำหรับผู้ใช้นั้น ๆ[\*]72--เมธอด recommendUsers ในคลาส MatrixFactorizationModel ใช้ในการแนะนำจำนวนผู้ใช้ที่ระบุสำหรับสินค้าที่กำหนด โดยเมธอดนี้จะคืนค่าเป็นอาร์เรย์ของ Rating โดยที่แต่ละอ็อบเจกต์ Rating จะระบุ user id, product id และคะแนนในฟิลด์ rating{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}val productId = 2{\*}val numUsers = 3{\*}val recommendedUsers = mfModel.recommendUsers(productId, numUsers){\*}```{\*}ในตัวอย่างนี้:{\*}- productId คือ ID ของสินค้าที่ต้องการทำการแนะนำผู้ใช้{\*}- numUsers คือจำนวนผู้ใช้ที่ต้องการทำการแนะนำ{\*}ผลลัพธ์ที่คืนคืออาร์เรย์ของ Rating โดยที่แต่ละ Rating จะระบุ user id, product id และคะแนนในฟิลด์ rating สำหรับผู้ใช้ที่แนะนำสำหรับสินค้าที่กำหนด[\*]73--เมธอด recommendUsers ในคลาส MatrixFactorizationModel ใช้ในการแนะนำจำนวนผู้ใช้ที่ระบุสำหรับสินค้าที่กำหนด โดยเมธอดนี้จะคืนค่าเป็นอาร์เรย์ของอ็อบเจกต์ Rating โดยที่แต่ละอ็อบเจกต์ Rating จะระบุ user id, product id และคะแนนในฟิลด์ rating{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}val productId = 2{\*}val numUsers = 3{\*}val recommendedUsers = mfModel.recommendUsers(productId, numUsers){\*}```{\*}ในตัวอย่างนี้:{\*}- productId คือ ID ของสินค้าที่ต้องการทำการแนะนำผู้ใช้{\*}- numUsers คือจำนวนผู้ใช้ที่ต้องการทำการแนะนำ{\*}ผลลัพธ์ที่คืนคืออาร์เรย์ของ Rating โดยที่แต่ละ Rating จะระบุ user id, product id และคะแนนในฟิลด์ rating สำหรับผู้ใช้ที่แนะนำสำหรับสินค้าที่กำหนด[\*]74--เมธอด recommendUsersForProducts ในคลาส MatrixFactorizationModel ใช้ในการแนะนำจำนวนผู้ใช้ที่ระบุสำหรับทุกสินค้า โดยเมธอดนี้จะคืนค่าเป็น RDD ที่ประกอบด้วยข้อมูลของสินค้าแต่ละชิ้นและผู้ใช้ที่แนะนำสำหรับแต่ละสินค้า{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}val numUsers = 2{\*}val recommendedUsersForAllProducts = mfModel.recommendUsersForProducts(numUsers){\*}val ruFor4Products = recommendedUsersForAllProducts.take(4){\*}```{\*}ในตัวอย่างนี้:{\*}- numUsers คือจำนวนผู้ใช้ที่ต้องการทำการแนะนำ{\*}- ruFor4Products เป็นอาร์เรย์ของ Rating สำหรับ 4 ผลิตภัณฑ์แรกที่ได้รับการแนะนำสำหรับแต่ละผู้ใช้{\*}ผลลัพธ์ที่คืนคือ RDD ที่ประกอบด้วยข้อมูลของสินค้าแต่ละชิ้นและผู้ใช้ที่แนะนำสำหรับแต่ละสินค้า โดยในแต่ละรายการจะมี product id และอาร์เรย์ของ Rating ซึ่งแต่ละ Rating จะระบุ product id, user id และคะแนนในฟิลด์ rating สำหรับผู้ใช้ที่แนะนำสำหรับสินค้าที่กำหนด[\*]75--เมธอด save ใช้ในการบันทึก MatrixFactorizationModel ลงบนดิสก์ โดยเมธอดนี้จะรับ SparkContext และเส้นทางของไฟล์ที่ต้องการบันทึกเป็นอาร์กิวเมนต์{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}mfModel.save(sc, "models/mf-model"){\*}```{\*}เมธอด load ใช้ในการอ่านโมเดลที่บันทึกไว้ก่อนหน้าจากไฟล์ โดยเมธอดนี้จะรับ SparkContext และเส้นทางของไฟล์ที่บันทึกเป็นอาร์กิวเมนต์{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}import org.apache.spark.mllib.recommendation.MatrixFactorizationModel{\*}val savedmfModel = MatrixFactorizationModel.load(sc, "models/mf-model"){\*}```{\*}ในตัวอย่างนี้:{\*}- sc เป็น