รหัสโครงการ SUT7-705-48-24-11

รายงานการวิจัย

การพัฒนาการทำเหมืองข้อมูลแบบจัดกลุ่ม

(The development of clustering data mining)

ผู้วิจัย

หัวหน้าโครงการ

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สำนักวิชาวิศวกรรมศาสตร์

ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ พ.ศ. 2548 และ 2549

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

ธันวาคม 2552

กิตติกรรมประกาศ

ผู้วิจัยขอขอบคุณมหาวิทยาลัยเทคโนโลยีสุรนารีและสำนักงานคณะกรรมการวิจัยแห่งชาติ ที่ได้จัดสรรงบประมาณในการทำวิจัยให้ในปีงบประมาณ 2548 และ 2549 โครงการนี้ยังได้รับงบประมาณบางส่วน รวมถึงความร่วมมือในการดำเนินงานจากหน่วยปฏิบัติการวิจัยด้านวิศวกรรมข้อมูลและการค้นหาความรู้ (Data Engineering and Knowledge Discovery -- DEKD -- Research Unit) ขอขอบคุณผู้ทรงคุณวุฒิที่ได้เสียสละเวลาทำหน้าที่ตรวจข้อเสนอโครงการ และตรวจร่างรายงานการวิจัยฉบับสมบูรณ์

บทคัดย่อภาษาไทย

การจัดกลุ่มข้อมูล เป็นงานรวมกลุ่มข้อมูลด้วยการพิจารณาความคล้ายคลึงกัน อัลกอริทึม k-means เป็นอัลกอริทึมมาตรฐานที่ทำการรวมกลุ่มข้อมูล โดยเริ่มต้นกระบวนการด้วยการคำนวณระยะห่างเพื่อกำหนดข้อมูลแต่ละตัวเข้ากลุ่มที่อยู่ใกล้ที่สุด จากนั้นคำนวณค่าจุดกึ่งกลางของแต่ละกลุ่มโดยใช้ค่าเฉลี่ยของข้อมูลทั้งหมดในกลุ่ม อัลกอริทึมจะทำสองขั้นตอนนี้ซ้ำจนกระทั่งค่าจุดกึ่งกลางไม่มีการเปลี่ยนแปลงและข้อมูลไม่มีการเปลี่ยนกลุ่ม ผู้วิจัยได้เสนออัลกอริทึมชื่อ density-biased clustering เพื่อปรับปรุง k-means ให้ทำงานกับข้อมูลขนาดใหญ่ได้รวดเร็ว ความเร็วในการทำงานจะได้จากขั้นตอนการสุ่มด้วยเทคนิคการสุ่มตามความหนาแน่น เพื่อคัดเลือกข้อมูลตัวแทนที่มีความหนาแน่นสูง ค่าความหนาแน่นคำนวณจากจำนวนข้อมูลที่อยู่ใกล้เคียง จากนั้นนำข้อมูลตัวแทนที่คัดเลือกไว้เข้าสู่กระบวนการจัดกลุ่ม งานวิจัยนี้ยังได้ปรับปรุงการจัดกลุ่มข้อมูลให้เป็นการจัดกลุ่มแบบเพิ่มพูนเพื่อให้สามารถจัดกลุ่มข้อมูลขนาดใหญ่ที่มีขนาดของกลุ่มแตกต่างกันได้ ผลการทดลองได้แสดงว่าวิธีการจัดกลุ่มข้อมูลตามความหนาแน่นสามารถคัดเลือกข้อมูลที่เป็นตัวแทนได้ดี และอัลกอริทึมทำงานได้ผลดีกับข้อมูลขนาดใหญ่ที่มีจำนวนมิติปานกลาง (มีจำนวนมิติหรือแอททริบิวต์ 8 ถึง 23 แอททริบิวต์) การจัดกลุ่มกับข้อมูลแบบเพิ่มพูนกับข้อมูลที่ถูกคัดเลือกตามความหนาแน่นให้ผลการจัดกลุ่มที่ใกล้เคียงกับวิธี k-means แต่ใช้เวลาประมวลผลที่น้อยกว่า

บทคัดย่อภาษาอังกฤษ

Clustering is a task of grouping data based on similarity. A standard k-means algorithm groups data by firstly assigning all data points to the closest clusters, then determining the new cluster mean based on the average values of its members. The algorithm repeats these two steps until it converges; that is until there is no change in cluster assignment among data points. We propose a variation called incremental density-biased clustering to improve the scalability of the clustering process. To speed up the clustering process, we develop the density-biased clustering algorithm as an efficient data clustering technique. Efficiency is due to the reduced data set. Density of each data point is calculated based on its surrounding neighbors. Only dense data are selected for further clustering. We also implement the incremental clustering algorithm to get the benefit from its advantage of efficient memory usage and extend it to deal with clustering large data of varying cluster sizes. Our experimental results reveal the effectiveness of drawing samples of high density from large data sets of moderate dimensions (approximately 8-23 attributes), then incrementally grouping these data into clusters. The clustering results produce almost the same results as the standard k-means, but our incremental algorithm takes less computational time.

สารบัญ

สารบัญตาราง

หน้า

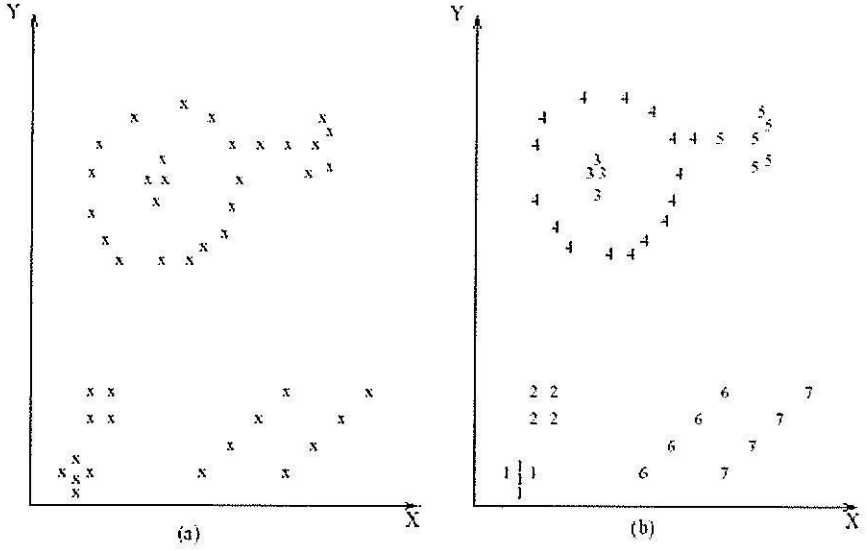
สารบัญภาพ

หน้า

บทที่ 1
บทนำ

ความสำคัญและที่มาของปัญหาการวิจัย

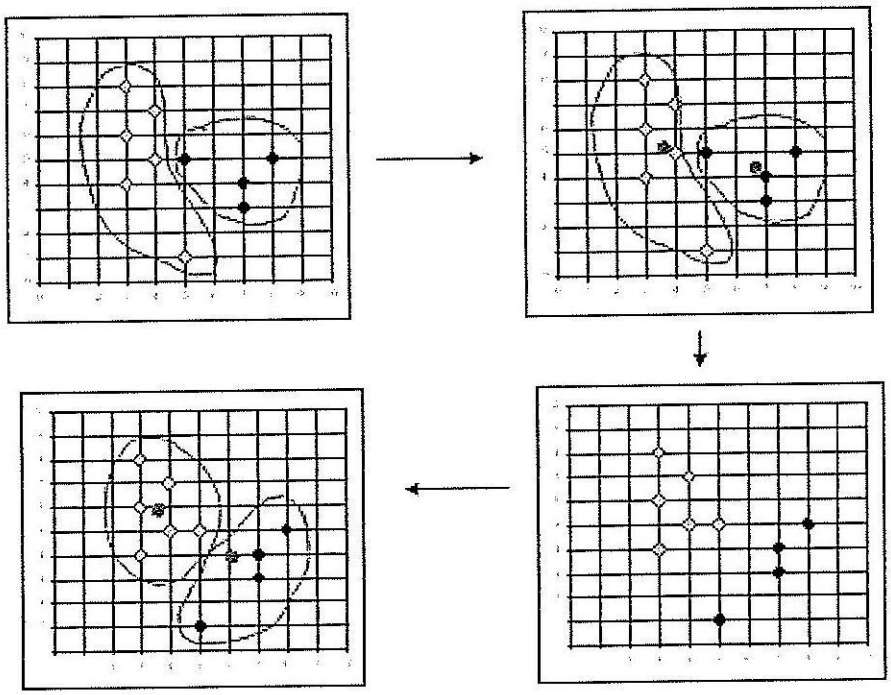
การจัดกลุ่มข้อมูล (data clustering) เป็นงานที่สำคัญและมีการนำไปใช้ประโยชน์ทั้งในด้านการทำเหมืองข้อมูล (data mining) การวิเคราะห์ข้อมูลเชิงสถิติ และด้านการรู้จำและวิเคราะห์รูปแบบ (pattern recognition and analysis) ในการทำเหมืองข้อมูล ข้อมูลจะต้องได้รับการจัดเป็นกลุ่ม (group or class or cluster) ก่อนที่จะมีการค้นหารูปแบบเพื่อการจำแนก (classification) หรือรูปแบบความสัมพันธ์ (association) ที่ซ่อนอยู่ในข้อมูลแต่ละกลุ่ม การจัดกลุ่มอาจจะทำด้วยมือโดยระบุกลุ่มหรือคลาสให้กับข้อมูลแต่ละตัว หรืออาจจะใช้วิธีอัตโนมัติโดยให้โปรแกรมจัดกลุ่มข้อมูลใช้เทคนิคการวัดความคล้ายคลึงกัน (similarity) ของข้อมูลแต่ละตัวเพื่อรวมกลุ่มข้อมูลที่คล้ายกันเข้าด้วยกัน ดังตัวอย่างในรูปที่ 1.1(a) ข้อมูล 43 ตัวแสดงด้วยสัญลักษณ์ x ในพิกัด X และ Y เมื่อวิเคราะห์ความคล้ายคลึงกันแล้วสามารถจัดเป็นกลุ่มได้ 7 กลุ่ม แต่ละกลุ่มแทนด้วยตัวเลข 1 ถึง 7 ดังแสดงในรูปที่ 1.1(b)



รูปที่ 1.1 การจัดกลุ่มข้อมูลอัตโนมัติ

การจัดกลุ่มข้อมูลโดยอัตโนมัติ เป็นงานวิจัยที่ได้รับความสนใจอย่างต่อเนื่องมาเป็นเวลานาน (Anderberg 1973; Jain and Dubes 1988; Michaud 1997; Jain,Murty and Flynn 1999; Han and Kamber 2001) เทคนิคที่นิยมใช้กันอย่างแพร่หลายคือ เทคนิคการแบ่ง หรือ partitioning โดยเริ่มใช้ในอัลกอริทึม k-means (MacQueen 1967) อัลกอริทึมนี้จะเริ่มต้นด้วยการแบ่งข้อมูล

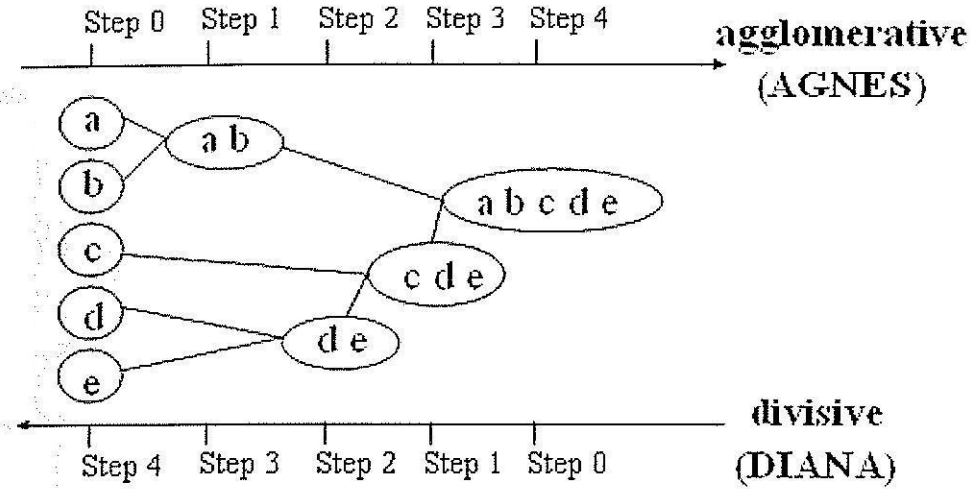
ออกเป็น k กลุ่ม จากนั้นคำนวณค่ากึ่งกลาง(ค่า mean) ของแต่ละกลุ่ม ข้อมูลจะถูกจัดเข้ากลุ่มที่อยู่ใกล้ที่สุด กระบวนการจะดำเนินไปจนกระทั่งข้อมูลทั้งหมดถูกจัดเข้ากลุ่ม แสดงตัวอย่างได้ดังรูปที่ 1.2



รูปที่ 1.2 แสดงขั้นตอนการทำงานของอัลกอริทึม k-means

ข้อด้อยของอัลกอริทึม k-means คือจะต้องมีการกำหนดค่า k (เป็นค่าที่ระบุจำนวนกลุ่ม) ล่วงหน้า และใช้วิธีคำนวณค่า mean เพื่อจัดกลุ่ม ทำให้ไม่สามารถใช้ได้กับกรณีข้อมูลเป็นข้อความหรือสัญลักษณ์ที่ไม่สามารถคำนวณค่า mean ได้ จึงมีผู้ปรับปรุงอัลกอริทึมเกิดเป็น k-modes (Huang 1998) และ CLARANS (Ng and Han 1994) Ester, Kreigel และ Xu (1995) ได้เสนอปรับปรุง CLARANS ด้วยโครงสร้างข้อมูล R*-tree และใช้เทคนิค focusing

อีกเทคนิคที่นิยมใช้ในการจัดกลุ่มข้อมูลอัตโนมัติคือ ใช้การจัดโครงสร้างเป็นลำดับชั้น (hierarchical clustering) โดยเริ่มใช้ในอัลกอริทึม AGNES และ DIANA (Kaufman and Rousseeuw 1990) ขั้นตอนการทำงานของอัลกอริทึมแสดงเป็นภาพได้ดังรูปที่ 1.3



รูปที่ 1.3 แสดงวิธีการจัดกลุ่มข้อมูลของอัลกอริทึม AGNES และ DIANA

Zhang, Ramakrishnan และ Livny (1996) ได้เสนอแนวทางการปรับปรุงอัลกอริทึม AGNES และ DIANA ด้วยการแบ่งการทำงานออกเป็นสองช่วง ช่วงแรกใช้การอ่านข้อมูลทั้งหมดเพื่อสร้าง clustering feature tree หรือ CF tree ซึ่งจะทำให้แทนข้อมูลทั้งหมดด้วยรูปแบบที่ย่อขึ้นพร้อมๆกับจัดกลุ่มข้อมูลออกเป็นหลายระดับ จากนั้นในช่วงที่สองใช้การประยุกต์อัลกอริทึมอื่นๆ เช่น k-means ในการจัดกลุ่มข้อมูลที่อยู่ในระดับโหนดใบของ CF tree

Guha, Rastogu และ Shim(1998) ได้พัฒนาวิธีการ hierarchical clustering ให้สมบูรณ์ขึ้นด้วยอัลกอริทึม CURE ซึ่งมีขั้นตอนการทำงานคือ

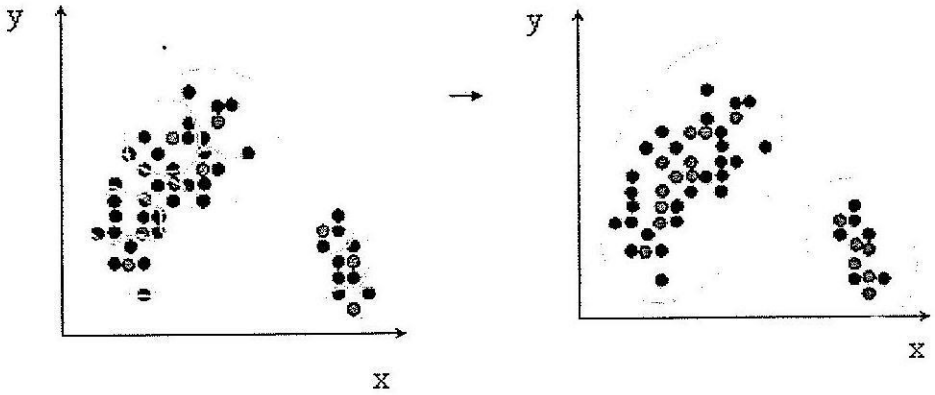
(1) สุ่มข้อมูลจากข้อมูลเริ่มต้นด้วยวิธีการ random sampling เพื่อให้ข้อมูลมีจำนวนลดลง

(2) นำข้อมูลที่สุ่มได้แบ่งออกเป็นส่วนๆ และจัดกลุ่มข้อมูลในแต่ละส่วน

(3) ในแต่ละกลุ่มข้อมูล หาข้อมูลที่เป็นตัวแทนกลุ่ม

(4) ข้อมูลที่เป็นตัวแทนกลุ่มสามารถถูกยุบรวมหรือโยกย้ายกลุ่มเพื่อให้ได้กลุ่มที่สมบูรณ์ขึ้น

ขั้นตอนสุดท้ายที่มีการยุบหรือโยกย้ายกลุ่มทำให้อัลกอริทึม CURE มีข้อได้เปรียบในการสร้างกลุ่มที่มีเส้นขอบเขตได้หลายรูปแบบ เช่น วงรี แทนที่จะจำกัดเฉพาะขอบเขตทรงกลมเหมือนอัลกอริทึมอื่นๆ และสามารถทำงานกับข้อมูลรบกวน (noise) ได้ ขั้นตอนการทำงานของอัลกอริทึมแสดงได้ดังรูปที่ 1.4



รูปที่ 1.4 แสดงการยุบรวมกลุ่มข้อมูลในอัลกอริทึม CURE

ในกรณีที่ข้อมูลมีจำนวนมากเกินกว่าจะบรรจุข้อมูลทั้งหมดในหน่วยความจำหลักได้ เทคนิคการจัดกลุ่มข้อมูลแบบเพิ่มพูน (incremental data clustering) สามารถช่วยให้อัลกอริทึมจัดกลุ่มข้อมูลปริมาณมากได้ โดยมีขั้นตอนการทำงานคือ

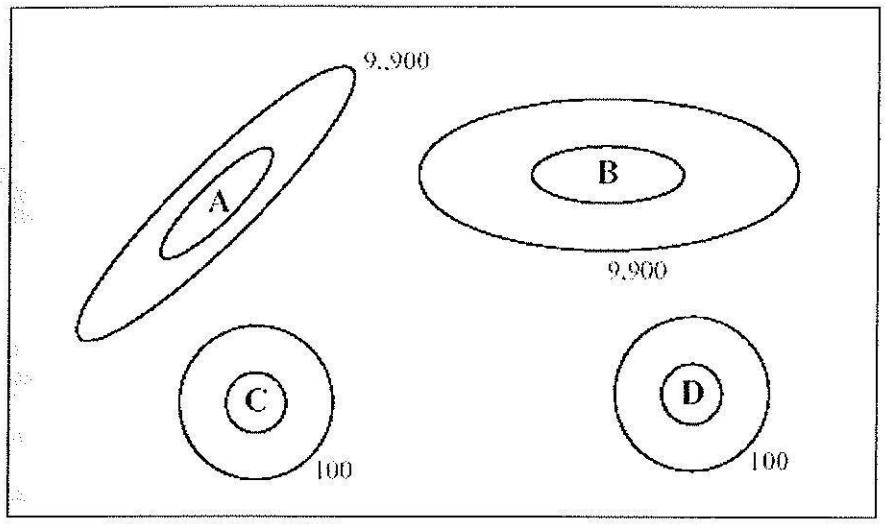
(1) อ่านข้อมูลเริ่มต้นคราวละตัว แล้วใช้เกณฑ์ที่ตั้งไว้ (เช่น ระยะห่างระหว่างข้อมูลกับจุดศูนย์กลางของกลุ่ม) จัดข้อมูลเข้ากลุ่ม

(2) อ่านข้อมูลตัวต่อไป แล้วพิจารณาจัดข้อมูลเข้ากลุ่ม อาจจะเป็นกลุ่มเดิมที่มีอยู่หรือกลุ่มใหม่ที่ใกล้เคียงกว่า

(3) ทำซ้ำในขั้นตอน 2 จนกว่าจะหมดข้อมูล

ข้อได้เปรียบของการทำ incremental คือ ใช้เนื้อที่หน่วยความจำน้อย อัลกอริทึมที่ใช้แนวทางนี้ได้แก่ leader (Hartigan 1975), ART network (Carpenter and Grossberg 1990), shortest spanning path (Slagle et el. 1975), cobweb (Fisher 1987)