SparkContext{\*}- "models/mf-model" เป็นเส้นทางไฟล์ที่โมเดลถูกบันทึกไว้และโหลดมาใช้งาน[\*]76--การประเมินโมเดลในงาน Regression สามารถทำได้โดยใช้คลาส RegressionMetrics ซึ่งสามารถคำนวณค่า mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE) และอื่น ๆ ได้{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}import org.apache.spark.mllib.evaluation.RegressionMetrics{\*}val numIterations = 100{\*}val lrModel = linearRegressionwithSGD.train(labeledPoints, numIterations){\*}// คำนวณค่าที่คาดการณ์และค่าจริงโดยใช้โมเดล{\*}val observedAndPredictedLabels = labeledPoints.map { observation =>{\*}val predictedLabel = lrModel.predict(observation.features){\*}(observation.label, predictedLabel){\*}}{\*}// สร้าง RegressionMetrics object โดยใช้ค่าที่คาดการณ์และค่าจริง{\*}val regressionMetrics = new RegressionMetrics(observedAndPredictedLabels){\*}// คำนวณค่า Mean Squared Error{\*}val mse = regressionMetrics.meanSquaredError{\*}// คำนวณค่า Root Mean Squared Error{\*}val rmse = regressionMetrics.rootMeanSquaredError{\*}// คำนวณค่า Mean Absolute Error{\*}val mae = regressionMetrics.meanAbsoluteError{\*}```{\*}ในตัวอย่างนี้:{\*}- labeledPoints คือ RDD ของ LabeledPoint ซึ่งเป็นข้อมูลสำหรับการฝึกฝนโมเดล Linear Regression{\*}- lrModel คือโมเดล Linear Regression ที่ได้ฝึกฝนขึ้น{\*}- observedAndPredictedLabels เป็น RDD ที่ประกอบด้วยค่าตั้งแต่และค่าที่คาดการณ์จากโมเดลสำหรับแต่ละจุดข้อมูล{\*}- regressionMetrics เป็น RegressionMetrics object ที่ใช้ในการคำนวณค่าประสิทธิภาพของโมเดล{\*}- mse, rmse, mae คือค่า Mean Squared Error, Root Mean Squared Error, และ Mean Absolute Error ตามลำดับ[\*]77--โค้ดที่ให้มาด้านล่างนั้นใช้ในการคำนวณค่าความแม่นยำของโมเดล Regression โดยใช้ RegressionMetrics โดยกำหนดค่า MSE (Mean Squared Error), RMSE (Root Mean Squared Error), และ MAE (Mean Absolute Error) โดยตรงจากข้อมูลที่คาดการณ์และข้อมูลจริงที่ได้จากโมเดล Linear Regression:{\*}```scala{\*}val regressionMetrics = new RegressionMetrics(observedAndPredictedLabels){\*}val mse = regressionMetrics.meanSquaredError{\*}val rmse = regressionMetrics.rootMeanSquaredError{\*}val mae = regressionMetrics.meanAbsoluteError{\*}```{\*}โดยที่:{\*}- `observedAndPredictedLabels` เป็นข้อมูลที่ประกอบด้วยค่าที่คาดการณ์และค่าจริงจากโมเดล{\*}- `regressionMetrics` เป็นอ็อบเจกต์ RegressionMetrics ที่ใช้ในการคำนวณค่าความแม่นยำ{\*}- `mse`, `rmse`, และ `mae` เป็นค่า MSE, RMSE, และ MAE ตามลำดับที่คำนวณได้จาก RegressionMetrics[\*]78--คลาส BinaryClassificationMetrics ใช้สำหรับการประเมินตัวแยกประเภทแบบไบนารี (binary classifiers) ซึ่งมีเมท็อดสำหรับคำนวณ Receiver Operating Characteristics (ROC) curve (เส้นโค้ง TPR vs FPR) และพื้นที่ใต้เส้น ROC (AUC){\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics{\*}val svmModel = SVMWithSGD.train(trainingData, numIterations){\*}// คำนวณค่าที่คาดการณ์และค่าจริงโดยใช้โมเดล{\*}val predictedAndActualLabels = testData.map { observation =>{\*}val predictedLabel = svmModel.predict(observation.features) // คำนวณค่าที่คาดการณ์ (ค่านี้จริง ๆ คือคะแนนระหว่าง 0 และ 1){\*}(predictedLabel, observation.label){\*}}{\*}// สร้าง BinaryClassificationMetrics object โดยใช้ค่าที่คาดการณ์และค่าจริง{\*}val metrics = new BinaryClassificationMetrics(predictedAndActualLabels){\*}// คำนวณพื้นที่ใต้เส้น ROC (AUC){\*}val auROC = metrics.areaUnderROC(){\*}```{\*}ในตัวอย่างนี้:{\*}- `svmModel` คือโมเดล SVM ที่ได้ฝึกฝนขึ้น{\*}- `predictedAndActualLabels` เป็นข้อมูลที่ประกอบด้วยค่าที่คาดการณ์และค่าจริง{\*}- `metrics` เป็นอ็อบเจกต์ BinaryClassificationMetrics ที่ใช้ในการคำนวณค่าประสิทธิภาพ{\*}- `auROC` เป็นค่าพื้นที่ใต้เส้น ROC (AUC) ซึ่งเป็นค่าประสิทธิภาพของโมเดลในการแบ่งแยกคลาสในรูปแบบ binary classification โดยใช้เส้น ROC ที่ได้จาก BinaryClassificationMetrics โดยค่านี้สามารถใช้ในการประเมินประสิทธิภาพของโมเดลได้[\*]79--โค้ดที่ให้มานั้นใช้ในการคำนวณค่าประสิทธิภาพของโมเดล Binary Classification โดยใช้ BinaryClassificationMetrics โดยกำหนดค่า Area under the Receiver Operating Characteristic (auROC) โดยตรงจากข้อมูลที่คาดการณ์และข้อมูลจริงที่ได้จากโมเดล:{\*}```scala{\*}val metrics = new BinaryClassificationMetrics(predictedAndActualLabels){\*}val auROC = metrics.areaUnderROC(){\*}```{\*}โดยที่:{\*}- `predictedAndActualLabels` เป็นข้อมูลที่ประกอบด้วยค่าที่คาดการณ์และค่าจริงจากโมเดล{\*}- `metrics` เป็นอ็อบเจกต์ BinaryClassificationMetrics ที่ใช้ในการคำนวณค่าประสิทธิภาพ{\*}- `auROC` เป็นค่า Area under the Receiver Operating Characteristic ที่คำนวณได้จาก BinaryClassificationMetrics ซึ่งเป็นค่าประสิทธิภาพของโมเดลในการแบ่งแยกคลาสในรูปแบบ binary classification โดยใช้เส้น ROC[\*]80--สำหรับการประเมินตัวแยกประเภทแบบหลายคลาส (multi-class classifiers) คลาส MulticlassMetrics ให้เมท็อดสำหรับคำนวณ precision, recall, F-Measure เป็นต้น{\*}ตัวอย่างการใช้งาน:{\*}```scala{\*}import org.apache.spark.mllib.classification.NaiveBayes{\*}import org.apache.spark.mllib.evaluation.MulticlassMetrics{\*}// ฝึกโมเดล Naive Bayes{\*}val model = NaiveBayes.train(trainingData){\*}// คำนวณค่าที่คาดการณ์และค่าจริงโดยใช้โมเดล{\*}val predictionsAndLabels = testData.map { d =>{\*}(model.predict(d.features), d.label){\*}}{\*}// สร้าง MulticlassMetrics object โดยใช้ค่าที่คาดการณ์และค่าจริง{\*}val metrics = new MulticlassMetrics(predictionsAndLabels){\*}// คำนวณ precision, recall, F-Measure{\*}val recall = metrics.recall{\*}val precision = metrics.precision{\*}val fMeasure = metrics.fMeasure{\*}```{\*}ในตัวอย่างนี้:{\*}- `model` คือโมเดล Naive Bayes ที่ได้ฝึกฝนขึ้น{\*}- `predictionsAndLabels` เป็นข้อมูลที่ประกอบด้วยค่าที่คาดการณ์และค่าจริง{\*}- `metrics` เป็นอ็อบเจกต์ MulticlassMetrics ที่ใช้ในการคำนวณค่า precision, recall, F-Measure{\*}- `recall`, `precision`, และ `fMeasure` เป็นค่า precision, recall, และ F-Measure ตามลำดับที่คำนวณได้จาก MulticlassMetrics โดยค่าเหล่านี้สามารถใช้ในการประเมินประสิทธิภาพของโมเดลได้[\*]81--รูปภาพประกอบด้วย ข้อความ, แผนภาพ, ไลน์, ภาพหน้าจอ

Description automatically generated

27

17

9

8