เทคนิคการจัดกลุ่มข้อมูลแบบเพิ่มพูนในแบบดั้งเดิม จะพิจารณาข้อมูลทีละตัวเพื่อจัดเข้ากลุ่มและพิจารณาข้อมูลทุกตัว ซึ่งจะไม่เหมาะสมในงานทำเหมืองข้อมูลที่ข้อมูลมีปริมาณมาก ผลกระทบโดยตรงของข้อมูลจำนวนมาก คือ จำนวนกลุ่มที่ได้อาจจะมากจนไม่สามารถบรรจุไว้ในหน่วยความจำหลักได้ เพื่อเพิ่มประสิทธิภาพด้านเนื้อที่และเพิ่มความเร็วในการทำงานของอัลกอริทึม โครงการวิจัยนี้จึงเสนอแนวทางการจัดกลุ่มข้อมูลแบบเพิ่มพูนที่พิจารณาข้อมูลที่เป็นตัวแทน แทนที่จะพิจารณาข้อมูลทุกตัว (แนวคิดเรื่องข้อมูลตัวแทนนี้เสนอไว้ในงานของ Bradley และ คณะ ในปี 1998) ข้อมูลที่เป็นตัวแทนนี้จะได้มาจากการสุ่ม แต่แทนที่จะใช้การสุ่มแบบ random เหมือนในอัลกอริทึม CURE โครงการนี้จะพิจารณาการสุ่มตามความหนาแน่น (density-biased sampling) ซึ่งได้รับการพิสูจน์โดย Palmer และ Faloutsos (2000) ว่าเหมาะสมกว่าในกรณีข้อมูลมีการกระจายตัวไม่สม่ำเสมอดังแสดงในรูปที่ 1.5



รูปที่ 1.5 ข้อมูลในสี่กลุ่มที่มีความหนาแน่นแตกต่างกัน

จากรูปที่ 1.5 ข้อมูลในกลุ่ม A และ B มีจำนวนข้อมูลในกลุ่ม 9900 ตัว กลุ่ม C และ D มีข้อมูล 100 ตัว ข้อมูลรวมทั้งหมดมี 20,000 ตัว ถ้าต้องการสุ่มข้อมูลออกมาเพียง 1% จะได้ข้อมูล 200 ตัว ถ้าเป็นการสุ่มตามปกติจะได้ข้อมูลจากกลุ่ม A และ B กลุ่มละ 99 ตัว และจากกลุ่ม C และ D เพียงกลุ่มละ 1 ตัว ซึ่งถ้าส่งข้อมูลทั้ง 200 ตัวนี้เข้าสู่อัลกอริทึม clustering ข้อมูล 1 ตัวจากกลุ่ม C และ 1 ตัวจากกลุ่ม D จะถูกพิจารณาว่าเป็นข้อมูลรบกวนและจะถูกละเลยไม่นำมาใช้ แนวทางที่เหมาะสมในการจัดการกับกรณีเช่นนี้ คือ ให้น้ำหนักกับข้อมูลที่สุ่มได้ไม่เท่ากัน เพื่อป้องกันไม่ให้ข้อมูลจากกลุ่มเล็กถูกละเลย การสุ่มโดยให้น้ำหนักแบบนี้เรียกว่า การสุ่มตามความหนาแน่น (density-biased sampling)

งานวิจัยนี้จะพิจารณาจัดกลุ่มข้อมูลตามความหนาแน่น แต่ในขั้นตอนของการจัดกลุ่มจะไม่ใช้การสุ่มเพื่อลดขนาดของข้อมูล ถ้าหากข้อมูลเริ่มต้นมีขนาดใหญ่เกินไปผู้ใช้สามารถใช้การคัดเลือกเฉพาะข้อมูลที่เป็นตัวแทนกลุ่มและใช้ข้อมูลตัวแทนนี้ในการทำ incremental clustering ได้ และในงานวิจัยนี้จะพัฒนาเกณฑ์ขึ้นใช้ในการผนวกกลุ่มย่อยจากการทำ incremental clustering เข้าด้วยกันได้

วัตถุประสงค์ของการวิจัย

เพื่อออกแบบและพัฒนาแนวทางการจัดกลุ่มข้อมูลในลักษณะเพิ่มพูน (incremental) ให้สามารถทำงานกับข้อมูลจำนวนมากในเวลาอันรวดเร็ว และพัฒนาเทคนิคการจัดกลุ่มข้อมูลตามความหนาแน่นให้สามารถประยุกต์ใช้กับกระบวนการจัดกลุ่มแบบเพิ่มพูนได้อย่างมีประสิทธิภาพ

ขอบเขตของการวิจัย

งานวิจัยนี้มีจุดมุ่งหมายที่จะพัฒนาแนวทางและเทคนิคของการทำ incremental clustering การทำ clustering จะเน้นเฉพาะเทคนิคในกลุ่ม partitioning ข้อมูลที่ใช้ในการทดสอบจะเป็นข้อมูลจากฐานข้อมูลมาตรฐาน (UCI machine learning repository) โดยจะสามารถจัดกลุ่มข้อมูลที่เป็นค่า nominal หรือ categorical ได้

การพัฒนาโปรแกรมใช้ภาษา Prolog ซึ่งเป็นภาษาเชิงตรรกะ เนื่องจากมีแนวคิดและรูปแบบที่เหมาะสมสำหรับงานที่จะพัฒนาเป็นฐานความรู้ต่อไปในอนาคต ภาษา Prolog ที่ใช้ในงานวิจัยนี้ใช้มาตรฐานของ SWI Prolog (www.swi-prolog.org) ซึ่งเป็นซอฟต์แวร์ประเภทโอเพนซอร์ส (open-source software) ทำให้ผู้ใช้สามารถนำไปใช้งานหรือนำไปพัฒนาต่อได้โดยไม่มีปัญหาเรื่องลิขสิทธิ์ซอฟต์แวร์

ประโยชน์ที่ได้รับจากการวิจัย

อัลกอริทึมและซอฟต์แวร์ที่พัฒนาขึ้นนี้มีจุดมุ่งหมายเพื่อใช้ประโยชน์ในงานวิเคราะห์ข้อมูลอัตโนมัติประเภทการจัดกลุ่มข้อมูล สามารถใช้งานได้จริงกับข้อมูลที่รวบรวมจากงานประเภทต่างๆ ดังนั้นซอฟต์แวร์ที่พัฒนาขึ้นจึงใช้ประโยชน์ได้กับทุกวงการที่เกี่ยวข้องกับงานจัดกลุ่มข้อมูลอัตโนมัติ ซอฟต์แวร์นี้ผู้วิจัยจะเผยแพร่เป็นสาธารณะ ทุกหน่วยงานสามารถนำไปใช้ได้ คาดว่าจะช่วยให้ประเทศชาติประหยัดเงินตราซื้อซอฟต์แวร์สำเร็จรูปเพื่อการจัดกลุ่มข้อมูลอัตโนมัติลงได้จำนวนมาก นอกจากนี้องค์ความรู้ใหม่เกี่ยวกับการทำ incremental clustering และเทคนิคในการทำ density-biased clustering ยังสามารถเผยแพร่ในวงวิชาการได้

การเผยแพร่ซอฟต์แวร์และคู่มือการใช้งานจะกระทำผ่านอินเตอร์เน็ต โดยให้ผู้ใช้และผู้ที่สนใจดาวน์โหลดได้จากเว็บไซต์ของสาขาวิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในส่วนของเทคนิคและอัลกอริทึมต่างๆ ที่ได้รับการคิดค้นพัฒนาขึ้นเป็นองค์ความรู้ใหม่จะเผยแพร่ด้วยการเขียนบทความนำเสนอและตีพิมพ์ในวารสารการประชุมวิชาการระดับประเทศและระดับนานาชาติ

บทที่ 2

วิธีดำเนินการวิจัย

กรอบแนวคิดของงานวิจัย

ขั้นตอนการทำงานของโปรแกรมจัดกลุ่มข้อมูลอัตโนมัติ โดยทั่วไปจะประกอบด้วย 5 ขั้นตอนหลัก คือ

(1) กำหนดจำนวนกลุ่มและกำหนดลักษณะที่จะใช้จัดข้อมูลเข้ากลุ่ม งานที่ต้องทำประกอบด้วย pattern representation, feature selection, feature extraction

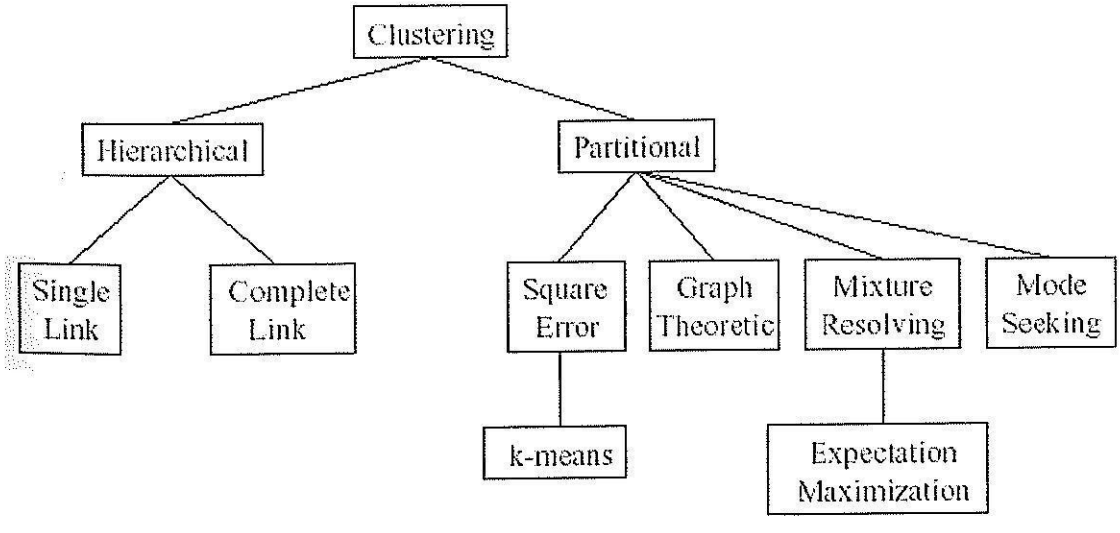
(2) กำหนดฟังก์ชันที่จะใช้วัดความคล้ายคลึงของข้อมูล งานที่ต้องทำประกอบด้วย pattern similarity measure

(3) จัดกลุ่มข้อมูล งานที่ต้องทำประกอบด้วย grouping

(4) กำหนดรูปแบบการแสดงกลุ่ม งานที่ต้องทำประกอบด้วย data abstraction

(5) วิเคราะห์ผลการจัดกลุ่ม งานที่ต้องทำประกอบด้วย cluster validation

การจัดกลุ่มข้อมูลอัตโนมัติ จะได้ผลลัพธ์ที่เป็นประโยชน์และมีประสิทธิภาพมากน้อยเพียงใดจะขึ้นอยู่กับเทคนิคที่ใช้ในขั้นตอนการจัดกลุ่มข้อมูล (grouping) ซึ่งโดยทั่วไปเทคนิค หรืออัลกอริทึมที่ใช้จะจัดอยู่ในสองกลุ่มใหญ่ คือ partitioning methods และ hierarchical methods ซึ่งในแต่ละกลุ่มยังสามารถแยกย่อยเป็นอีกหลายเทคนิค ดังแสดงด้วยแผนภาพในรูปที่ 2.1



รูปที่ 2.1 แสดงการจัดหมวดหมู่เทคนิคที่ใช้ในการทำ clustering

Partitioning methods

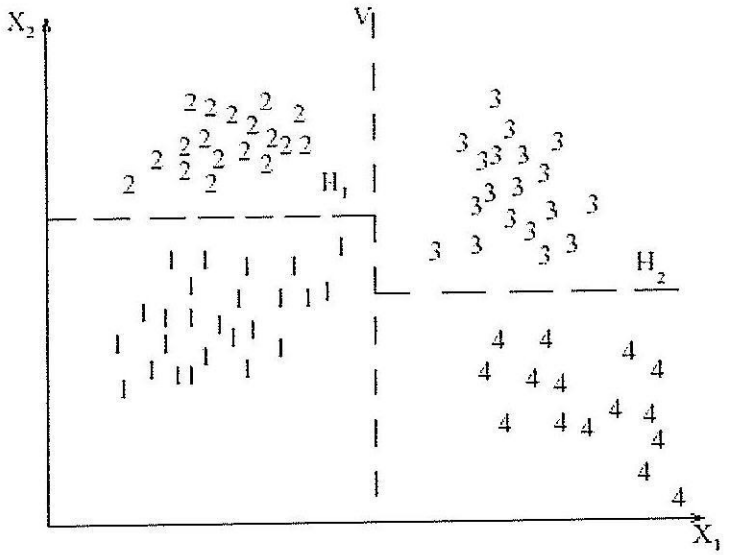
ข้อมูล n ตัวจะถูกแบ่งออกเป็น k กลุ่ม โดยที่ $k \leq n$ และมีข้อกำหนดว่าแต่ละกลุ่มจะต้องมีข้อมูลอยู่อย่างน้อยหนึ่งตัว และข้อมูลหนึ่งตัวจะต้องอยู่ในกลุ่มเดียวเท่านั้น (ข้อกำหนดประการหลังนี้จะไม่มีในกรณีของ fuzzy clustering ที่ข้อมูลสามารถถูกพิจารณาให้อยู่ในหลายกลุ่มได้) ตัวอย่างอัลกอริทึมในกลุ่มนี้ได้แก่ k-means, k-medoids, k-modes, k-prototypes, EM (Expectation Maximization), CLARANS ตัวอย่างการทำงานของ partitioning methods แสดงได้ดังรูปที่ 2.2 จากรูปแกน x_1 และ x_2 คือ feature หรือคุณลักษณะของข้อมูลที่ใช้ในการพิจารณาจัดกลุ่ม การจัดกลุ่มครั้งแรกพิจารณาที่ feature x_1 ทำให้ได้เส้น V ที่ใช้แบ่งกลุ่มข้อมูลส่วนแรก จากนั้นใช้ feature x_2 พิจารณาแบ่งกลุ่มต่อไปทำให้ได้เส้นแบ่งกลุ่มในแนวนอน คือ H_1 และ H_2 จากตัวอย่างนี้จะจัดข้อมูลได้ 4 กลุ่ม และลักษณะข้อมูลในแต่ละกลุ่มคือ

Cluster 1 : $[x_1 < V]$ and $[x_2 < H_1]$

Cluster 2 : $[x_1 < V]$ and $[x_2 > H_1]$

Cluster 3 : $[x_1 > V]$ and $[x_2 > H_2]$

Cluster 4 : $[x_1 > V]$ and $[x_2 < H_2]$

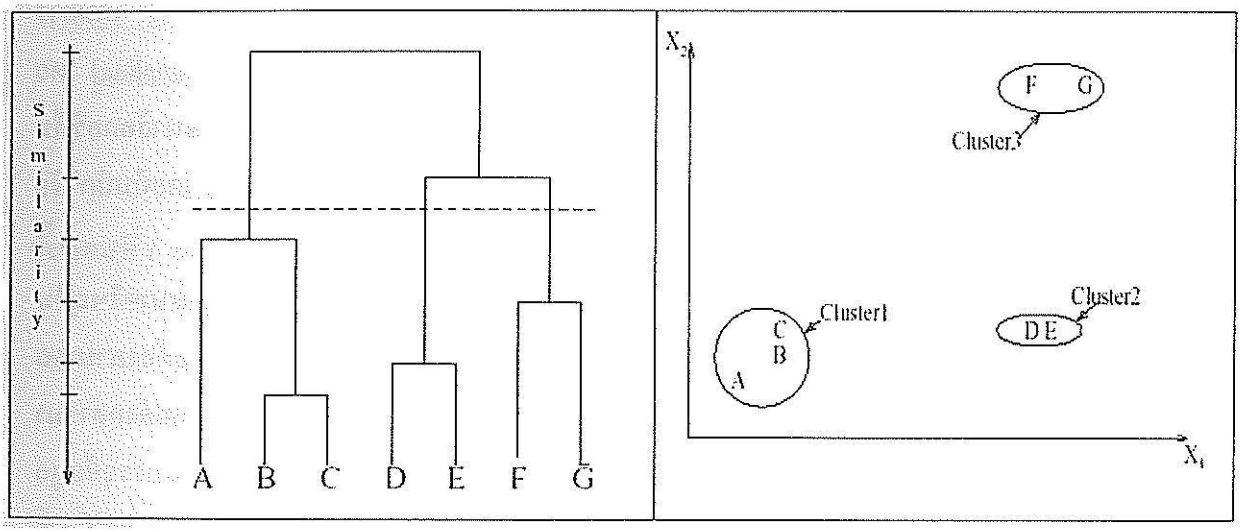


รูปที่ 2.2 การจัดกลุ่มข้อมูลด้วยเทคนิค partitioning

อัลกอริทึมในกลุ่ม partitioning จะใช้ได้ดีกับกรณีข้อมูลจำนวนน้อยและมี feature ไม่มาก ถ้าข้อมูลมี feature มากจะทำให้ได้จำนวนกลุ่มที่มากเกินไป นอกจากนี้เกณฑ์ในการจัดกลุ่ม เช่น squared error จะทำงานได้ดีกับข้อมูลที่รวมกลุ่มหนาแน่น

Hierarchical methods

เทคนิคในกลุ่มนี้แบ่งย่อยออกเป็น agglomerative และ divisive เทคนิค agglomerative จะเริ่มทำงานด้วยการจัดข้อมูลหนึ่งตัวเป็นหนึ่งกลุ่ม จากนั้นพิจารณารวมกลุ่มข้อมูลที่คล้ายคลึงกันเข้าด้วยกัน ทำในลักษณะนี้ไปจนกระทั่งไม่สามารถรวมกลุ่มข้อมูลต่อไปได้ เทคนิค divisive มีหลักการทำงานแบบเดียวกันแต่ทำในทิศทางตรงกันข้าม โดยเริ่มจากข้อมูลทั้งหมดจัดเป็นกลุ่มเดียวแล้วกระจายกลุ่มที่แตกต่างกันออกตามลำดับจนกระทั่งไม่สามารถกระจายต่อไปได้ ภาพในรูปที่ 2.3 แสดงการจัดข้อมูล 7 ตัว (ได้แก่ข้อมูล A, B, C, D, E, F และ G) เข้าเป็น 3 กลุ่มด้วยเทคนิค hierarchical



รูปที่ 2.3 การจัดกลุ่มข้อมูลด้วยเทคนิค hierarchical

อัลกอริทึมในกลุ่ม hierarchical ประกอบด้วย CURE, Chamelon, BIRCH ข้อดีของอัลกอริทึมในกลุ่มนี้คือ ทำงานกับข้อมูลที่จัดกลุ่มได้หลากหลายลักษณะกว่าวิธีการ partitioning แต่ข้อเสียคือใช้เวลาในการทำงานนานกว่าและใช้เนื้อที่หน่วยความจำมากกว่า

อัลกอริทึมในการจัดกลุ่มข้อมูลทั้งในแบบ partitioning และ hierarchical ทำงานได้ดีกับข้อมูลขนาดเล็ก แต่เมื่อข้อมูลมีจำนวนมากขึ้นอัลกอริทึมจะเริ่มด้อยประสิทธิภาพ ดังแสดงด้วยผลการวิเคราะห์เปรียบเทียบเวลาและหน่วยความจำที่ใช้สำหรับแต่ละอัลกอริทึม ในตารางที่ 2.1 โดย n คือ จำนวนข้อมูล k คือ จำนวนกลุ่มของข้อมูล และ l คือ จำนวนรอบที่อัลกอริทึมใช้ในการทำงาน

ตารางที่ 2.1 เปรียบเทียบเวลาและหน่วยความจำที่ใช้ในแต่ละอัลกอริทึม

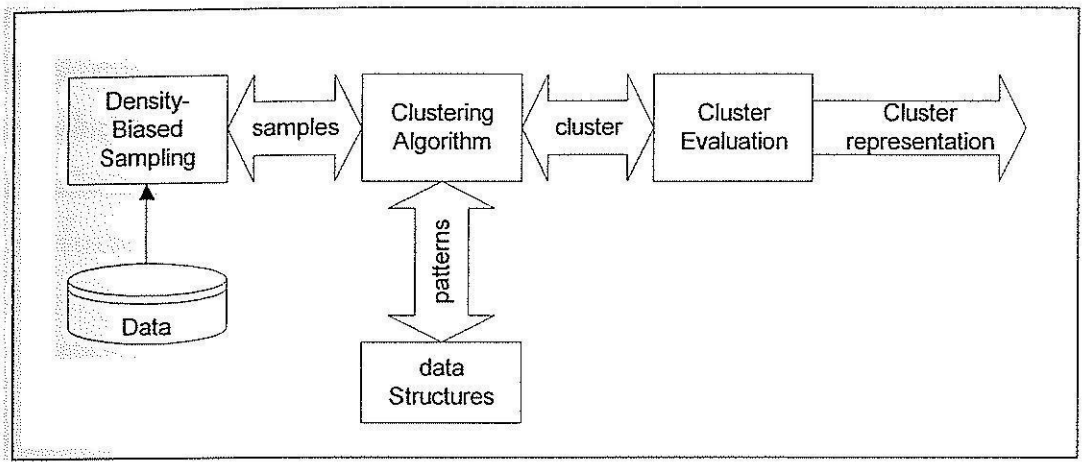
Clustering Algorithm	Time Complexity	Space Complexity
leader	$O(kn)$	$O(k)$
k-means	$O(nkl)$	$O(k)$
ISODATA	$O(nkl)$	$O(k)$
shortest spanning path	$O(n^2)$	$O(n)$
single-line	$O(n^2 \log n)$	$O(n^2)$
complete-line	$O(n^2 \log n)$	$O(n^2)$

ในกรณีของการจัดกลุ่มในงานทำเหมืองข้อมูลซึ่งมีปริมาณข้อมูลจำนวนมาก ได้มีการพัฒนาอัลกอริทึมที่สามารถรองรับข้อมูลขนาดใหญ่ได้ ในกลุ่มที่ใช้เทคนิค partitioning ได้มีการพัฒนาอัลกอริทึม CLARANS (Clustering Large Applications based on RANdom Sampling) โดยทีมนักวิจัย R.Ng และ J.Han ในปี 1994 ในกลุ่มเทคนิค hierarchical ได้มีการพัฒนาอัลกอริทึม BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) โดยทีมนักวิจัย T.Zhang,R. Ramakrishnan และ M.Livny ในปี 1996

ถึงแม้อัลกอริทึม CLARANS และ BIRCH จะสามารถทำงานกับข้อมูลขนาดใหญ่ในเวลาที่เร็วขึ้นได้ (O(n)) แต่ข้อจำกัดที่สำคัญของอัลกอริทึมคือ ข้อมูลเริ่มต้นทั้งหมดจะต้องสามารถบรรจุอยู่ในหน่วยความจำหลักได้ ข้อจำกัดนี้เป็นอุปสรรคที่สำคัญในการนำอัลกอริทึมดังกล่าวมาใช้ในงานทำเหมืองข้อมูล เนื่องจากข้อมูลที่เกิดขึ้นจริงมักจะมีปริมาณมากเกินกว่าจะบรรจุข้อมูลทั้งหมดไว้ในหน่วยความจำหลักได้ แนวทางจัดการกับปัญหานี้เป็นได้ 2 แนวทาง คือ (1) ใช้วิธีการประมวลผลแบบขนาน (parallel clustering) หรือ (2) ใช้วิธีนำข้อมูลเพิ่มเข้ามาทีละส่วนเพื่อจัดกลุ่ม หรือเรียกว่าการจัดกลุ่มแบบเพิ่มพูน (incremental clustering)

โครงการวิจัยนี้จะมุ่งเน้นไปที่แนวทาง incremental clustering เนื่องจากนำไปใช้ประโยชน์ได้กว้างขวาง และมีข้อกำหนดน้อยกว่าแนวทางการประมวลผลแบบขนาน โดยจะพิจารณานำเทคนิคการสุ่มมาใช้ช่วยเพื่อใช้ข้อมูลที่เป็นตัวแทน แทนที่จะใช้ข้อมูลทุกตัวซึ่งคาดว่าจะช่วยเพิ่มความเร็วในการทำงานของอัลกอริทึม โดยที่การสุ่มจะพิจารณาให้น้ำหนักตามความหนาแน่นของข้อมูล (density-biased sampling) แทนที่จะเป็นแบบ random sampling ทั้งนี้เพื่อป้องกันข้อมูลกลุ่มเล็กถูกลดความสำคัญหรือถูกกำจัดไปในระหว่างการจัดกลุ่มเนื่องจากความเข้าใจผิดว่าเป็นข้อมูลรบกวน (noise)

แนวทางการวิจัยที่เสนอขึ้นนี้ จะประกอบด้วยการออกแบบและพัฒนาโครงสร้างของส่วนประกอบหลักสองส่วน (ดังแสดงในรูปที่ 2.4) ที่จะต้องใช้ในการทำ incremental clustering ได้แก่ density-biased sampling technique และ incremental clustering algorithm



รูปที่ 2.4 กรอบแนวคิดของงานออกแบบและพัฒนาการจัดกลุ่มข้อมูลตามความหนาแน่น

งานวิจัยตลอดทั้งโครงการประกอบด้วยขั้นตอนหลัก ดังต่อไปนี้

ขั้นตอนที่ 1 *ออกแบบและพัฒนาเทคนิคการสุ่มตามความหนาแน่น*

ขั้นตอนการทำงานในส่วนนี้จะประกอบด้วย การกำหนดเกณฑ์การวิเคราะห์ความหนาแน่นของข้อมูล การกำหนดฟังก์ชันเพื่อรักษาคุณสมบัติความหนาแน่นของข้อมูล และการกำหนดน้ำหนักที่ควรจะให้กับข้อมูลแต่ละตัว จากนั้นจะเป็นการออกแบบอัลกอริทึม density-biased sampling

ขั้นตอนที่ 2 *วิเคราะห์และออกแบบโครงสร้างข้อมูลที่ใช้ช่วยในการทำ clustering*

โครงสร้างข้อมูลที่นิยมใช้ช่วยในกระบวนการทำ clustering มีได้หลากหลาย เช่น ใช้โครงสร้างแฮช ใช้โครงสร้างต้นไม้สมดุล งานในขั้นตอนนี้จึงเป็นการศึกษาแนวทางที่มีผู้เสนอไว้เพื่อวิเคราะห์จุดเด่น-จุดด้อย ซึ่งจะเป็นประโยชน์ในการออกแบบโครงสร้างข้อมูลของอัลกอริทึม incremental clustering โครงสร้างข้อมูลที่มีประสิทธิภาพจะมีผลอย่างมากต่อความเร็วและความสามารถของอัลกอริทึม

ขั้นตอนที่ 3 *ออกแบบอัลกอริทึมในการทำ incremental clustering*

ในขั้นตอนนี้จะเป็นการกำหนดมาตรวัดที่จะใช้จัดข้อมูลเป็นกลุ่ม หรือ คลัสเตอร์ รวมทั้งกำหนดเกณฑ์ที่จะใช้พิจารณาปรับปรุงคลัสเตอร์ที่ได้ให้เหมาะสมยิ่งขึ้น จากนั้นกำหนดค่าที่จะใช้ในการปรับจุดกึ่งกลางของคลัสเตอร์ เทคนิคของการสุ่มตามความหนาแน่นจะถูกนำมาใช้ร่วมกับการออกแบบอัลกอริทึม incremental clustering โดยพยายามลดจำนวนครั้งของการ scan ข้อมูลในฐานข้อมูลเพื่อเพิ่มความเร็วของอัลกอริทึม

การออกแบบอัลกอริทึมเพื่อจัดกลุ่มข้อมูลตามความหนาแน่น

การทำงานของอัลกอริทึมจัดกลุ่มข้อมูลตามความหนาแน่น จะประกอบด้วยการทำงานสองขั้นตอนใหญ่คือ การวัดความหนาแน่นของข้อมูลเพื่อคัดเลือกข้อมูลเป็นตัวแทนในการจัดกลุ่ม และการจัดข้อมูลตัวแทนเข้ากลุ่มโดยใช้ค่า similarity ของแอททริบิวต์เป็นเกณฑ์ในการจัดเข้ากลุ่ม

Algorithm 1 Density-biased clustering

Input: a data file,
minimum number of matched attributes M,
minimum density D

Output: a set of clusters with cluster means information

Phase 1 Selecting samples from dense data

(1) Show the GUI of density-biased clustering component

(2) Get the user's response to obtain the data file name,
minimum number of matched attributes M, density threshold D, and
number of cluster K

/* Compute similar instances and their density values */

(3) Open data file and read data instance

(4) For each data instance do

(4.1) Scan data file to collect instances, Ins, with matched attributes ≥ M

(4.2) Compute density, Den, as proportion of Ins to total instances in data file

(4.3) If Den ≥ D, then record this instance in temporary file F

Phase 2 Grouping data from the dense area

(5) Taking the first K data instances in F as temporary cluster means

(6) Repeat

(6.1) Assign each data instance in F into closest cluster, based on similarity

(6.2) Compute new cluster means

(6.3) Until each data instance does not change its cluster

(7) Return the cluster means and cluster members

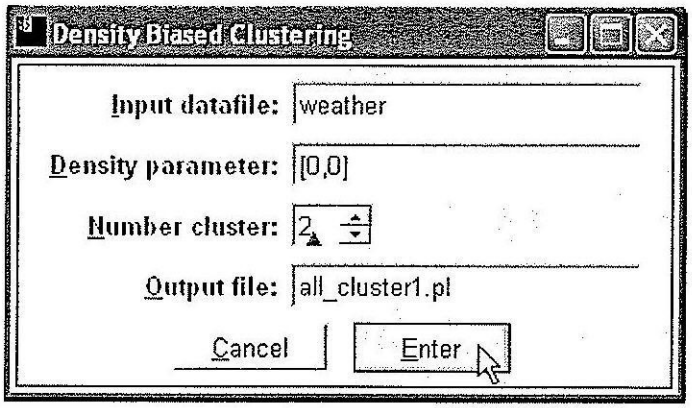
ในช่วงของการคัดเลือกข้อมูลตามความหนาแน่น จะให้ผู้ใช้กำหนดเกณฑ์ขั้นต่ำว่าจะพิจารณาความหนาแน่นด้วยค่าในกี่แอททริบิวต์ (M, Minimum number of attributes) และจะต้องการสุ่มข้อมูลที่มีความหนาแน่นขั้นต่ำเท่าไร (D, Density) โดยที่ $D \in [0..1]$ จากนั้นเริ่มต้นทำงานด้วยการเปิดไฟล์และอ่านข้อมูลทีละรายการเพื่อตรวจสอบข้อมูลที่อยู่ใกล้เคียง การวัดความใกล้เคียงใช้การเปรียบเทียบค่าในแต่ละแอททริบิวต์ ถ้ามีค่าคล้ายกันอย่างน้อย M แอททริบิวต์ถือว่า

เป็นข้อมูลที่เกาะกลุ่มอยู่ใกล้กัน ตรวจสอบข้อมูลใกล้เคียงเช่นนี้กับข้อมูลทุกรายการ จากนั้นนับจำนวนว่ามีข้อมูลกี่รายการที่จัดว่าอยู่ใกล้เคียง คำนวณค่าข้อมูลใกล้เคียงให้เป็นค่าสัดส่วนโดยหารด้วยจำนวนข้อมูลทั้งหมดในไฟล์ ทำการตรวจสอบเช่นนี้กับข้อมูลทุกรายการ จากนั้นคัดเลือกไว้เฉพาะข้อมูลที่มีค่า D ถึงเกณฑ์ที่ระบุ แล้วนำข้อมูลที่คัดเลือกไว้จัดกลุ่มเป็น K กลุ่ม โดยค่า K จะต้องมีค่าไม่มากกว่าจำนวนข้อมูลที่คัดเลือกไว้ ข้อมูลในรูปที่ 2.5 เป็นข้อมูลที่ใช้เป็นตัวอย่างเพื่อแสดงขั้นตอนการทำงานของโปรแกรมจัดกลุ่มข้อมูล

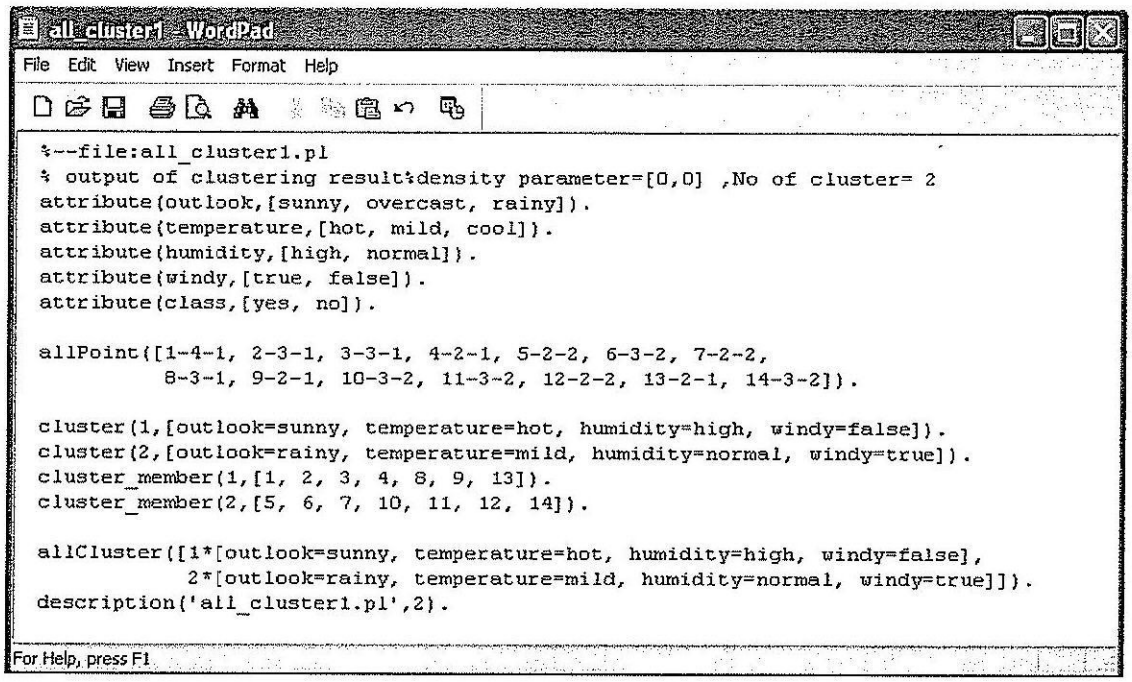
```
%% Data weather
%
%attributes: names and their possible values
%
attribute(outlook,      [sunny, overcast, rainy] ).
attribute(temperature, [hot, mild, cool]         ).
attribute(humidity,     [high, normal]           ).
attribute(windy,        [true, false]            ).
attribute(class,        [yes, no]                ).
%data
instance(1, class=no, [outlook=sunny, temperature=hot, humidity=high, windy=false]).
instance(2, class=no, [outlook=sunny, temperature=hot, humidity=high, windy=true]).
instance(3, class=yes, [outlook=overcast, temperature=hot, humidity=high, windy=false]).
instance(4, class=yes, [outlook=rainy, temperature=mild, humidity=high, windy=false]).
instance(5, class=yes, [outlook=rainy, temperature=cool, humidity=normal, windy=false]).
instance(6, class=no, [outlook=rainy, temperature=cool, humidity=normal, windy=true]).
instance(7, class=yes, [outlook=overcast, temperature=cool, humidity=normal, windy=true]).
instance(8, class=no, [outlook=sunny, temperature=mild, humidity=high, windy=false]).
instance(9, class=yes, [outlook=sunny, temperature=cool, humidity=normal, windy=false]).
instance(10, class=yes, [outlook=rainy, temperature=mild, humidity=normal, windy=false]).
instance(11, class=yes, [outlook=sunny, temperature=mild, humidity=normal, windy=true]).
instance(12, class=yes, [outlook=overcast, temperature=mild, humidity=high, windy=true]).
instance(13, class=yes, [outlook=overcast, temperature=hot, humidity=normal, windy=false]).
instance(14, class=no, [outlook=rainy, temperature=mild, humidity=high, windy=true]).
%
```

รูปที่ 2.5 ตัวอย่างไฟล์ข้อมูลที่จะนำเข้ายังโปรแกรมจัดกลุ่มข้อมูลตามความหนาแน่น

โปรแกรมจัดกลุ่มข้อมูลตามความหนาแน่น เริ่มต้นทำงานโดยแสดงจอภาพดังรูปที่ 2.6 ให้ผู้ใช้ระบุชื่อแฟ้มข้อมูล จำนวนกลุ่ม (K) และพารามิเตอร์ที่ใช้ในการคำนวณความหนาแน่นของข้อมูล โดยพารามิเตอร์นี้จะประกอบด้วยจำนวนเลขสองจำนวน [M, D] ค่าM จะเป็นค่าจำนวนเต็มหมายถึงจำนวนแอททริบิวต์ของข้อมูลที่จะใช้คำนวณ similarity และค่า D จะเป็นค่าขั้นต่ำของความหนาแน่น (หมายถึง สัดส่วนข้อมูลที่อยู่ใกล้เคียงกับข้อมูลแต่ละเรคคอร์ด) โดย D จะมีค่าอยู่ระหว่าง 0.0 ถึง 1.0 การระบุค่า D เป็น 0 หรือ 0.0 หมายถึงให้จัดกลุ่มข้อมูลทุกเรคคอร์ดโดยไม่ต้องคำนึงถึงความหนาแน่นข้อมูล จากข้อมูลตัวอย่างในรูปข้างต้นเมื่อจัดกลุ่มโดยระบุความหนาแน่นเป็น 0 จะได้ผลลัพธ์ดังรูปที่ 2.7



รูปที่ 2.6 จอภาพเริ่มต้นของโปรแกรมจัดกลุ่มข้อมูลตามความหนาแน่น



รูปที่ 2.7 ผลลัพธ์ของการจัดกลุ่มข้อมูลเมื่อระบุค่าความหนาแน่นเป็นศูนย์

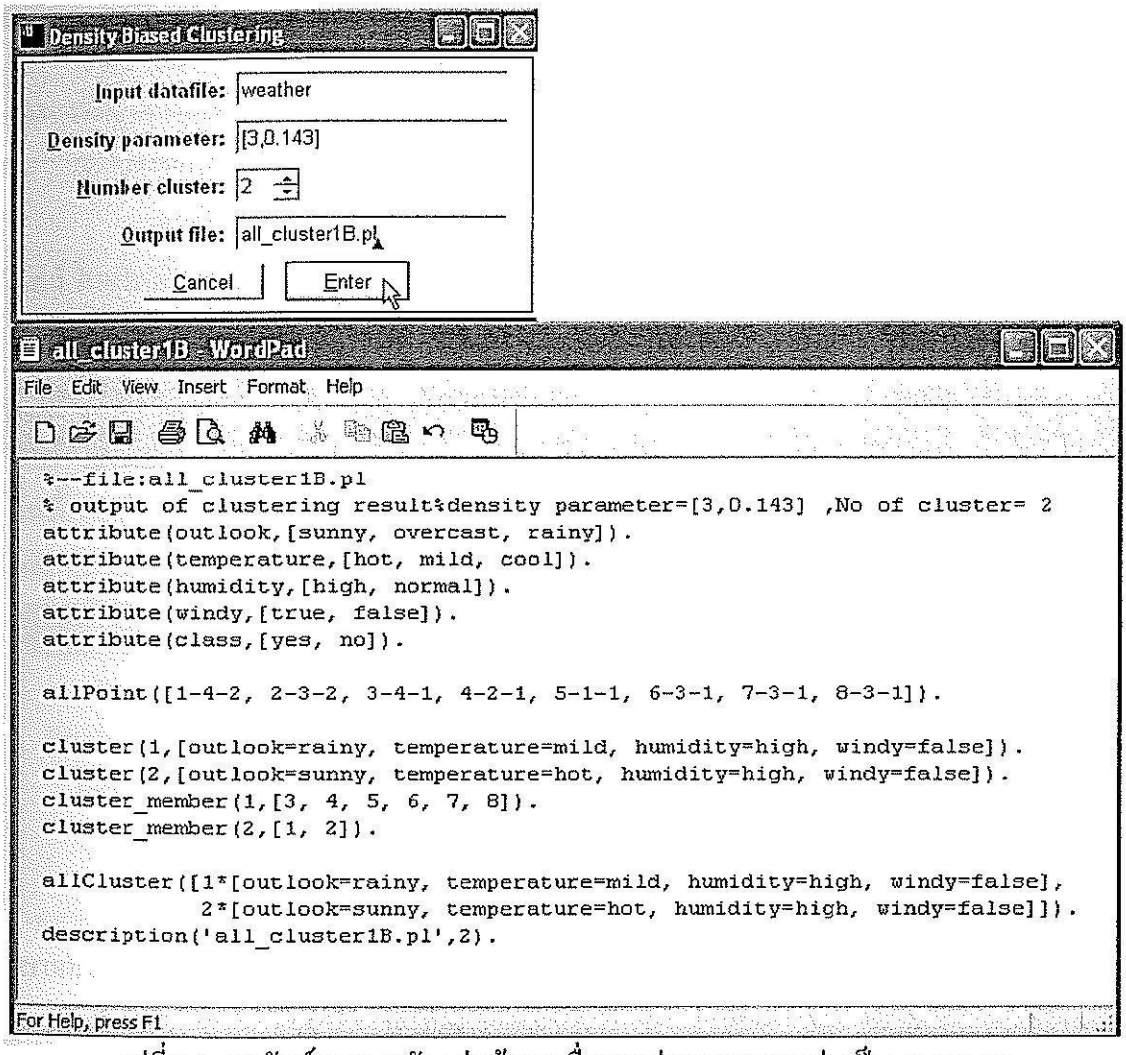
ผลลัพธ์ของการจัดกลุ่มเมื่อกำหนดพารามิเตอร์เป็น [0,0] และกำหนดให้จัดข้อมูลเป็นสองกลุ่ม ได้ลักษณะของกลุ่มเป็น

```
cluster(1,[outlook=sunny,temperature=hot,humidity=high,windy=false]).
cluster(2,[outlook=rainy,temperature=mild,humidity=normal,windy=true]).
```

และจัดข้อมูลทั้ง 14 เรคคอร์ดเข้ากลุ่มได้ดังนี้

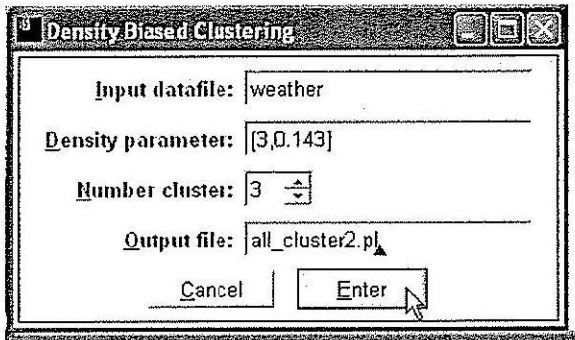
```
cluster_member(1,[1, 2, 3, 4, 8, 9, 13]).
cluster_member(2,[5, 6, 7, 10, 11, 12, 14]).
```

เมื่อกำหนดพารามิเตอร์ความหนาแน่นของข้อมูลเป็น [3, 0.143] ซึ่งหมายถึงวัดความใกล้เคียงของข้อมูลโดยใช้แอททริบิวต์อย่างต่ำ 3 แอททริบิวต์ และคัดเลือกข้อมูลที่มีข้อมูลอื่นอยู่ใกล้เคียงคิดเป็นสัดส่วนอย่างน้อย 0.143 หรือ 14.3% (นั่นคือ ข้อมูลที่ถูกคัดเลือกจะต้องมีข้อมูลอื่นอยู่ใกล้เคียงคิดเป็น 0.143 × 14 เรคคอร์ด = 2 เรคคอร์ด) ข้อมูลที่มีค่าความหนาแน่นตรงตามเกณฑ์ดังกล่าวมีจำนวน 8 เรคคอร์ด และผลลัพธ์ของการจัดกลุ่มตามเกณฑ์ดังกล่าวแสดงดังรูปที่ 2.8



รูปที่ 2.8 ผลลัพธ์ของการจัดกลุ่มข้อมูลเมื่อระบุค่าความหนาแน่นเป็น [2, 0.143]

เมื่อพิจารณาลักษณะของข้อมูลในทั้งสองกลุ่ม จะเห็นว่าค่าของแอททริบิวต์ humidity และwindy ของทั้งสองกลุ่มจะเหมือนกัน ส่วนที่แตกต่างกันจะมีเพียงค่าของแอททริบิวต์ outlook และ temperature และเมื่อจัดกลุ่มข้อมูลด้วยเกณฑ์ความหนาแน่น [3, 0.143] และเปลี่ยนจำนวนกลุ่มของข้อมูลจากสองกลุ่มเป็นสามกลุ่ม จะได้ผลลัพธ์ดังรูปที่ 2.9



all_cluster2 - WordPad

File Edit View Insert Format Help

```
%--file:all_cluster2.pl
% output of clustering result%density parameter=[3,0.143] ,No of cluster= 3
attribute(outlook,[sunny, overcast, rainy]).
attribute(temperature,[hot, mild, cool]).
attribute(humidity,[high, normal]).
attribute(windy,[true, false]).
attribute(class,[yes, no]).

allPoint([1-4-1, 2-4-2, 3-3-3, 4-3-3, 5-2-3, 6-3-1, 7-4-3, 8-2-3]).

cluster(1,[outlook=sunny, temperature=hot, humidity=high, windy=false]).
cluster(2,[outlook=overcast, temperature=hot, humidity=high, windy=false]).
cluster(3,[outlook=rainy, temperature=mild, humidity=normal, windy=false]).
cluster_member(1,[1, 6]).
cluster_member(2,[2]).
cluster_member(3,[3, 4, 5, 7, 8]).

allCluster([1*[outlook=sunny, temperature=hot, humidity=high, windy=false],
            2*[outlook=overcast, temperature=hot, humidity=high, windy=false],
            3*[outlook=rainy, temperature=mild, humidity=normal, windy=false]]).
description('all_cluster2.pl',3).
```

รูปที่ 2.9 ผลลัพธ์ของการจัดข้อมูลเป็นสามกลุ่มและระบุค่าความหนาแน่นเป็น [3, 0.143]

การจัดกลุ่มข้อมูลแบบเพิ่มพูน

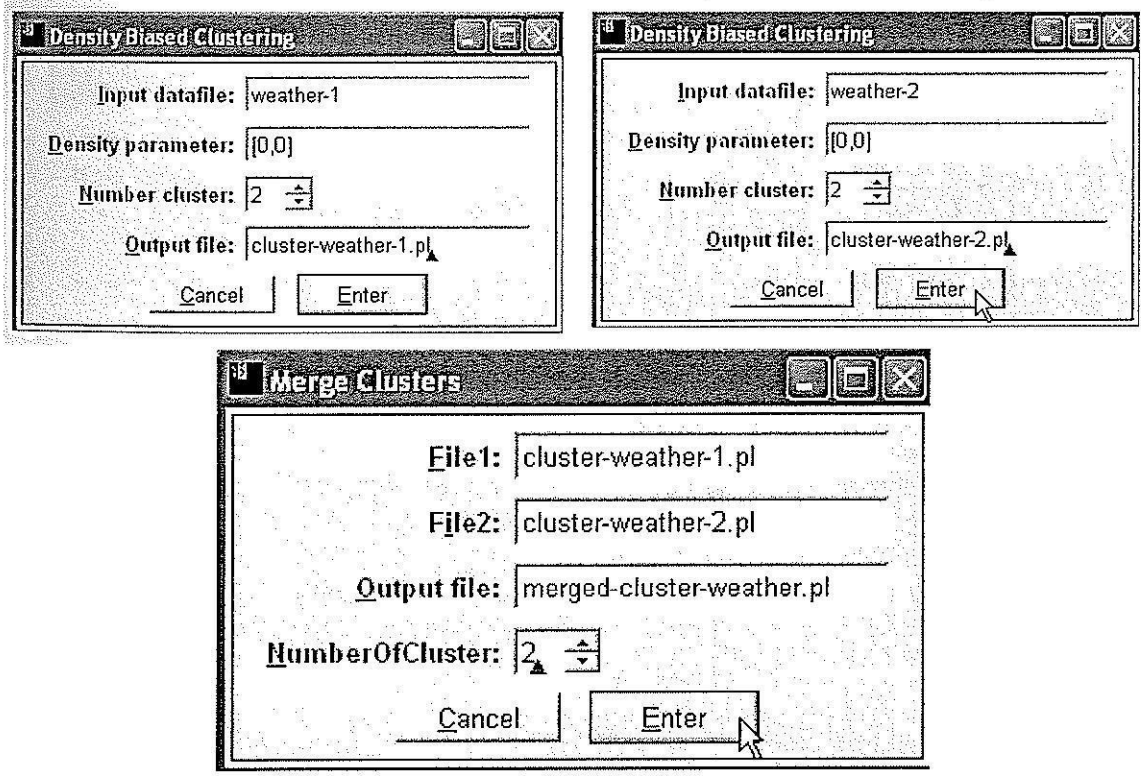
แนวคิดเกี่ยวกับการจัดกลุ่มข้อมูลแบบเพิ่มพูน หรือ incremental clustering เกิดจากการพยายามจัดกลุ่มกับข้อมูลที่มีขนาดใหญ่มาก ทำให้ต้องแบ่งจัดข้อมูลเป็นกลุ่มย่อยๆ จากนั้นจึงจะรวมข้อมูลในกลุ่มย่อย (merge clusters) ให้เป็นกลุ่มใหญ่ ในงานวิจัยนี้ใช้วิธีการรวมกลุ่มหรือคลัสเตอร์ โดยพิจารณาแต่ละ cluster mean ให้เป็นเสมือนหนึ่งรายการข้อมูล จากนั้น merge cluster means ให้ได้ค่า means ใหม่ ขั้นตอนการทำงานแสดงดังอัลกอริทึมต่อไปนี้

Algorithm 2 Incremental clustering

Input: cluster means from density-biased clustering
Output: a new set of merged cluster means

(1) Show GUI to obtain file names, F1 and F2, which are outputs from density-biased clustering
(2) Read number of desired clusters, K
(3) Read cluster descriptions from F1 and F2
(4) Treat cluster descriptions as data points and call the clustering process
(5) Output cluster means

จากข้อมูล weather ที่มีจำนวนข้อมูล 14 เรคคอร์ด เมื่อแบ่งข้อมูลออกเป็นไฟล์ย่อยสองไฟล์ โดยไฟล์แรก (weather-1) บันทึกข้อมูลเรคคอร์ที่ 1-7 และไฟล์ที่สอง (weather-2) บันทึกข้อมูลเรคคอร์ดที่ 8-14 จากนั้นทำ density-biased clustering กับข้อมูลแต่ละไฟล์ด้วยการกำหนดค่า K เป็น 2 และกำหนดพารามิเตอร์ความหนาแน่นของข้อมูลเป็น [0,0] บันทึกผลลัพธ์ของการทำ clustering กับข้อมูลทั้งสองชุดไว้ในไฟล์ cluster-weather-1.pl และ cluster-weather-2.pl ทำการรวมคลัสเตอร์ในทั้งสองไฟล์ด้วยการเรียกใช้โปรแกรม incremental clustering จะปรากฏจอภาพดังรูปที่ 2.10 เมื่อคลิกปุ่ม Enter จะได้ผลลัพธ์ของการรวมคลัสเตอร์ และปรากฏเป็นค่าคลัสเตอร์ใหม่ดังรูปที่ 2.11



รูปที่ 2.10 จอภาพของโปรแกรมการรวมคลัสเตอร์ในงานจัดกลุ่มข้อมูลแบบเพิ่มพูน

```
%--file:cluster-weather-1.pl
% output of clustering result
%density parameter=[0,0] ,No of cluster= 2
attribute(outlook,[sunny, overcast, rainy]).
attribute(temperature,[hot, mild, cool]).
attribute(humidity,[high, normal]).
attribute(windy,[true, false]).
attribute(class,[yes, no]).
allPoint([1-4-1, 2-3-1, 3-3-1, 4-2-1, 5-3-2, 6-4-2, 7-3-2]).

cluster(1,[outlook=sunny, temperature=hot,
        humidity=high, windy=false]).
cluster(2,[outlook=rainy, temperature=cool,
        humidity=normal, windy=true]).
cluster_member(1,[1, 2, 3, 4]).
cluster_member(2,[5, 6, 7]).

allCluster([1*[outlook=sunny, temperature=hot,
humidity=high, windy=false], 2*[outlook=rainy,
temperature=cool, humidity=normal, windy=true]]).
description('cluster-weather-1.pl',2).
```

(a) ผลลัพธ์จากการจัดกลุ่มข้อมูลในไฟล์ weather-1

```
%--file:cluster-weather-2.pl
% output of clustering result
%density parameter=[0,0] ,No of cluster= 2
attribute(outlook,[sunny, overcast, rainy]).
attribute(temperature,[hot, mild, cool]).
attribute(humidity,[high, normal]).
attribute(windy,[true, false]).
attribute(class,[yes, no]).
allPoint([1-3-1, 2-4-2, 3-2-2, 4-3-1, 5-3-1, 6-2-2, 7-3-1]).

cluster(1,[outlook=sunny, temperature=mild,
        humidity=high, windy=true]).
cluster(2,[outlook=sunny, temperature=cool,
        humidity=normal, windy=false]).
cluster_member(1,[1, 4, 5, 7]).
cluster_member(2,[2, 3, 6]).

allCluster([1*[outlook=sunny, temperature=mild,
humidity=high, windy=true], 2*[outlook=sunny,
temperature=cool, humidity=normal, windy=false]]).
description('cluster-weather-2.pl',2).
```

(b) ผลลัพธ์จากการจัดกลุ่มข้อมูลในไฟล์ weather-2

```
%file tmp.pl
:-dynamic attribute/2, instance/3.
attribute(outlook,[sunny, overcast, rainy]).
attribute(temperature,[hot, mild, cool]).
attribute(humidity,[high, normal]).
attribute(windy,[true, false]).
attribute(class,[yes, no]).
instance(1,class=yes,[outlook=sunny, temperature=hot, humidity=high, windy=false]).
instance(2,class=yes,[outlook=rainy, temperature=cool, humidity=normal, windy=true]).
instance(3,class=no,[outlook=sunny, temperature=mild, humidity=high, windy=true]).
instance(4,class=no,[outlook=sunny, temperature=cool, humidity=normal, windy=false]).
```

(c) ข้อมูลในไฟล์ชั่วคราวที่เกิดจากการรวมผลลัพธ์ของการจัดกลุ่มข้อมูลในขั้นตอน (a) และ (b)

merged-cluster-weather - Notepad

File Edit Format View Help

```
%--file:merged-cluster-weather.pl
% output of clustering result%density parameter=[0,0] ,No of cluster= 2
attribute(outlook,[sunny, overcast, rainy]).
attribute(temperature,[hot, mild, cool]).
attribute(humidity,[high, normal]).
attribute(windy,[true, false]).
attribute(class,[yes, no]).

allPoint([1-4-1, 2-4-2, 3-2-1, 4-2-1]).

cluster(1,[outlook=sunny, temperature=hot, humidity=high, windy=false]).
cluster(2,[outlook=rainy, temperature=cool, humidity=normal, windy=true]).
cluster_member(1,[1, 3, 4]).
cluster_member(2,[2]).

allCluster([1*[outlook=sunny, temperature=hot, humidity=high, windy=false],
        2*[outlook=rainy, temperature=cool, humidity=normal, windy=true]]).
description('merged-cluster-weather.pl',2).
```

(d) ผลลัพธ์สุดท้ายของการรวมคลัสเตอร์ในขั้นตอน (a) และ (b)

รูปที่ 2.11 ผลลัพธ์ของการจัดกลุ่มข้อมูลแบบเพิ่มพูนกับข้อมูล weather

เมื่อพิจารณาผลของการจัดกลุ่มข้อมูลแบบปกติ (เรียกว่าแบบ batch) ที่รวมข้อมูลทั้งหมดแล้วจัดเข้ากลุ่มในคราวเดียว เปรียบเทียบกับวิธี incremental ที่มีการแบ่งข้อมูลให้เป็นไฟล์ขนาดเล็กหลายไฟล์แล้วแยกจัดกลุ่มข้อมูลในแต่ละไฟล์ จากนั้นนำคลัสเตอร์ที่ได้มารวมจัดกลุ่มใหม่จะได้ผลลัพธ์เป็นค่า means (หรือลักษณะส่วนใหญ่ของกลุ่ม) ที่ใกล้เคียงกันมาก มีเพียงลักษณะในแอททริบิวต์ temperature ของคลัสเตอร์ที่สองเท่านั้นที่ค่าแตกต่างกัน สรุปการเปรียบเทียบได้ดังตารางที่ 2.2

ถ้าหากใช้ลักษณะค่า means นี้ไปจัดข้อมูลแต่ละเรคคอร์ดเข้ากลุ่มจะได้ผลการจัดกลุ่มที่ค่อนข้างใกล้เคียงกัน ดังแสดงในตารางที่ 2.3 การจัดข้อมูลเข้ากลุ่มในกรณีที่ข้อมูลมีระยะห่างจากคลัสเตอร์ที่หนึ่งและสองเท่ากัน จะแสดงผลการจัดกลุ่มเป็น “กลุ่ม 1 หรือ 2” ซึ่งหมายถึงจะจัดข้อมูลเข้ากลุ่มใดก็ได้ ถ้าหากผลการจัดกลุ่มเป็น “กลุ่ม 1 หรือ 2” แล้วตัดสินให้จัดข้อมูลเข้ากลุ่ม 1 จะได้ผลสรุปว่าวิธีการจัดข้อมูลเข้ากลุ่มแบบ incremental ให้ผลแตกต่างจากวิธีแบบ batch เพียงหนึ่งเรคคอร์ดจากข้อมูลทั้งหมด 14 เรคคอร์ด หรือคิดเป็น 7.14% ซึ่งจากผลการเปรียบเทียบนี้ทำให้คาดหมายได้ว่าเมื่อข้อมูลมีปริมาณมากกว่านี้ ความแตกต่างในผลการจัดกลุ่มแบบ batch และแบบ incremental จะมีนัยสำคัญที่ลดลง

ตารางที่ 2.2 เปรียบเทียบค่า means ของการจัดกลุ่มแบบ batch และแบบ incremental

วิธีการจัดกลุ่มข้อมูล	**ค่า means**
แบบ batch	Cluster1:[outlook=sunny,temperature=hot,humidity=high,windy=false] Cluster2:[outlook=rainy,temperature=mild,humidity=normal,windy=true]
แบบ incremental	Cluster1:[outlook=sunny,temperature=hot,humidity=high,windy=false] Cluster2:[outlook=rainy,temperature=cool,humidity=normal,windy=true]

ตารางที่ 2.3 ผลการจัดข้อมูลเข้ากลุ่มของวิธีการจัดกลุ่มแบบ batch และแบบ incremental

ข้อมูลเรคคอร์ดที่	ผลการจัดกลุ่มแบบ batch	ผลการจัดกลุ่มแบบ incremental
1	กลุ่ม 1	กลุ่ม 1
2	กลุ่ม 1	กลุ่ม 1
3	กลุ่ม 1	กลุ่ม 1
4	กลุ่ม 1	กลุ่ม 1 หรือ 2
5	กลุ่ม 2	กลุ่ม 2
6	กลุ่ม 2	กลุ่ม 2
7	กลุ่ม 2	กลุ่ม 2
8	กลุ่ม 1	กลุ่ม 1
9	กลุ่ม 1 หรือ 2	กลุ่ม 1
10	กลุ่ม 2	กลุ่ม 2
11	กลุ่ม 2	กลุ่ม 2
12	กลุ่ม 1 หรือ 2	กลุ่ม 2
13	กลุ่ม 1	กลุ่ม 1
14	กลุ่ม 2	กลุ่ม 2

บทที่ 3
การทดสอบโปรแกรม

วิธีการทดสอบประสิทธิภาพของการจัดกลุ่ม

โปรแกรมที่ใช้ในงานจัดกลุ่มข้อมูลประกอบด้วยโปรแกรม density-biased clustering ทำหน้าที่คัดเลือกข้อมูลตามความหนาแน่นเพื่อนำไปจัดกลุ่ม และโปรแกรม incremental clustering ทำหน้าที่จัดกลุ่มข้อมูลเช่นเดียวกัน แต่ใช้เทคนิคการแบ่งข้อมูลเป็นส่วนเล็กๆหลายส่วน เพื่อจัดข้อมูลในแต่ละกลุ่มเล็กให้เป็นคลัสเตอร์ จากนั้นขยายขอบเขตการจัดกลุ่มโดยรวมลักษณะเด่นในแต่ละกลุ่มย่อยของข้อมูล (merge clusters)โดยใช้ค่า means ของแต่ละกลุ่มย่อย มาพิจารณาเพื่อรวมเป็นกลุ่มใหญ่ ทำให้ได้ผลลัพธ์เป็นค่า means ค่าใหม่ที่เป็นตัวแทนข้อมูลของหลายกลุ่มย่อย จากนั้นรวมกลุ่มย่อยของข้อมูลเช่นนี้จนครบจำนวนข้อมูลทั้งหมด

การทดสอบประสิทธิภาพของโปรแกรม จะใช้วิธีทดสอบผลการรวมกลุ่มข้อมูลของวิธี incremental clustering ที่ทยอยจัดกลุ่มข้อมูลที่มีขนาดเล็กแล้วรวมค่า means ของข้อมูลกลุ่มย่อย ให้เป็นค่า means ของกลุ่มที่ใหญ่ขึ้น เปรียบเทียบกับวิธีการจัดกลุ่มข้อมูลแบบ batch ซึ่งจะต้องรวบรวมข้อมูลทั้งหมดให้เป็นไฟล์เดียวแล้วจึงทำการจัดกลุ่มข้อมูล การเปรียบเทียบผลการจัดกลุ่มจะใช้การพิจารณาข้อมูลที่เป็นสมาชิกในแต่ละกลุ่มของวิธีจัดกลุ่มแบบ incremental เปรียบเทียบกับสมาชิกของกลุ่มที่จัดแบบ batch และเปรียบเทียบคุณภาพการรวมกลุ่มของข้อมูลด้วยการคำนวณค่า sum of squared errors (SSE) ค่านี้จะเป็นค่าโดยอ้อมในการบ่งชี้ความหนาแน่นของกลุ่มข้อมูล ค่า SSE ที่ต่ำกว่าจะหมายถึงการเกาะกลุ่มของข้อมูลที่ดีกว่า

ข้อมูลที่ใช้ในการทดสอบประกอบด้วย 3 ชุดข้อมูลได้แก่ (1) ข้อมูล post-operative (เป็นข้อมูลเกี่ยวกับการสังเกตอาการของคนไข้ภายหลังการผ่าตัด จำนวนข้อมูล 86 เรคคอร์ด ข้อมูลแต่ละเรคคอร์ดประกอบด้วย 8 แอททริบิวต์), (2) ข้อมูล breast cancer (เป็นข้อมูลเกี่ยวกับการวินิจฉัยการเกิดซ้ำของมะเร็งในคนไข้ที่เคยเป็นมะเร็งเต้านม จำนวนข้อมูล 191 เรคคอร์ด ข้อมูลแต่ละเรคคอร์ดประกอบด้วย 9 แอททริบิวต์), และ (3) ข้อมูล mushroom (เป็นข้อมูลเกี่ยวกับลักษณะของเห็ดที่เป็นพิษและเห็ดที่รับประทานได้ จำนวนข้อมูล 5,416 เรคคอร์ด ข้อมูลแต่ละเรคคอร์ดประกอบด้วย 23 แอททริบิวต์)

ขั้นตอนการทดลองจะเริ่มต้นด้วยการเตรียมข้อมูลเพื่อจะเป็นข้อมูลเข้าให้กับโปรแกรม density-biased clustering โดยโปรแกรมนี้สามารถจัดกลุ่มข้อมูลแบบ batch ได้โดยการกำหนดพารามิเตอร์ความหนาแน่นของข้อมูลเป็น [0,0] ซึ่งหมายถึงไม่ต้องมีการคัดเลือกข้อมูลที่หนาแน่นถึงเกณฑ์ แต่จะใช้ข้อมูลทั้งหมดในการจัดกลุ่มเพื่อหาลักษณะเด่นของกลุ่มข้อมูล จากนั้นใช้ลักษณะเด่นที่ได้นี้ไปเป็นเกณฑ์ในการจัดข้อมูลแต่ละเรคคอร์ดเข้ากลุ่ม

ในการเตรียมข้อมูลเพื่อทดสอบกับโปรแกรม incremental clustering จะแบ่งข้อมูลเป็นสี่ส่วนย่อย จากนั้นทำการจัดกลุ่มในแต่ละส่วนย่อย ผลลัพธ์ที่ได้จะเป็นลักษณะเด่นของกลุ่มข้อมูลในแต่ละส่วนย่อย (หรือค่า means) จากนั้นใช้ลักษณะเด่นนี้เป็นเสมือนข้อมูล เพื่อทำการ merge ค่า means ของแต่ละคลัสเตอร์ เมื่อรวมค่า means ของข้อมูลทั้งสี่ส่วนย่อยเสร็จ จะได้ค่า means สุดท้ายของข้อมูลทั้งหมด นำค่า means สุดท้ายนี้เป็นเกณฑ์ในการจัดข้อมูลแต่ละเรคคอร์ดเข้ากลุ่ม ตรวจสอบผลลัพธ์ของวิธีจัดกลุ่มแบบ incremental เปรียบเทียบกับแบบ batch โดยการพิจารณาความแตกต่างของการจัดข้อมูลเข้ากลุ่มว่ามีข้อมูลกี่เรคคอร์ดที่ถูกจัดเข้ากลุ่มแตกต่างกัน และพิจารณาการเกาะกลุ่มของข้อมูลที่ได้จากการจัดทั้งสองแบบด้วยค่า SSE รวมถึงพิจารณาเวลาที่ใช้ในกระบวนการจัดกลุ่มทั้งหมด

ผลการทดสอบ

การทดสอบประสิทธิภาพของวิธีการการจัดกลุ่มข้อมูลในแบบ batch เปรียบเทียบกับแบบ incremental ของข้อมูล post-operative แสดงผลการทดสอบได้ดังตารางที่ 3.1 ผลการทดสอบกับข้อมูล breast cancer แสดงได้ดังตารางที่ 3.2 และผลการทดสอบกับข้อมูล mushroom แสดงได้ดังตารางที่ 3.3 ผลการทดสอบที่แสดงในตารางทั้งสามนี้เป็นขั้นตอนเริ่มต้นที่ยังไม่พิจารณาคัดเลือกข้อมูลตามความหนาแน่น โดยในการรันโปรแกรม density-biased clustering จะกำหนดพารามิเตอร์ค่าความหนาแน่นเป็น [0,0]

การทดสอบในขั้นตอนที่สอง เป็นการตรวจสอบผลของการคัดเลือกข้อมูลตามความหนาแน่นที่เกณฑ์ขั้นต่ำขนาดต่างๆเพื่อจัดกลุ่มข้อมูลทั้งในแบบ batch และแบบ incremental เกณฑ์ความหนาแน่นจะเริ่มทดสอบที่ [1,0.1] หมายถึงการคำนวณความหนาแน่นของข้อมูลจะพิจารณาค่า similarity ของหนึ่งแอททริบิวต์และจะต้องมีข้อมูลที่คล้ายคลึงกับข้อมูลนั้นอย่างต่ำ 0.1 หรือ 10% เกณฑ์ที่ใช้จะเพิ่มขึ้นเป็น [1,0.15], [1,0.2], ..., [4,0.2] ผลการทดสอบในขั้นตอนนี้จะแสดงเฉพาะค่า SSE ของผลการจัดกลุ่มข้อมูลที่แต่ละค่าความหนาแน่น และที่จำนวนกลุ่มข้อมูล (ค่า K) ที่ขนาดต่างๆกัน ผลการทดสอบกับข้อมูล post-operative แสดงดังตารางที่ 3.4 ผลการทดสอบกับข้อมูล breast cancer แสดงดังตารางที่ 3.5 และผลการทดสอบกับข้อมูล mushroom แสดงดังตารางที่ 3.6

ตารางที่ 3.1 เปรียบเทียบผลการจัดกลุ่มข้อมูล post-operative ด้วยวิธีจัดกลุ่มแบบ batch และแบบ incremental

จำนวนกลุ่ม (K)	การจัดกลุ่มแบบ batch			การจัดกลุ่มแบบ incremental			ความแตกต่างของการจัดข้อมูลเข้ากลุ่ม
	จำนวนข้อมูลในกลุ่ม	เวลาที่ใช้ (วินาที)	ค่า SSE	จำนวนข้อมูลในกลุ่ม	เวลาที่ใช้ (วินาที)	ค่า SSE	
2	Cluster1=40 Cluster2=46	1.23	1742.88	Cluster1=42 Cluster2=44	1.06	1695.53	6.97%
3	Cluster1=36 Cluster2=22 Cluster3=28	1.40	1401.66	Cluster1=33 Cluster2=26 Cluster3=27	1.18	1368.74	9.30%
4	Cluster1=26 Cluster2=21 Cluster3=21 Cluster4=18	1.44	1281.33	Cluster1=28 Cluster2=20 Cluster3=21 Cluster4=17	1.17	1195.67	8.14%
5	Cluster1=20 Cluster2=19 Cluster3=14 Cluster4=17 Cluster5=16	1.51	1017.34	Cluster1=23 Cluster2=20 Cluster3=16 Cluster4=17 Cluster5=10	1.33	1145.98	11.63%
6	Cluster1=17 Cluster2=14 Cluster3=16 Cluster4=14 Cluster5=15 Cluster6=10	1.49	967.85	Cluster1=18 Cluster2=15 Cluster3=15 Cluster4=14 Cluster5=14 Cluster6=10	1.40	913.33	6.97%
7	Cluster1=15 Cluster2=11 Cluster3=13 Cluster4=10 Cluster5=19 Cluster6=11 Cluster7=7	1.62	712.63	Cluster1=14 Cluster2=10 Cluster3=14 Cluster4=11 Cluster5=13 Cluster6=12 Cluster7=12	1.48	654.87	10.46%

ตารางที่ 3.2 เปรียบเทียบผลการจัดกลุ่มข้อมูล breast cancer ด้วยวิธีจัดกลุ่มแบบ batch และแบบ incremental

จำนวนกลุ่ม (K)	การจัดกลุ่มแบบ batch			การจัดกลุ่มแบบ incremental			ความแตกต่างของการจัดข้อมูลเข้ากลุ่ม
	จำนวนข้อมูลในกลุ่ม	เวลาที่ใช้ (วินาที)	ค่า SSE	จำนวนข้อมูลในกลุ่ม	เวลาที่ใช้ (วินาที)	ค่า SSE	
2	Cluster1=123 Cluster2=68	1.54	2013.33	Cluster1=131 Cluster2=80	1.37	1967.67	11.52%
3	Cluster1=81 Cluster2=63 Cluster3=47	2.13	1846.78	Cluster1=79 Cluster2=65 Cluster3=47	1.43	1731.11	3.66%
4	Cluster1=51 Cluster2=46 Cluster3=46 Cluster4=48	2.09	1701.26	Cluster1=49 Cluster2=47 Cluster3=49 Cluster4=46	1.51	1691.98	5.76%
5	Cluster1=43 Cluster2=38 Cluster3=35 Cluster4=37 Cluster5=38	1.47	1568.14	Cluster1=45 Cluster2=32 Cluster3=34 Cluster4=39 Cluster5=41	1.39	1446.72	7.33%
6	Cluster1=34 Cluster2=29 Cluster3=30 Cluster4=27 Cluster5=37 Cluster6=34	1.56	1432.59	Cluster1=31 Cluster2=31 Cluster3=32 Cluster4=30 Cluster5=35 Cluster6=32	1.48	1303.33	6.28%
7	Cluster1=28 Cluster2=28 Cluster3=24 Cluster4=23 Cluster5=25 Cluster6=33 Cluster7=30	2.17	1173.79	Cluster1=29 Cluster2=29 Cluster3=28 Cluster4=29 Cluster5=28 Cluster6=24 Cluster7=24	1.56	1072.13	8.37%

ตารางที่ 3.3 เปรียบเทียบผลการจัดกลุ่มข้อมูล mushroom ด้วยวิธีจัดกลุ่มแบบ batch และแบบ incremental

จำนวนกลุ่ม (K)	การจัดกลุ่มแบบ batch			การจัดกลุ่มแบบ incremental			ความแตกต่างของการจัดข้อมูลเข้ากลุ่ม
	จำนวนข้อมูลในกลุ่ม	เวลาที่ใช้ (วินาที)	ค่า SSE	จำนวนข้อมูลในกลุ่ม	เวลาที่ใช้ (วินาที)	ค่า SSE	
2	Cluster1=2011 Cluster2=3405	36.23	9523.61	Cluster1=2004 Cluster2=3412	31.15	9601.33	6.92%
3	Cluster1=1611 Cluster2=1804 Cluster3=2001	41.13	9411.79	Cluster1=1596 Cluster2=1732 Cluster3=2088	33.26	9398.41	7.40%
4	Cluster1=1114 Cluster2=1096 Cluster3=1354 Cluster4=1852	51.48	8713.33	Cluster1=1212 Cluster2=1107 Cluster3=1257 Cluster4=1840	39.17	9005.17	8.01%
5	Cluster1=993 Cluster2=1091 Cluster3=1014 Cluster4=1176 Cluster5=1142	72.11	7017.51	Cluster1=1023 Cluster2=1220 Cluster3=1016 Cluster4=1072 Cluster5=1085	51.48	6805.22	12.72%
6	Cluster1=817 Cluster2=941 Cluster3=766 Cluster4=1004 Cluster5=783 Cluster6=1105	81.47	5967.85	Cluster1=882 Cluster2=815 Cluster3=1015 Cluster4=914 Cluster5=747 Cluster6=1043	73.59	5913.33	7.97%
7	Cluster1=615 Cluster2=711 Cluster3=813 Cluster4=810 Cluster5=919 Cluster6=911 Cluster7=637	93.47	3712.63	Cluster1=714 Cluster2=710 Cluster3=723 Cluster4=915 Cluster5=699 Cluster6=825 Cluster7=830	78.29	3654.87	9.16%

ตารางที่ 3.4 ค่า SSE ของการจัดกลุ่มข้อมูลตามความหนาแน่นแบบ batch และแบบ incremental ของข้อมูล post-operative

K	Clustering method	ค่า SSE ตามความหนาแน่นของข้อมูล										
		[1,0.1]	[1,0.15]	[1,0.2]	[2,0.1]	[2,0.15]	[2,0.2]	[3,0.1]	[3,0.15]	[3,0.2]	[4,0.1]	[4,0.15]
2	Batch	1739.1	1740.6	1749.3	1702.3	1716.9	1741.1	1713.3	1701.1	1722.4	1756.3	1754.8
	Incremental	1670.3	1687.3	1675.1	1689.0	1691.8	1703.7	1643.4	1629.7	1707.4	1775.6	1684.6
3	Batch	1445.6	1462.9	1508.9	1677.5	1544.2	1493.2	1389.9	1401.2	1599.7	1611.0	1543.5
	Incremental	1395.5	1401.2	1434.7	1504.1	1400.2	1542.7	1358.2	1341.4	1477.8	1489.5	1531.2
4	Batch	1372.2	1387.1	1401.0	1512.9	1488.7	1395.5	1321.1	1356.7	1407.9	1417.3	1399.0
	Incremental	1311.3	1402.6	1398.7	1476.5	1359.2	1344.6	1305.2	1296.1	1307.7	1369.0	1304.4
5	Batch	1196.7	1209.3	1136.5	1227.6	1235.5	1409.9	1287.6	1012.5	1293.7	1785.2	1303.3
	Incremental	1112.2	1301.0	1198.6	1211.1	1109.5	1245.0	1193.3	1074.6	1249.0	1329.5	1268.7
6	Batch	1071.9	1192.7	1068.5	1201.9	1137.8	1224.3	1125.6	1016.5	1298.0	1173.4	1243.0
	Incremental	1132.4	1098.3	1163.4	1255.7	1190.4	1207.3	1114.8	1079.8	1212.5	1249.0	1300.7
7	Batch	978.3	895.6	913.2	965.1	970.2	899.0	901.7	816.5	893.2	907.6	954.5
	Incremental	913.3	941.6	932.8	899.0	928.4	919.6	829.5	807.7	901.3	916.5	907.4
8	Batch	965.4	1008.2	974.7	966.2	1105.7	989.3	891.4	902.6	972.4	1123.7	1065.9
	Incremental	891.2	903.4	876.2	1025.3	998.6	916.7	875.3	892.0	913.4	936.5	997.8
9	Batch	901.3	932.7	890.5	912.7	934.1	899.6	813.2	829.4	937.6	929.4	956.5
	Incremental	876.3	832.7	890.1	885.9	906.7	813.2	836.7	891.0	903.7	919.5	896.0
10	Batch	896.5	942.1	937.6	881.3	920.5	874.3	832.6	879.5	916.7	903.2	924.0
	Incremental	901.2	928.7	936.5	890.1	879.0	892.6	851.1	900.3	899.6	917.2	906.1
11	Batch	872.4	884.5	832.1	864.3	901.1	856.2	813.4	876.7	913.2	892.0	936.5
	Incremental	906.3	872.5	843.1	892.7	819.9	769.5	793.3	812.7	939.6	895.4	901.8
12	Batch	883.1	872.9	853.1	902.4	875.9	836.2	813.7	859.0	913.4	898.2	913.2
	Incremental	792.8	813.5	826.7	913.8	885.4	763.2	799.0	834.2	865.9	913.4	920.6

ตารางที่ 3.5 ค่า SSE ของการจัดกลุ่มข้อมูลตามความหนาแน่นแบบ batch และแบบ incremental ของข้อมูล breast cancer

K	Clustering method	ค่า SSE ตามความหนาแน่นของข้อมูล											
		[1,0.1]	[1,0.15]	[1,0.2]	[2,0.1]	[2,0.15]	[2,0.2]	[3,0.1]	[3,0.15]	[3,0.2]	[4,0.1]	[4,0.15]	[4,0.2]
2	Batch	2046.3	2137.0	2246.9	2012.5	2354.1	2168.4	2079.3	1958.2	2143.2	2013.4	2019.9	2245.6
	Incremental	1975.7	1983.2	2034.1	1989.7	2154.3	2011.9	1964.3	1941.7	1985.0	2054.2	2098.7	2213.7
3	Batch	1982.3	2013.5	2163.4	2098.2	2175.9	1972.6	2001.8	1935.1	1972.7	1908.3	2085.2	2163.0
	Incremental	1763.4	1865.4	1795.0	1932.4	1955.7	1843.9	1932.3	1832.7	1954.0	1885.9	1932.7	1915.4
4	Batch	1712.6	1698.7	1734.6	1800.3	1763.9	1756.4	1831.2	1749.2	1654.1	1693.2	1834.9	1755.3
	Incremental	1678.3	1732.1	1688.7	1709.7	1789.5	1703.2	1688.4	1603.0	1642.5	1659.2	1701.3	1685.4
5	Batch	1543.1	1612.3	1593.8	1600.1	1572.8	1559.6	1482.0	1501.3	1459.7	1532.7	1468.9	1577.0
	Incremental	1437.3	1389.3	1372.7	1401.3	1388.2	1369.4	1419.5	1344.2	1465.1	1370.6	1458.7	1492.1
6	Batch	1402.9	1472.5	1439.0	1372.6	1385.0	1409.2	1376.9	1312.1	1308.9	1427.1	1365.7	1433.0
	Incremental	1371.7	1362.8	1398.4	1432.5	1411.0	1349.6	1451.0	1327.8	1309.5	1420.7	1372.6	1419.6
7	Batch	1372.5	1419.2	1352.6	1275.7	1293.0	1321.7	1286.5	1203.4	1384.2	1358.7	1402.6	1326.7
	Incremental	1218.9	1239.0	1311.2	1168.7	1203.4	1195.9	1202.7	1134.9	1310.2	1287.4	1197.3	1246.5
8	Batch	1132.7	1068.2	1143.7	1079.2	1159.0	1086.5	1034.2	1007.8	1013.4	1179.0	1092.3	1154.7
	Incremental	1107.6	1195.7	1183.2	1096.7	1132.4	1097.5	1106.3	1002.4	1146.9	1093.7	1089.0	1103.1
9	Batch	893.6	912.4	895.6	903.1	943.5	879.6	907.4	859.6	935.2	911.7	909.3	892.1
	Incremental	932.7	942.7	939.5	891.2	903.4	950.2	875.1	832.7	913.2	894.7	953.0	912.7
10	Batch	793.9	765.9	801.4	865.9	793.2	786.4	713.0	759.2	803.4	860.9	903.1	895.2
	Incremental	684.9	713.2	695.7	801.5	738.2	679.8	695.0	713.2	699.3	715.4	769.2	814.0
11	Batch	714.5	726.9	774.0	693.1	732.9	680.2	613.4	695.2	787.0	695.3	761.2	730.8
	Incremental	732.2	695.3	689.4	719.0	677.0	613.5	589.0	632.4	715.6	709.8	674.9	632.1
12	Batch	632.8	579.2	684.3	588.7	516.5	543.2	508.7	640.1	632.2	619.5	593.2	587.6
	Incremental	588.7	535.2	645.9	607.1	580.3	511.7	573.2	546.0	532.9	611.4	590.2	578.4

ตารางที่ 3.6 ค่า SSE ของการจัดกลุ่มข้อมูลตามความหนาแน่นแบบ batch และแบบ incremental ของข้อมูล mushroom

K	Clustering method	ค่า SSE ตามความหนาแน่นของข้อมูล										
		[1,0.1]	[1,0.15]	[1,0.2]	[2,0.1]	[2,0.15]	[2,0.2]	[3,0.1]	[3,0.15]	[3,0.2]	[4,0.1]	[4,0.15]
2	Batch	9132.3	9027.6	9016.9	9012.5	9004.1	8968.4	8319.3	7758.2	7143.2	7013.4	6819.9
	Incremental	8995.2	8883.1	8834.1	8819.7	8354.3	8011.9	7964.3	6941.7	6685.0	6454.2	6098.7
3	Batch	8992.5	8213.7	8166.7	8098.2	7575.9	6912.6	6001.8	5935.1	5872.7	5808.3	5085.2
	Incremental	8769.1	7865.9	7795.2	6932.7	6905.7	6843.9	5932.7	5832.3	4954.0	4885.9	4732.7
4	Batch	8712.6	8698.7	7934.6	7800.3	7763.9	6756.4	5831.2	5749.2	5654.1	4693.2	3834.9
	Incremental	8678.3	7732.1	6188.7	6009.7	5789.5	5703.2	5688.4	4693.9	4602.5	3959.2	3701.3
5	Batch	8543.1	8612.3	8593.8	8300.1	8272.8	7559.6	7482.0	6501.3	5956.7	5532.7	5468.9
	Incremental	8437.3	7389.3	7372.7	7401.3	7388.2	6869.4	6419.5	6014.2	4651.1	4370.6	4582.7
6	Batch	8402.9	8472.5	7939.0	7872.6	7385.0	6809.2	6376.9	6312.1	6008.9	5927.1	5665.7
	Incremental	8371.7	8362.8	7398.4	7432.5	6941.0	6349.6	6151.0	5927.8	5909.2	5820.7	5372.6
7	Batch	7372.5	7119.2	7052.6	6975.7	6893.0	6321.7	6286.5	6103.4	5884.2	5358.7	5402.6
	Incremental	7018.9	6939.0	6911.2	6808.7	6503.4	6195.9	6002.7	5934.9	5710.2	5287.4	5197.3
8	Batch	7132.7	7068.2	6943.7	6079.2	5809.0	5786.5	5034.2	5007.8	4713.4	4739.0	4092.3
	Incremental	6907.6	6895.7	6183.2	5996.7	5732.4	5697.5	4906.3	4802.4	4693.9	4093.7	3789.0
9	Batch	6893.6	7912.4	7895.6	6903.1	5943.5	5279.6	4907.4	4859.6	3735.2	3011.7	2909.3
	Incremental	6932.7	7942.7	6939.5	6891.2	5103.4	4950.2	4815.1	4762.7	2913.2	2894.7	2153.0
10	Batch	5793.9	5065.9	4801.4	4865.9	4793.2	4386.4	3713.0	3759.2	3203.4	2860.9	2003.1
	Incremental	5684.9	4913.2	4695.7	4201.5	4038.2	3979.8	3695.0	3013.2	2999.3	2715.4	1969.2
11	Batch	4714.5	4726.9	4774.0	3693.1	3732.9	3680.2	3613.4	3695.2	2787.0	2695.3	1961.2
	Incremental	4732.2	4695.3	4689.4	3719.0	3677.0	3613.5	3589.0	3632.4	2015.6	1979.8	1784.9
12	Batch	2632.7	2539.2	2484.3	2388.7	2316.5	2143.2	1978.7	1640.1	1632.2	1609.5	1593.2
	Incremental	2588.1	2435.7	2375.9	2207.1	2180.3	2011.7	1573.2	1546.0	1502.9	1491.4	1390.2

อภิปรายผล

การจัดกลุ่มข้อมูลแบบ batch และแบบ incremental เมื่อไม่ต้องคำนึงถึงความหนาแน่นของข้อมูล (ผลการทดสอบตามตารางที่ 3.1, 3.2 และ 3.3) พบว่าการจัดกลุ่มข้อมูลแบบ incremental ใช้เวลาโดยรวมน้อยกว่าแบบ batch และข้อมูลที่ถูกจัดเข้ากลุ่มในขั้นสุดท้ายของทั้งสองวิธีการมีความแตกต่างกันไม่เกิน 13% เมื่อพิจารณาคุณภาพของกลุ่มข้อมูลจากค่า SSE พบว่าวิธีจัดกลุ่มข้อมูลแบบ incremental ให้ผลลัพธ์เป็นกลุ่มข้อมูลที่เกาะกลุ่มกันค่อนข้างดี และการเกาะกลุ่มจะดีขึ้นเมื่อเพิ่มจำนวนกลุ่ม แต่ความแตกต่างของค่า SSE จะไม่สูงมากนัก โดยเฉลี่ยในข้อมูลทุกชุดค่าความแตกต่างจะอยู่ระหว่าง 2-8%

เมื่อทดลองคัดเลือกข้อมูลตัวแทนตามค่าความหนาแน่น เพื่อนำมาใช้จัดกลุ่มและหาค่า mean ของกลุ่ม พบว่า (ตามผลที่ได้ในตารางที่ 3.4 และ 3.5) วิธีการคัดเลือกข้อมูลตัวแทน จะให้ผลการจัดกลุ่มที่ดีกว่าการใช้ข้อมูลทั้งหมดมาคำนวณหาค่า mean ของกลุ่ม และเกณฑ์ขั้นต่ำในการคัดเลือกข้อมูลตัวแทนอยู่ที่ประมาณ [3,0.1], [3,0.15] และ [4,0.1] นั่นคือควรจะพิจารณาคัดเลือกข้อมูลตัวแทนที่มีค่าความหนาแน่นอยู่ระหว่าง 0.1 ถึง 0.15 (หรือเป็นข้อมูลที่มีข้อมูลอื่นอยู่ใกล้เคียงอย่างต่ำ 10-15%) และการพิจารณาความใกล้เคียงใช้การเปรียบเทียบค่าของแอททริบิวต์อย่างต่ำ 3 หรือ 4 แอททริบิวต์ แต่ในข้อมูล mushroom ที่เป็นข้อมูลขนาดใหญ่ ผลการทดลองตามตารางที่ 3.6 ชี้ให้เห็นว่าการพิจารณาคัดเลือกข้อมูลตัวแทนที่มีค่าความหนาแน่นขั้นต่ำ 0.2 และการพิจารณาความใกล้เคียงใช้การเปรียบเทียบค่าของแอททริบิวต์อย่างต่ำ 4 แอททริบิวต์ จะให้ผลการจัดกลุ่มที่ดี

บทที่ 4
บทสรุป

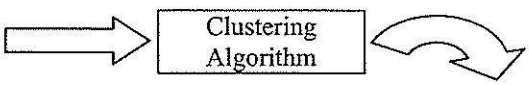
สรุปผลการวิจัย

การวิเคราะห์ข้อมูลด้วยวัตถุประสงค์ที่จะค้นหารูปแบบ หรือลักษณะเด่นภายในกลุ่มข้อมูลเพื่อที่จะนำความรู้ที่ได้ไปใช้ประโยชน์ มักจะเริ่มต้นกระบวนการวิเคราะห์ด้วยการแยกข้อมูลออกเป็นกลุ่ม ภายในแต่ละกลุ่มจะประกอบด้วยข้อมูลที่คล้ายกัน ต่อจากนั้นจึงจะเป็นการวิเคราะห์โดยละเอียดกับข้อมูลแต่ละกลุ่มเพื่อหาลักษณะร่วม, แนวโน้มการเปลี่ยนแปลง หรือลักษณะอื่นๆ ตามที่นักวิเคราะห์สนใจ การจัดกลุ่มจึงมักจะเป็นขั้นตอนแรกของกระบวนการวิเคราะห์ข้อมูล และเป็นขั้นตอนการเรียนรู้ที่เรียกว่า unsupervised learning เนื่องจากไม่มีการชี้แนะว่าควรจะจัดข้อมูลเป็นกลุ่มโดยพิจารณาจากลักษณะใดของข้อมูล การจัดกลุ่มจึงต้องพิจารณาทุกลักษณะในข้อมูลแต่ละตัว ซึ่งต่างจากขั้นตอนหลังจากนี้ที่ข้อมูลได้รับการกำหนดกลุ่มแล้ว การวิเคราะห์ข้อมูลจึงจะมีลักษณะเจาะจงมากขึ้น เช่น กำหนดให้ศึกษาลักษณะของข้อมูลในกลุ่มที่ 1 ว่าแตกต่างจากข้อมูลในกลุ่มที่ 2 อย่างไร การศึกษาลักษณะภายในข้อมูลที่เจาะจงมากขึ้นนี้เรียกว่า supervised learning

ตัวอย่างในรูปที่ 4.1 แสดงการจัดกลุ่มข้อมูล iris ด้วยโปรแกรม k-means ข้อมูลประกอบด้วยความยาวและความกว้างของกลีบดอกไอริส 3 สายพันธุ์ ผลลัพธ์ของการจัดกลุ่มจะได้เป็นค่า mean ของความยาวและความกว้างของกลีบดอกไอริสในแต่ละกลุ่ม เมื่อจัดกลุ่มข้อมูลได้แล้วจึงจะนำข้อมูลที่มีการระบุกลุ่ม (รูปที่ 4.2) ไปวิเคราะห์ด้วยอัลกอริทึม classification ซึ่งจัดเป็นอัลกอริทึมประเภท supervised learning เนื่องจากได้รับการชี้แนะว่าให้จำแนกข้อมูลตามกลุ่ม (cluster) ผลลัพธ์ที่ได้จึงเป็นการจำแนกลักษณะของข้อมูลในกลุ่มที่ 1, 2 และ 3 หรืออาจพิจารณาได้ว่าเป็นการบรรยาย (describe) ลักษณะของข้อมูลในแต่ละกลุ่ม

ข้อมูลที่ยังไม่ได้จัดกลุ่ม

Sepal Length	Sepal Width	Petal Length	Petal Width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
5.4	3.9	1.7	0.4
7.0	3.2	5.8	2.2
6.5	3.0	5.8	2.2
...	...	...	...
5.9	3.0	5.1	1.8

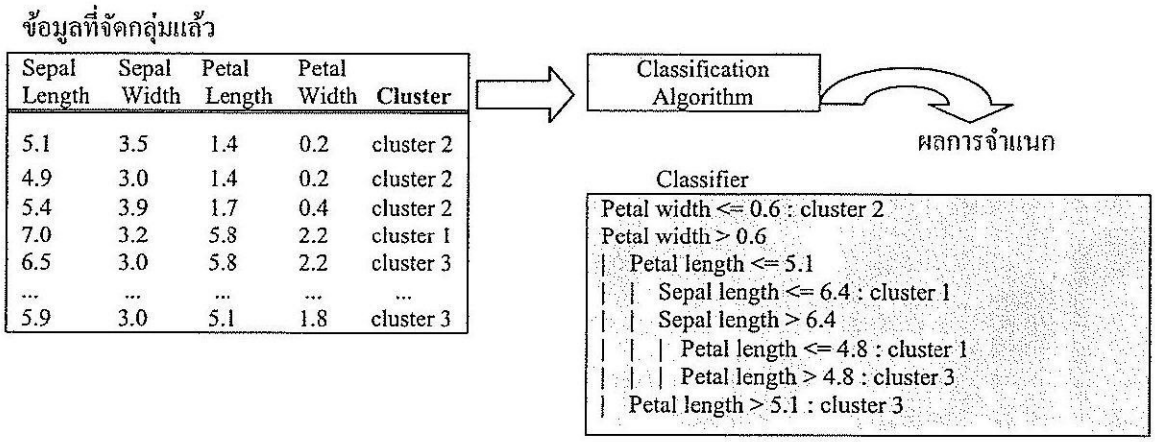


ผลการจัดกลุ่ม

Cluster centers:

	Sepal Length	Sepal Width	Petal Length	Petal Width
Cluster 1	5.8885	2.7377	4.3967	1.418
Cluster 2	5.006	3.418	1.464	0.244
Cluster 3	6.8462	3.0281	5.7026	2.0795

รูปที่ 4.1 การทำ unsupervised learning ด้วย clustering algorithm



รูปที่ 4.2 การทำ supervised learning ด้วย classification algorithm

การจัดกลุ่มข้อมูลจึงเป็นขั้นตอนที่สำคัญของงานวิเคราะห์ข้อมูล และจากลักษณะของการเรียนรู้แบบ unsupervised learning เวลาที่ใช้ในการทำงานจึงมักจะสูงกว่าการเรียนรู้ในแบบ supervised learning เวลาที่ใช้นี้จะผันแปรตามจำนวนมิติของข้อมูลและปริมาณของข้อมูล เมื่อข้อมูลมีปริมาณสูงมากอัลกอริทึมการจัดกลุ่มข้อมูลจะทำงานได้ช้าลง หรือถ้าข้อมูลมีปริมาณมากจนเกินไปจะไม่สามารถทำงานได้เนื่องจากปริมาณข้อมูลมีมากเกินกว่าความจุของหน่วยความจำ ถ้าต้องการปรับปรุงอัลกอริทึมจัดกลุ่มข้อมูลให้สามารถทำงานกับข้อมูลปริมาณสูงมากได้จะต้องใช้วิธีการจัดกลุ่มข้อมูลแบบเพิ่มพูนและใช้ข้อมูลตัวแทนในการพิจารณาค่า mean ของกลุ่ม

โครงการวิจัยนี้มีวัตถุประสงค์หลักคือ การออกแบบและพัฒนาเทคนิคการจัดกลุ่มข้อมูล โดยในโครงการวิจัยนี้ได้นำเสนอวิธีการจัดกลุ่มข้อมูลในแบบปกติ เรียกว่าการจัดกลุ่มข้อมูลแบบ batch นั่นคือใช้ข้อมูลทั้งหมดในคราวเดียวเพื่อการจัดกลุ่มข้อมูล และนำเสนอวิธีการจัดกลุ่มข้อมูลแบบเพิ่มพูน หรือ incremental clustering ที่ใช้การแบ่งข้อมูลที่จะจัดกลุ่มออกเป็นส่วนย่อย จากนั้นจัดกลุ่มข้อมูลในส่วนย่อย แล้วจึงนำค่า mean ของกลุ่มข้อมูลซึ่งเป็นผลลัพธ์ที่ได้จากการจัดกลุ่มข้อมูลย่อย มา merge เพื่อค้นหาค่า mean ใหม่ที่เหมาะสมจะเป็นลักษณะตัวแทนของกลุ่ม ผลการทดลองพบว่า วิธีการจัดกลุ่มข้อมูลแบบ batch และแบบ incremental ให้ผลลัพธ์สุดท้ายเป็นกลุ่มข้อมูลที่มีสมาชิกในกลุ่มไม่แตกต่างกันมาก แต่วิธี incremental clustering จะใช้เวลาในการประมวลผลน้อยกว่า

นอกจากเทคนิค incremental clustering แล้ว โครงการวิจัยนี้ยังได้นำเสนอเทคนิคการจัดกลุ่มข้อมูลตามความหนาแน่น โดยใช้วิธีการคัดเลือกข้อมูลที่มีความหนาแน่นตรงตามเกณฑ์ที่กำหนด มาพิจารณาค่า mean ของกลุ่ม เพื่อที่ในขั้นตอนสุดท้ายจะใช้ค่า mean ที่ได้เพื่อการพิจารณาจัดข้อมูลเข้ากลุ่ม ผลการทดลองพบว่าวิธีการจัดกลุ่มข้อมูลตามความหนาแน่น สามารถให้ผลการ

จัดกลุ่มที่ดี โดยมีค่า sum of squared error รวมในทุกกลุ่มต่ำกว่าวิธีจัดกลุ่มข้อมูลที่ใช้ข้อมูลทุกตัวเพื่อคำนวณค่า mean

ข้อเสนอแนะ

การทดลองในโครงการวิจัยนี้ ได้ทดสอบประสิทธิภาพของเทคนิคการจัดกลุ่มข้อมูลกับข้อมูลขนาดเล็ก (14 เรคคอร์ด) และข้อมูลขนาดกลาง (81 และ 191 เรคคอร์ด) ถึงแม้ว่าผลการทดลองจะยืนยันว่าเทคนิค density-biased clustering และวิธีการจัดกลุ่มข้อมูลแบบ incremental ให้ผลลัพธ์ที่ดีกว่าการจัดกลุ่มข้อมูลตามปกติ แต่การนำซอฟต์แวร์ที่พัฒนาขึ้นไปใช้งานจริง ยังต้องการการทดสอบกับข้อมูลจำนวนมากกว่านี้ และควรจะทดสอบเพิ่มเติมกับข้อมูลขนาดใหญ่ที่มีจำนวนมากกว่า 10,000 เรคคอร์ด

ซอฟต์แวร์ที่พัฒนาขึ้นนี้เน้นการจัดกลุ่มข้อมูลที่มีชนิดข้อมูลเป็นข้อความ (nominal, categorical) การจัดกลุ่มข้อมูลที่แอททริบิวต์มีค่าเป็นจำนวนเลข สามารถทำได้โดยปรับปรุงซอฟต์แวร์นี้เพียงเล็กน้อย แต่ถ้าค่าตัวเลขในแอททริบิวต์มีช่วงของค่าที่เป็นไปได้กว้างมาก เช่นในกรณีแอททริบิวต์อายุ ที่มีค่าอยู่ระหว่าง 0-120 ควรเพิ่มฟังก์ชัน normalization เช่นใช้เทคนิค max-min normalization หรือ z-score standardization เพื่อปรับลดช่วงกว้างของค่าสูงสุด-ต่ำสุด ทั้งนี้เพื่อลด bias ของการคำนวณค่า mean ของกลุ่มข้อมูล

บรรณานุกรม

M. R. Anderberg. Cluster Analysis for Applications. Academic Press, 1973.

P.S. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998.

G. Carpenter and S. Grossberg. ART3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. Neural Network, 3:129-152, 1990.

M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. SSD'95, 1995.

D. Fisher. Knowledge acquisition via incremental conceptual clustering. Machine Learning, 2:139-172, 1987.

S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In Proc. SIGMOD, 1998.

J. Han and M. Kamber. Data mining: Concepts and techniques. Morgan Kaufmann, 2001.

J.A. Hartigan. Clustering Algorithms. John Wiley and Sons, 1975.

Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Mining and Knowledge Discovery, 2:283-304, 1998.

A. K. Jain and R. C. Dubes. Algorithms for Clustering Data. Prentice Hall, 1988.

A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. ACM Computing Survey, 31(3):264-323, 1999.

L. Kaufman and P. J. Rousseeuw. Finding Groups in Data: an Introduction to Cluster Analysis. John Wiley and Sons, 1990.

J.MacQueen. Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symp. Math. Stat. Prob., 281-297, 1967.

P. Michaud. Clustering techniques. Future Generation Computer systems, 13, 1997.

R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. VLDB, 1994.

C.R.Palmer and C. Faloutsos. Density biased sampling: An improved method for data mining and clustering. In Proc.SIGMOD, 2000.

J.R. Slagle, C.L. Chang, and S.R. Heller. A clustering and data-reorganizing algorithm. IEEE Trans. Syst. Man Cybern., 5:125-128, 1975.

T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH : an efficient data clustering method for very large databases. In Proc. SIGMOD, 1996.

ภาคผนวก

ภาคผนวก ก

รหัสต้นฉบับของโปรแกรมทำเหมืองข้อมูลแบบจัดกลุ่ม

```
/* ============  Data Clustering ========= */
%
% To run the clustering program, call this procedure:
%                              dens_clust_menu.
%
% To run the merge clustering program, call this procedure:
%                              merge_clust_menu.
%-------------------------------------

:-dynamic amountItem/1,instance/3,attribute/2,columnN/1 ,no_rec/2,c/1.

%-------------------Find density ----------
:- dynamic instanceR/3,attributeR/2, dens_list/1,dens_rec/1.

:-dynamic cluster/2,description/2.

comp(E1,E2,V):- (E1==E2->V=1; V=0).

mycompare(L1,L2,CL):- maplist(comp,L1,L2,CL).

%test 2 instances, M=number of match
similar(I1,I2,M,V):- instance(I1,_,L1),
                     instance(I2,_,L2),
                     mycompare(L1,L2,VL), %compare 2 instances
                     sumlist(VL,SumV),
                     (SumV>=M ->V=1 ; V=0  ).

all_dens(IL1,IL2,M,L):-
       findall((X,Y,V),(member(X,IL1),member(Y,IL2),similar(X,Y,M,V)),L).

each_dens(L,I,I-Dens):- findall(V,(member((I,_,V),L)),VL),
                        amountItem(Len),
                        sumlist(VL,SumDens),
                        Dens is SumDens/Len .

% assert(dens_list([1-1,2-1,...]))
% +No_of_att_match,-assert(dens_list)
all_inst(M):- findall(X,instance(X,_,_),AllAttr),
              length(AllAttr,Len),
              numlist(1,Len,L1),
              all_dens(L1,L1,M,L),
              maplist(each_dens(L),L1,DL),
              retractall(dens_list(_)),
              assert(dens_list(DL)),!.

maindens(M):- all_inst(M),
              listing(dens_list).

all_rec_dens(M,D):-  retractall(dens_rec(_)),
                     ((M==0;D==0)-> ( writeln(m_______+M),
                                      findall(X,instance(X,_,_),AllRec))
                                      ;
                                      (maindens(M),dens_list(L),
                                      findall(X,(member(X-D1,L),D1>=D),AllRec))),
                                      writeln(allRec_________+AllRec),
                                      assert(dens_rec(AllRec)),!.

%-------------------PREPROCESS---------------
% pre.pl
%
create_attr([]):-    H=class,
                     attribute(H,R1),
                     assert(attributeR(H,R1)),!.
create_attr([H|T]):-
                     attribute(H,R1),
                     assert(attributeR(H,R1)),!,
                     create_attr(T).
```

```
choose_sampling(Per,C,A):-
        write('Per,Type,AttrList'+[Per,C,A]),
        sampling(C,Per,A),
        writeln(samplingA+A).

init:-
        retractall(c(_)),
        findall(X,(instance(X,_,_)),AttL),
        length(AttL,Len),
        numlist(1,Len,ColList),
        assert(c(ColList)),
        retractall(amountItem(_)),
        retractall(instanceR(_,_,_)),
        retractall(attributeR(_,_)),
        retractall(no_rec(_,_)),
        assert(amountItem(Len)),
        !,true.

%random without replacement- L1 for temp List
%rand1(+100,+30,+[],-Res).
rand(1,_,Nsel,L1,[]):-
                    length(L1,Len),
                     Len is Nsel,!.

rand(1,Nall,Nsel,L1,L2):-
              dens_rec(DensRec),
              H is random(Nall-1)+1,%shift to 1...Nall
              ( (memberchk(H,L1); not(memberchk(H,DensRec)) )
                      -> rand(1,Nall,Nsel,L1,L2);
                         (  L2=[H|T],
                            L=[H|L1],
                            rand(1,Nall,Nsel,L,T)
                            )
               ).

%random with replacement -L1 for temp List
%rand2(+100,+30,+[],-Res).
rand(2,_,Nsel,L1,[]):-
                    length(L1,Len),
                     Len is Nsel,!.

rand(2,Nall,Nsel,L1,[H|T]):-
             dens_rec(DensRec),
             H1 is random(Nall-1)+1,  %<<<< here density
             (memberchk(H1,DensRec)->(H=H1,rand(2,Nall,Nsel,[H|L1],T));
                                     rand(2,Nall,Nsel,L1,[H|T]) ).

%sampling(+Type,+Per,+Attribute)
%quit if NoSampling < len of Density List
%
%sampling(+Type,+Per,+Attribute)

sampling(C,Per,A):-
        amountItem(N),
        NoSel is round( (Per/100)*N),
        writeln(no_____________sel+NoSel),
        dens_rec(Rec),
        length(Rec,DensLen),
        assert(no_rec(NoSel,DensLen)),!,
        ( NoSel > DensLen ->
              (NumSel=DensLen,
              format('~n--densRec size<sampling size->
                            choose all density list=-a records:-n',
                     [DensLen])) ;
              NumSel=NoSel),
              (Per==100->(dens_rec(DensRec),LS=DensRec);
```

```
            rand(C,N,NumSel,[],LS)    ),
        create_rec(0,LS,A).

%create sampling rec
%(+0,+ListofRandkey,+Attribute)

create_rec(_,[],_):-true.
create_rec(N,[H|T],A):-
        instance(H,R1,R2),
        include(filter(A),R2,R22),
        N1 is N+1,
        assert(instanceR(N1-H,R1,R22)),!,
        create_rec(N1,T,A).

%filter(+AttrList,+Element)
% true or false -- filter for selected attributes
filter([],_):-false.
filter([H|_],(H=_)):-true,!.
filter([H|T],(M=V)):-M\==H,
                        filter(T,(M=V)). %<<<<<<< HERE density

%TLL=[[outlook+sunny+3, outlook+overcast+1, outlook+rainy+3],...]
%tally(-TLL)

tally(TLL):-
        findall(A+VL,attributeR(A,VL),L),
        maplist(map,L,LL),
        tallyAtt(LL,TLL).

tallyAtt(LL,TLL):-maplist(tallyEach,LL,TLL).

tallyEach(L,TL):-maplist(finda,L,TL).

finda(A+V,(A+V+N)):-
        findall(A+V,( instanceR(_,class=C,L),
                        (member(A=V,L);(A=class,V=C))),
                Res),
        length(Res,N).

map(A+VL,EL):- maplist(add(A),VL,EL).

add(A,B,A+B).

% called from MENU GUI.

mainp(D1,ParaDens1,K,Fout):-
        init,
        reconsult(D1),
        Per=100, S=1,
        findall(X,(attribute(X,_),X\=class), AL),
        term_to_atom(ParaDL,ParaDens1),
        [M,D]=ParaDL,
        all_rec_dens(M,D),
        create_attr(AL),
        choose_sampling(Per,S,AL),
        tally(TLL),writeln(TLL),
        writeln(end+main),
        no_rec(Want,Actual),
        tell('out.pl'),  % save densed instance --extra file
        format('~n%Density Parameter=[~a,~a]
                        Sampling[Percent,Type]=[~a,~a]~n',[M,D,Per,S]),
        format('~n%Want ~a records, but has ~a records~n',[Want,Actual]),
        (attributeR(X,Y),
         write(attribute(X,Y)),
         writeln('.'),
         fail
         ;true),
```

```
        (instanceR(N4-_,K4,L4),
          write(instance(N4,K4,L4)),
          writeln('.'),
          fail
          ;
          true),
        !,told,  %   InFile='out.pl',
        format('~ncaling ...cluster precess....~n'),
        tell(Fout),
        format('%--file:~w~n% output of clustering result',[Fout]),
        format('%density parameter=~w ,No of cluster= ~w',[ParaDens1,K]),
        findall(_,(attributeR(X,Y),format('~nattribute(~w,~w).',[X,Y])),_),
        nl,
        mainClust(K,AllCluster),
        format('~n~nallCluster(~w).~ndescription(''~w'',~w).~n',
                    [AllCluster,Fout,K]),
        told.

%--------------------MENU---------------------

dens_clust_menu:-
      new(Dialog,dialog('Density Biased Clustering')),
      send_list(Dialog, append,
            [ new(D1, text_item(input_datafile,'filename')),
              new(Dens, text_item(density_parameter,'[3,0.143]')),
              new(Per, int_item('number_cluster', low := 2, high := 80)),
              new(D2, text_item(output_file,'all_cluster1.pl')),
              button(cancel, message(Dialog, destroy)),
              button(enter, and(message(@prolog,mainp,
                          D1?selection,
                          Dens?selection,
                          Per?selection,
                          D2?selection
                         ),
              message(Dialog, destroy)))  % enter&destroy

            ]),
      send(Dialog, default_button, enter),
      send(Dialog, open).

%   working-- for discrete attributes
%+Kcluter,-AllClust

makeInitCluster(K,AllClust):- initClust(K,1,AllClust).

%+Kcluster,+L0startK,-AllClust
%4=Kcluster,1=start,[1*center,2*center]

initClust(K,L0,[]):-L0>K ,! .

initClust(K,L0,[L0*L|T]):-  instanceR(L0~_,_,L),
                            L1 is L0+1,
                            initClust(K,L1,T).

%+recNum,+clustNo*[allAttri],-freqMatch)
% 3      , 2*[...]            , 2*15 )

freq(X,N*Y,N*F) :- instanceR(X-_,_,L1),
                   intersection(L1,Y,I),
                   length(I,F).

cvalue(_*V,V).

cmax(L,A*V):- maplist(cvalue,L,L2),
              max_list(L2,V),
              member(A*V,L),!.
```

```
%+AllClust,+MaxInstanceNO,+StartAt,-AllFormatedPointList).
% AllClust,14            ,1    ,[1-15-2,2-11-1,...point-maxFreq-clustNo])

assignPoint(_,U,M,[]):-M>U,!.

assignPoint(AllClust,U,M,[M-V-A|T]):-
              maplist(freq(M),AllClust,Res),
              cmax(Res,A*V),
              M1 is M+1,
              assignPoint(AllClust,U,M1,T).

%+Kclusters,+StartAt,+AllFormatedPoint,-AllCluster
%2,          1,       AllPoint,[1*[outlook=sunny,temp=hot,..],2*[..]]   )

reComputeCenter(K,S,_Allpoint,[]):- S>K,!.

reComputeCenter(K,S,AllPoint,[S*NewCenter|T]):-
       findall(P,member(P-_-S,AllPoint),Z),%Z is AllP
       allPointAtAllAttr(Z,NewCenter),
       S1 is S+1,
       reComputeCenter(K,S1,AllPoint,T).

%+AllPoints,-AllClust
%[1,3,...],[1*[outlook=sunny,temp=hot,...],2*[...]]   )

allPointAtAllAttr(AllP,NewClusters) :-
       findall(AttName,(attribute(AttName,_),AttName\==class),AttNameL),
                                          % find all AttName
       maplist(allPoint(AllP),AttNameL,NewClusters).

%+AllPoints,+AttrName,-Attr=AttrVal
%[1,3,4],outlook,outlook=sunny

allPoint(AllP,Att,A):-
       findall(Att=V, (instanceR(X-_,_,K),member(X,AllP),
                          member(Att=V,K)) , Z),
       maxFreq(Z,A*V).

%+List,-MaxElement*Freq
%[a,a,b],a*2.

maxFreq(L,A*V):-
       findall(X*C,(member(X,L),count(X,L,C)),Z),
       cmax(Z,A*V).

write_cluster_member(K,AllPoint,NewClust):-
       numlist(1,K,KList),
       findall(Z-X,member(X-_-Z,AllPoint),R1),
       maplist(filter1,NewClust),
       maplist(filter11(R1),KList,_).

filter1(N*L):-format('~ncluster(~w,~w).',[N,L]).

filter11(L,X,Res):-
       findall(Y,member(X-Y,L),Res),
       format('~ncluster_member(~w,~w).',[X,Res]).
```

```
%+Kcluster,+AllPoints,+AllClust
%2,       [1,3,4,5], OldClust

repeatCompute(K,AllPoint,OldClust,Ret) :-
              reComputeCenter(K,1,AllPoint,NewClust),
              ( OldClust==NewClust -> (
                              format('~nallPoint(~w).~n',[AllPoint]),
                              write_cluster_member(K,AllPoint,NewClust),
                              Ret=NewClust);% exit when Old=New
                      (   %% write New Cluster
                          dens_rec(Rec),length(Rec,LastItemNo),
                          assignPoint(NewClust,LastItemNo,1,AllPoint2),
                          % assign points to new clusters
                          %writeln(allNewPoint-AllPoint2),
                          repeatCompute(K,AllPoint2,NewClust,Ret)
                      )
               ).

%+Element,+List,-Count
%a,[a,b,a,a],3
count(_,[],0).
count(H,[X|T],C):-    H\==X,!,
                      count(H,T,C).

count(H,[H|T],C):-    count(H,T,C1),!,
                      C is 1+ C1.



%--------- main----
% +K cluster, -Ret ReturnedCluster.

mainClust(K,Ret):-
       findall(X,instanceR(X-_,_,_),InsL),
       length(InsL,Len1),
       ( Len1<K ->
              format('~n% not_working Instance(~w)< K(~w)~n',[Len1,K]) ;
              (
                     makeInitCluster(K,AllClust),
                     assignPoint(AllClust,Len1,1,AllPoint),
                     OldClust=AllClust,
                     repeatCompute(K,AllPoint,OldClust,Ret)
               )
       ).

%------------ MERGE CLUSTER MENU ----
merge_clust_menu:-
       new(Dialog,dialog('Merge Clusters')),
       send_list(Dialog, append,
       [ new(D1, text_item(file1,'all_cluster1.pl')),
         new(D2, text_item(file2,'all_cluster2.pl')),
         new(D3, text_item(output_file,'all_cluster.pl')),
         new(K, int_item(numberOfCluster, low := 2, high := 20)),
         button(cancel, message(Dialog, destroy)),
         button(enter, and(message(@prolog,mergeClust,
                                D1?selection,
                                D2?selection,
                                D3?selection,
                                K?selection
                               ),
         message(Dialog, destroy)))  % enter&erase
  ]),
  send(Dialog, default_button, enter),
  send(Dialog, open).

%%     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%+inputfile,+inputfile

mergeClust(D1,D2,Fout,K):-
        format('~nmerging...from~w ~w~n',[D1,D2]),
        tell('tmp.pl'),
        format('%file tmp.pl~n:-dynamic attribute/2, instance/3.'),
        consult(D1),description(_,K1),
        findall(_,(attribute(X,Z),
                        format('~nattribute(~w,~w).',[X,Z])),_),
        findall(_,(cluster(X,Z),
                        format('~ninstance(~w,class=yes,~w).',[X,Z])),_),
        consult(D2),
        description(_,K2),nl,
        KLast is K1+K2,
        findall(_,(cluster(X,Z),
        N is X+K1,
        format('~ninstance(~w,class=no,~w).',[N,Z])),_),
        told,
        mainp('tmp.pl','[0,0]',K,'x.pl').


% ========== End of Program ===========
```

ภาคผนวก ข

ผลงานวิจัยที่ได้รับการตีพิมพ์เผยแพร่

- K. Kerdprasop, N. Kerdprasop and P. Sattayatham (2005). Weighted k-means for density-biased clustering. *Lecture Notes in Computer Science*, Volume 3589 Data Warehousing and Knowledge Discovery (DaWak), August, pp.488-497.

- N. Kerdprasop and K. Kerdprasop (2005). Multiple principal component analyses and projective clustering. *Proceedings of 16th International Workshop on Database and Expert Systems Applications (DEXA)*, Copenhagen, Denmark, August 22-26, pp.1132-1136.

- K. Kerdprasop and N. Kerdprasop (2009). Knowledge discovery from databases with higher-order predicates and logic programming. *Proceedings of the 2nd PSU Phuket Conference*, Phuket, Thailand, 18-20 November 2009.

Lecture Notes in Computer Science, Volume 3589 Data Warehousing and Knowledge
(DaWak), August 2005, pp

WEIGHTED K-MEANS FOR DENSITY-BIASED CLUSTERING

Kittisak Kerdprasop[1], Nittaya Kerdprasop[1], and Pairote Sattayatham[2]

[1]Data Engineering and Knowledge Discovery (DEKD) Research Unit,
School of Computer Engineering, Suranaree University of Technology,
Nakhon Ratchasima, Thailand

[2]School of Mathematics, Suranaree University of Technology
Nakhon Ratchasima, Thailand
kerdpras@sut.ac.th

ABSTRACT

Clustering is a task of grouping data based on similarity. A popular k-means algorithm groups data by firstly assigning all data points to the closest clusters, then determining the cluster means. The algorithm repeats these two steps until it has converged. We propose a variation called weighted k-means to improve the clustering scalability. To speed up the clustering process, we develop the reservoir-biased sampling as an efficient data reduction technique since it performs a single scan over a data set. Our algorithm has been designed to group data of mixture models. We present an experimental evaluation of the proposed method.

1. INTRODUCTION

Clustering is the automatic grouping of data based on similarity. There exists a large number of clustering techniques, but the most classical and popular one is the k-means algorithm [1]. Given a data set containing n objects, k-means partitions these objects into k groups. Each group is represented by the centroid of the cluster. Once cluster representatives are selected, data objects are assigned to the nearest centers. The algorithm iteratively selects new better representatives and reassigns data objects until no change is made. At this point the algorithm is said to converge. Even though k-means is an effective clustering algorithm, it can sometimes converge to a local optimum. Many methods [2,3,4,5] have been developed to extend the k-means with the common objective of avoiding converging to a bad local optimum. Some methods [6,7,8] search for the best initialization because k-means is known to be sensitive to initial point selection. Other research [9] seeks for the global optimum, at the cost of computation. These researches try to solve the problem of sub-optimal clustering and estimation the appropriate number of clusters [10,11].

Another difficulty of clustering with k-means is that it fails to identify clusters with large variation in sizes since original large clusters tend to be split. Clustering algorithms, such as DBSCAN [12] and CURE [13], have been developed to overcome this kind of difficulty. DBSCAN associates a data point with its density obtained by counting the number of points in a region of radius ε. The algorithm discovers clusters by connecting regions with sufficient high density, a *MinPts* threshold. DBSCAN works well in spatial clustering, but it is sensitive to the selection of ε and *MinPts* and it fails to efficiently discover clusters with highly different densities. CURE algorithm represents a cluster by a set of points, instead of a single representative. Once the representative points are chosen, the algorithm then shrinks these points toward the centroid of the cluster according to a shrinking factor. CURE is an iterative hierarchical-based clustering that works well with discovering cluster of different sizes, but it is sensitive to the selection of representatives and shrinking factor. Moreover, with very large data set, these algorithms degrade considerably.

When clustering massive data set, data reduction is an effective technique to speed up the algorithm. Sampling [14,15,16] is a powerful data reduction paradigm to remedy the inherent complexity of clustering. Uniform random sampling in which every data point has the same probability of being selected has been used extensively in data mining and databases [17,18,19,20]. In the case of data sets with large variation in cluster

sizes, density biased sampling [21,22,23] tends to be a better scheme. In density biased sampling, the probability that a data point will be included in the sample is varied by the density of a cluster.

Recent researches [21,22,23] propose several techniques to density biased sampling. Our work also follows this path with a step further on extending the k-means algorithm to work with a weighted sample. We propose an algorithm on density biased sampling based on the reservoir technique and a weighted k-means algorithm to cluster a data sample augmented with weights. The proposed algorithms are explained in Sections 2 and 3, respectively. We present the experimental results in Section 4. The conclusion and our future work are discussed in Section 5.

2. Data Reduction Biased by Density

On scalable popular and successful clustering methods such as k-means to work against large data sets, many algorithms like BIRCH [24] and CLARANS [14] employ the sampling technique to minimize data sets. In BIRCH, a CF-tree structure is built after an initial random sampling step. The CF-tree is used as a summarized data structure with statistical representations of space regions stored on leaf nodes. After the phase of CF-tree building, any clustering algorithm can be applied to the leaf nodes. CLARANS also uses uniform sampling to derive initial representative objects for the clusters.

The sampling technique used in these algorithms is uniform random sampling, which assigns every object the same probability of being included in the sample. But many data sets in real life do not follow the uniform distribution scheme. It instead seems to follow the Zipf's distribution [25], for instance, income and population distribution. In these data sets, some areas such as large metropolitan area have much higher population density than the small cities. If all the populations have equal opportunity of being selected as a representative, sparse areas may be missed and not be included in the sample.

2.1 Density-Biased Sampling

Density biased sampling [21] is a sampling technique that takes into account the different sizes of the groups. Small groups or sparse regions are assigned higher probability to be included in the sample than the large groups or dense regions. By biasing the sampling process, small clusters will not be missed or overlooked as outliers.

Recent advancement on clustering very large data sets in which summarized data structure is even too big to fit into main memory, sampling is independently applied to the data set prior to the subsequent clustering phase. Palmer and Faloutsos [21] develop a non-uniform sampling method for clusters that differ very much in size and density. Their method is a generalization of uniform random sampling in that every group of data sets can be assigned different probability of being drawn. When sampling is biased by group density, smaller groups are oversampling, whereas larger groups are under- sampling. Since clusters are not known a priori, Palmer and Faloutsos combine the phase of density information extraction with the biased sampling phase using the hash-based approach. They argue that the inherent collision problem of any hash-based approach will not dramatically degrade the sample.

Nevertheless, their method is significantly affected by noise due to the tendency of oversampling noisy area. Our approach adopts the reservoir technique to eliminate the collision problem of hash-based approach and it is independent on the assumption regarding cluster distribution to avoid the impact of noise.

2.2 Density-Biased Reservoir Sampling

We propose a novel approach of adapting reservoir technique [26,27] to perform a density biased sampling on large data sets. Our algorithm can obtain a desired sample through a single data set scan. The proposed method is simpler and requires less resource than the hash-based method [21].

A reservoir-sampling algorithm [26,27] is a simple, unbiased random sampling algorithm for drawing a sample of size n without replacement from a population of size N ($N \geq n$). Vitter [26] has developed a one-pass

reservoir-sampling algorithm when the population size (N) is unknown and cannot be determined efficiently. The term "reservoir" defines a storage area j ($j \geq n$, but mostly $j = n$) to store the potential candidates of the sample. The j reservoirs is initialized to store the first j records of the file, that is, all areas of the reservoir pool are initially filled up. Then the algorithm starts scanning the remaining part of the file with a randomly skipping step. The random picked record is evaluated whether to replace the existing one in the reservoir pool. If it passes the test, the position in the reservoir is also randomly selected. The process stops when the end of file has been reached and the records in the reservoir form a simple random sample of the population. The general procedure of reservoir-sampling algorithm [27,28] is given in Figure 1.

The time complexity of the algorithm is shown [26,27] to be O(n (1+ log(N/n)). In the reservoir-sampling algorithm, each record of the file is assigned a uniform (0,1) random number. When the reservoir is needed to be updated, each record in the reservoir has the same chance to be replaced by the new record.

Algorithm Reservoir sampling
Input: a sequential file of N population
Output: a random sample of size n ($n \leq N$)

1) Initialize the reservoir $X_1, \ldots, X_n$ to be the first n records of the file
2) Initialize W to be the largest value in a sample of size n from the uniform distribution on the interval (0, 1)
3) While not eof do
4) Generate the random variate S to denote the number of records to be skipped over before a new record can enter the reservoir
5) If (not eof) Then Search for the next potential record to be in the reservoir
6) Else return $X_1, \ldots, X_n$
7) Update X and W

Fig. 1. Reservoir-sampling algorithm

Our sampling algorithm generalizes the reservoir scheme for the case of data with different density distribution. In our proposed method, the initial step of partitioning data into groups resembles that of Palmer and Faloutsos [21]. But our subsequent steps are not based on hashing scheme in order to avoid the effect of noise and collision problems.

After the initial step of dividing the data space into bins of equal size, the informa- tion of the first n groups are put into the n reservoirs residing in main memory (see Figure 2a). The collected information includes the number of points in each group and the id of the group.
The algorithm performs a single scan on a data set in a random manner controlled by a random variate S with the distribution W. The density biasing (step 7 in Figure 3) is achieved through the consideration of two consecutive data groups. If the density difference of the two data groups is above some threshold δ (i.e., detecting cluster edge) or the sum of density on both groups is above the threshold value ε (i.e., avoiding noisy cases), then the denser group is a candidate to be included in a sample. This new candidate is put into a reservoir pool at a random position (the reservoir update is pictorially shown in Figure 2b). The density-biased sampling proceeds until the skipping variate S reaches the end of the data groups.

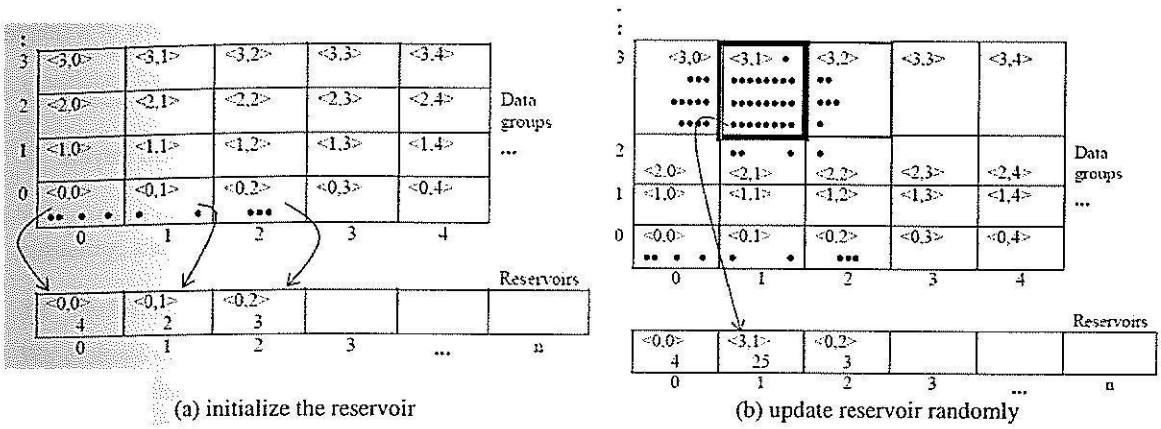


Fig. 2. Density biasing in a reservoir scheme

Algorithm Density-biased reservoir sampling

Input: a data set of N objects

Output: a density-biased sample of size n ($n \leq N$) associated with weight w

1) Partition data into g groups (with group-id $1,2,\ldots, g$), $g \geq n$

2) Initialize the reservoir $X_1, \ldots, X_n$ to be the first n <group-id, density>-pairs of the data groups

3) Set $W \leftarrow \exp(\log(\text{random}()) / n)$ // initialize W that will be used in the // generation step of random variate S

4) Set $S \leftarrow \lfloor \log(\text{random}()) / \log(1-W) \rfloor$

5) While $S < g$ do

6) Read data groups g_S and g_{S+1} // read two consecutive data groups

7) If ($\|\text{density}(g_S) - \text{density}(g_{S+1})\| > \delta$) OR ($(\text{density}(g_S) + \text{density}(g_{S+1})) > \varepsilon$)

// δ and ε are predefined density threshold values

Then $X_{1+\lfloor n * \text{random}() \rfloor} \leftarrow$ <group-id, density> of maximum density$\{g_S, g_{S+1}\}$

// randomized the reservoir area to be updated

8) $W \leftarrow W * \exp(\log(\text{random}()) / n)$ // update W for the skipping process

9) $S \leftarrow \lfloor \log(\text{random}()) / \log(1-W) \rfloor$ // generate S to denote the number of // groups to be skipped over

10) Return $X_1, \ldots, X_n$

Fig. 3. Density-biased reservoir sampling algorithm

3. Weighted K-Means Algorithm

The classical k-means algorithm [1] is a fast method to perform clustering. The algorithm consists of a simple re-estimation procedure as outlined in Figure 4.

Algorithm K-means

Input: a set of n data points, and the number of clusters (K)
Output: centroids of the K clusters

1) Initialize the K cluster centers
2) Repeat
   Assign each data point to its nearest cluster center
3) Recompute the cluster centers using the current cluster memberships
4) Until there is no further change in the assignment of the data points to new cluster centers

---

Fig. 4. K-means algorithm

The original n data points to be clustered are contained in the dataset $X = \{x_1, ..., x_n\}$. The k-means algorithm partitions n data points into K sets. The assignment of a data point x_i to its nearest cluster center c_j (step 2) is decided on the basis of the membership function, $m(c_j|x_i)$. The function returns either one of the $\{0,1\}$ values: $m(c_j|x_i) = 1$ if $j = argmin_k||x_i - c_k||^2$; it is zero, otherwise. In step 3, the new centroids of clusters can be computed from all data points x_i in the cluster. The objective function J of the algorithm is to minimize the sum of error squared, $J = \sum_{i=1:n} min_{j \in \{1..k\}} || x_i - c_j ||^2$.

In k-means algorithm, every data point has equal importance in locating the centroid of the cluster. This property does no longer hold in the case of density-biased sample clustering, for which each data point represents varied density in the original data. Therefore, the clustering algorithm has to consider a weight associated with each data point in the computation of cluster centers. The proposed extension to the k-means algorithm is called weighted k-means. Figure 5 outlines the algorithm.

Algorithm Weighted k-means

Input: a set of n data points obtained from the density-biased reservoir sampling, and the number of clusters (K)
Output: centroids of the K clusters

1) Initialize the K cluster centers
2) Repeat
   Assign each data point to its nearest cluster center according to the membership function,

$$m(c_j|x_i) = \frac{\| x_i - c_j \|^{-p-2}}{\sum_{j=1:k} \| x_i - c_j \|^{-p-2}}$$

3) For each center c_j, recompute the cluster center c_j using the current cluster memberships and weights,

$$c_j = \frac{\sum_{i=1:n} m(c_j|x_i)\, w(x_i)\, x_i}{\sum_{i=1:n} m(c_j|x_i)\, w(x_i)}$$

   where $w(x_i)$ is a weight associated with each data point
4) Until there is no reassignment of data points to new cluster centers

---

Fig. 5. Weighted k-means algorithm

The membership function in the weighted k-means algorithm resembles that of the k-harmonic means algorithm [5]. Zhang [5] also introduces the weight function, $w(x_i)$, in his algorithm to accelerate the recomputation of the new centroids in the next iteration. The weight function in our algorithm, however, is introduced for the different purpose. It represents the density of the original data points.

4. Experiments and Results

We perform two sets of experiments to test the quality of our sampling method, which is the step prior to clustering, and to measure the quality of the weighted k-means algorithm.

4.1 Performance of Density-Biased Reservoir Sampling

We evaluate the performance of the proposed reservoir-based density bias sampling method against the hash-based sampling method [21]. The efficiency regarding memory usage of our reservoir-based sampling method is obviously better than the hash-based method. In the hashing scheme, some amount of memory is needed to store the hashing table in addition to the memory required for storing the drawn sample. Thus, it requires twice the amount of memory comparative to those required by our method.

Effectiveness of the proposed sampling method is examined by measuring the quality of a sample with respect to the number of correctly found clusters. We run clustering using the k-means algorithm. We use a synthetic data generator to generate d-dimensional data sets having k clusters and N data points. We vary d from 2 to 5, k from 2 to 10, and N from 5,000 to 100,000.

The measurement *Number of Clusters found* (NC) is the metric defined in [21]. NC is calculated by comparing the distances of the cluster centers found by the clustering algorithm with the true cluster centers. We say that the cluster is found if the calculated distance is less than a predefined threshold (e.g., 0.001).

The results in Figure 6 show the NC when run clustering on various sample sizes with the presence of noise. The reported results are observed from the experiments using 3-dimensional data set having 7 clusters. One cluster contains 50,000 points and the other six clusters contain 500 points. The results obtained from other experiments on data sets with different dimensions, various number of clusters, and varied number of data points are conformed with the one presented in Figure 6, so we omit them for brevity. The experimental results reveal the efficiency of the biased reservoir method especially in the presence of noise.

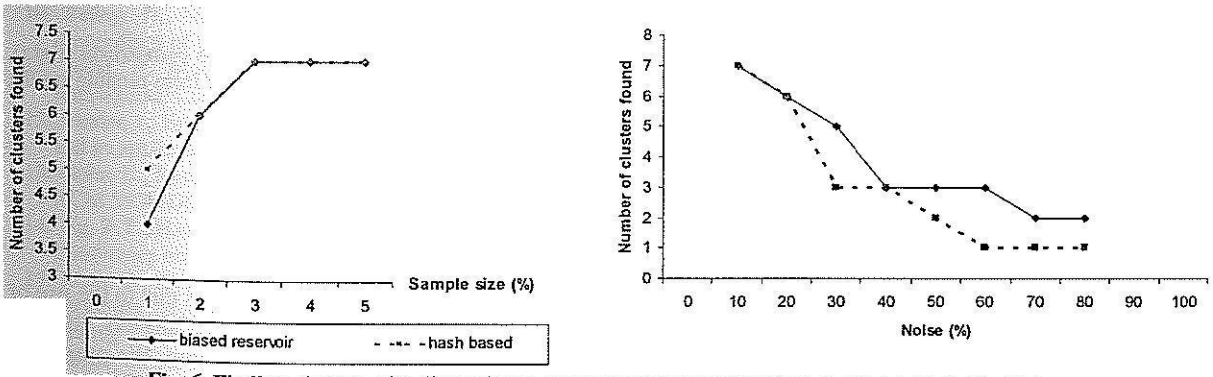


Fig. 6. Finding clusters of 3-dimensional data on various sample sizes, in the presence of noise

4.2 Performance of Weighted K-Means Algorithm

We evaluate the quality of the weighted k-means algorithm against the k-means algorithm by using the squared objective function. Lower value of a squared objective function reflects a better quality on clustering. The experiments perform on the syntactic data sets explained in Section 4.1. The initialization step randomly selects data points as initial cluster centroids. We also consider running time of both algorithms.

The performance evaluation as shown on top of Figure 7 is obtained from running k-means and weighted k-

means algorithms on 3-dimensional data sets of sizes varied from 5000, 10000, 20000, 35000, 55000, 75000, to 100000 data points. The number of clusters is set to be 10. The experiments are performed on the PC computer with CPU speed 800 MHz, memory 512 MB. Since all data points are used in weighted k-means algorithm, the weight function is set to be 1. The parameter p in the membership function is set to be 1.3.

The comparison on clustering quality and running time shown at the bottom of Figure 7 reveals the efficiency of running weighted k-means on density-biased sample. The experiments are performed on 10% sample of data with two methods of sampling: simple random sampling (RS) and density-biased reservoir sampling (DBS). The weight function of the weighted k-means algorithm is varied according to the density of the original data.

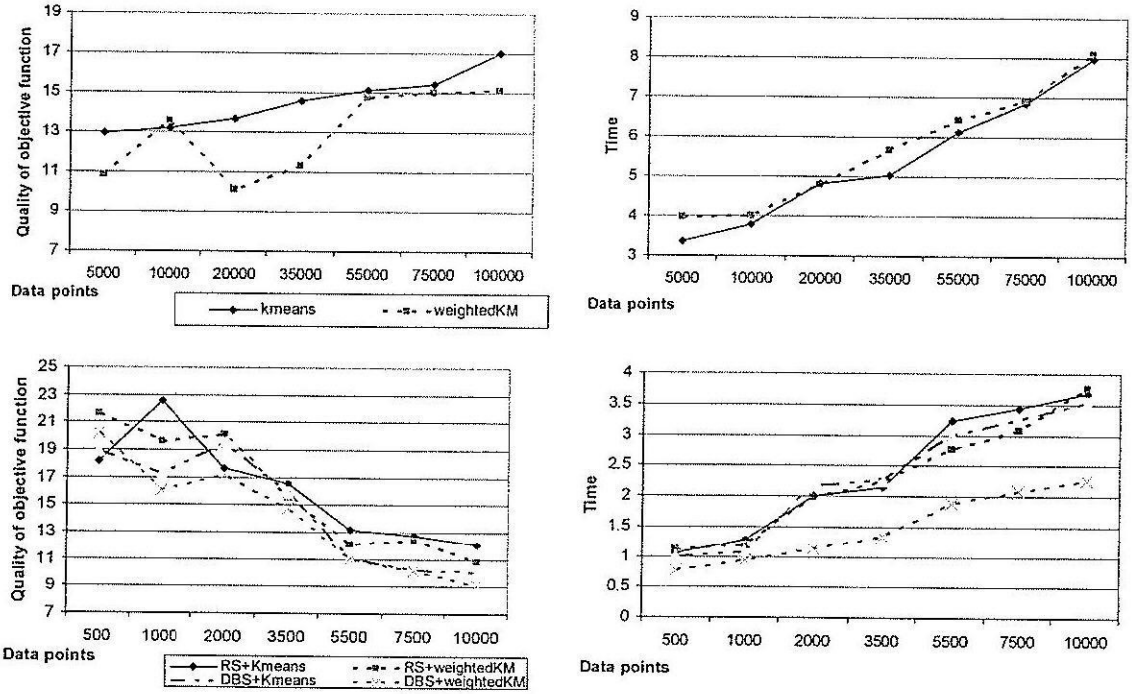


Fig. 7. Performance comparison of weighted k-means against k-means (left) and the running time comparison (right), results on top are experiments running on the whole data set while results at the bottom obtained from running on the sample data

5. Conclusions

The k-means is the simplest and most commonly used clustering algorithm. The simplicity is due to the use of squared error as the stopping criteria, which tends to work well with isolated and compact clusters. Its time complexity depends on the number of data points to be clustered and the number of iteration. We propose a variation of the k-means to better work with a large data set having much difference in cluster density. Our intuition idea is that to cope with massive data set, sampling should be the efficient data reduction method. Since the original data is assumed to be much varied in cluster sizes, density-biased sampling is an appropriate method to preserve the density.

We propose a density biased sampling technique based on the reservoir method. The inherent advantage of efficient memory usage in the reservoir scheme is adopted and extended with the additional capability of dealing with data that are much different in density distribution. The proposed technique is designed to lessen the effect of noise as it is the case in the hash-based approach. The experimental results have shown that the

proposed method is as good as the hash-based method in discovering correct number of clusters. Our method, moreover, is less sensitive to noisy data even when the percentage of noise is greater than 20.

We also develop the weighted k-means algorithm to better cluster a sample data biased by its density. The results demonstrate the efficiency of the algorithm. The evaluation of the proposed methods on real large databases and the consideration of outliers are our future work.

Acknowledgements

This research has been supported by grants from the Thailand Research Fund (TRF, MRG4780170), and the National Research Council. The Data Engineering and Knowledge Discovery Research Unit is fully supported by the research grants from Suranaree University of Technology.

References

1. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Vol.1. University of California Press (1967) 281-297
2. Hamerly, G., Elkan, C.: Alternatives to the k-means algorithm that find better clusterings. In Proceedings of 11th ACM CIKM International Conference on Information and Knowledge Management (2002) 600-607
3. Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum Press, New York (1981)
4. Pellog, D., Moore, A.: Accelerating exact k-means algorithms with geometric reasoning. In Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1999) 277-281
5. Zhang, B.: Generalized k-harmonic means - boosting in unsupervised learning. Technical Report HPL-2000-137. Hewlett-Packard Labs (2000)
6. Bradley, P.S., Fayyad, U.M.: Refining initial points for k-means clustering. In Proceedings of the 15th International Conference on Machine Learning (1998) 91-99
7. Meila, M., Heckerman, D.: An experimental comparison of model-based clustering methods. Machine Learning 42(2001) 9-29
8. Pena, J., Lozano, J., Larranaga, P.: An empirical comparison of four initialization methods for the k-means algorithm. Pattern Recognition Letters 20(1999) 1027-1040
9. Likas, A., Vlassis, N., Verbeek, J.: The global k-means clustering algorithm. Technical Report IAS-UVA-01-02. Computer Science Institute, University of Amsterdam, The Netherlands (2001)
10. Pelleg, D., Moore, A.: X-means: Extending k-means with efficient estimation of the number of clusters. In Proceedings of the 17th International Conference on Machine Learning (2000) 727-734
11. Sand, P., Moore, A.: Repairing faulty mixture models using density estimation. In Proceedings of the 18th International Conference on Machine Learning (2001)
12. Sander, J., Ester, M., Kriegel, H.-P., Xu, X.: Density-based clustering in spatial databases: The algorithm GDBSCAN and its application. Data Mining and Knowledge Discovery 2(1998) 169-194
13. Guha, S., Rastogi, R., Shim, K.: CURE: An efficient clustering algorithm for large databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data (1998) 73-84
14. Ng, R.T., Han, J.: Efficient and effective clustering methods for spatial data mining. In Proceedings of International Conference on Very Large Data Bases (1994) 144-155
15. Zhou, S., Zhou, A., Cao, J., Wen, J., Fan, Y., Hu., Y: Combining sampling technique with DBSCAN algorithm for clustering large spatial databases. In Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining (2000) 169-172
16. Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y.: C^2P: Clustering based on closest pairs. In Proceedings of International Conference on Very Large Data Bases (2001) 331-340
17. Singh, G., Rajagopalan, S., Lindsay, B.: Random sampling techniques for space efficient of large data sets. In Proceedings of ACM SIGMOD International Conference on Management of Data (1999)
18. Toivonen, H.: Sampling large databases for association rules. In Proceedings of International Conference on Very Large Data Bases (1996) 134-145
19. Thompson, S.K., Seber, G.A.F.: Adaptive Sampling. John Wiley & Sons, New York (1996)
20. Olken, F., Rotem, D.: Sampling from spatial databases. In Proceedings of International Conference on Data Engineering (1993) 199-208
21. Palmer, C., Faloutsos, C.: Density biased sampling: An improved method for data mining and clustering. In Proceedings of ACM SIGMOD International Conference on Management of Data (2000) 82-92

22. Nanoupoulos, A., Theodoridis, Y., Manolopoulos, Y.: An efficient and effective algorithm for density biased sampling. In Proceedings of 11th International Conference on Information and Knowledge Management (2002) 63-68
23. Kollios, G., Gunopulos, D., Koudas, N., Berchtold, S.: Efficient biased sampling for approximate clustering and outlier detection in large data sets. IEEE Transactions on Knowledge and Data Engineering 15(2003) 1-18
24. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. In Proceedings of ACM SIGMOD International Conference on Management of Data (1996) 103-114
25. Zipf, G.K.: Human Behavior and Principle of Least Effort: An Introduction to Human Ecology. Addison Wesley, Cambridge, MA (1949)
26. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software 11(1985) 37-57
27. Li, K.-H.: Reservoir-sampling algorithms of time complexity O(n(1 + log(N/n))). ACM Transactions on Mathematical Software 20(1994) 481-493
28. Devroye, L.: Non-Uniform Random Variate Generation. Springer-Verlag, New York (1986)

Proceedings of 16th International Workshop on Database and Expert Systems Applications (DEXA), Copenhagen, Denmark, August 22-26, pp.1132-1136.

MULTIPLE PRINCIPAL COMPONENT ANALYSIS AND PROJECTIVE CLUSTERING

Nittaya Kerdprasop and Kittisak Kerdprasop

Data Engineering and Knowledge Discovery (DEKD) Research Unit,
School of Computer Engineering, Suranaree University of Technology,
Nakhon Ratchasima, Thailand
nittaya@sut.ac.th

ABSTRACT

Projective clustering is a clustering technique for high dimensional data with the inherent sparsity of the data points. To overcome the unreliable measure of similarity among data points in high dimensions, all data points are projected to a lower dimensional subspace. Principal component analysis (PCA) is an efficient method to dimensionality reduction by projecting all points to a lower dimensional subspace so that the information loss is minimized. However, PCA does not handle well the situation that different clusters are formed in different subspaces. We propose a method of multiple principal component analysis for iteratively computing projective clusters. The objective function is designed to determine the subspace associated with each cluster. Some experiments have been carried out to show the effectiveness of the proposed method.

1. INTRODUCTION

Clustering is a widely used technique to discover homogeneous groups, or clusters, of data according to a certain similarity measure. Many algorithms have been designed [10] to compute a partition on full-dimensional data set. While these approaches work successfully on low-dimensional data sets, their efficiency decrease significantly in higher dimensional space [9, 12]. In high dimensional data, some dimensions tend to be redundant or irrelevant. Massive dimensions can confuse the clustering algorithms. It is also difficult to group similar data points in very high dimensions because the distance between any two data points becomes almost the same [5, 8]. The most difficult problem on clustering high dimensional data is that different clusters may exist in different subspaces of different dimensions [4].

A possible solution to these problems is to use dimension reduction or feature selection techniques. By means of dimension reduction, one first reduces the dimensions of the original data set by removing less important dimensions or by transforming the original data set into a lower dimensional space. The conventional clustering algorithms can then be applied to the new data set. However, an attempt to reduce dimensions of all data points results in significant information loss.

Recognizing the need for an efficient algorithm for clustering high dimensional data, the concept of subspace clustering or projective clustering has thus been proposed [1, 2, 3, 4]. The goal of projective clustering is to find clusters embedded in lower dimensional subspaces. It can minimize the information loss in the process of dimension reduction by projecting those high dimensional data points into different lower dimensional subspaces for different clusters.

Statistical methods such as Principal Component Analysis (PCA) [11] can effectively reduce the dimensionality of the original data by projecting all points on a subspace so that the information loss is minimized. Then, a standard clustering method can be used in this subspace. However, PCA does not work well when different subsets of data points embedded in different lower-dimensional subspaces. We, thus, propose the method to iteratively apply PCA aiming at transforming the original data set into various lower-dimensional subspaces. The subsequent steps compute a partitioning of data points into disjoint groups. We

briefly explain the concept of PCA in Section 2. The proposed method of multiple PCA and the partitional clustering steps are then presented in Section 3. The experimental results shown in Section 4 verify the efficiency of the proposed method. Finally, conclusion remarks are presented in Section 5.

2. Dimensionality reduction with PCA

Principal Component Analysis (PCA) [11,14], sometimes called the Karhunen-Loeve (KL) transformation [4], is a widely used method for reducing the number of dimensions of a data set. The goal of PCA is to find basis vectors for a subspace which maximizes the least square reconstruction error. Let $X= (x_1, ..., x_n)$ be a d-dimensional data matrix of points. PCA projects the correlated high-dimensional data onto a hyperplane. This mapping uses only the first few q nonzero eigenvalues and the corresponding eigenvectors of the covariance matrix F, $F = U\Lambda U^T$ where Λ is a matrix that includes the eigenvalues λ_i of F in its diagonal in decreasing order, and U is a matrix that includes the eigenvectors corresponding to the eigenvalues in its column. The vector $y = W^T(x_i)$ is a q-dimensional reduced representation of the observed vector x_i where the W weight matrix contains the q principal orthonormal axes in its column $W = U_q\Lambda_q^{1/2}$.

Tipping and Bishop [13] developed a method called probabilistic PCA (PPCA) to associate a proper probability model for PCA. The advantage of the PPCA model is that it can easily be extended to mixture model where data can be viewed as arising from several populations mixed in varying proportions. The entire data set is then modeled by a Gaussian with restricted covariance matrix:

$$p(x) = \frac{1}{(2\pi)^{d/2}\,|\det A|^{1/2}} \exp\left(-\frac{1}{2}(x-\bar{x})^T A^{-1}(x-\bar{x})\right),$$

where $A = \sigma^2 I + WW^T$ is the modified covariance matrix and I is the identity matrix. W is found as in the original PCA algorithm, and σ^2 is found by calculating the average of the variance in the discarded dimensions:

$$\sigma^2 = \frac{1}{d-q}\sum_{i=q+1}^{d}\lambda_i.$$

3. Multiple PCA with projective clustering

Our approach aims at performing clustering in low dimensional subspaces, instead of the original high dimensional space where clusters are not well-separated. The intuition idea is to reduce the dimensions using PCA. Since the embedded clusters may lie in different subspaces, the subspace initially obtained using PCA does not necessary coincide with the subspace spanned by the K cluster centers. Therefore, we propose to iteratively perform PCA and clustering on the reduced subspace until the convergence criteria has been reached. The algorithm can be defined as follows.

Algorithm Clustering with multiple PCA

Input: a set of data vectors $X = [x_1, ..., x_n]$ of d dimensions and the number of cluster (k)

Output: a set of cluster centers $C_\mu = [\mu_1, ..., \mu_k]$ and the relevant attributes

Steps:

1. Initialization: Refine the initial points for clustering (as proposed in [7]) on sample data, and center the data matrix X so that the value of each variable is subtracted for that variable.
2. Do the first dimension reduction using PCA to obtain the q-dimensional subspace, q<d.
3. While not convergence
   - 3.1 Run clustering algorithm on the q-dimensional subspace to obtain clusters.
   - 3.2 Use cluster membership to construct the k cluster centroids in the original space.
   - 3.3 Compute the span of K centroids using singular value decomposition.
   - 3.4 Apply PCA to obtain a new q-dimensional subspace.
4. Return a set of cluster centers associated with relevant attributes

We propose the method for iterative high-dimensional clustering on the basis of fuzzy k-means [6]. A prior probability of the cluster can be computed as:

$$\alpha_i = \frac{1}{n}\sum_{j=1}^{n}\gamma_{i,j}$$, where γ is the degree of membership.

The cluster centers are determined from

$$v_i^x = \frac{\sum_{k=1}^{n}(\gamma_{i,k})^m (x_k - W_i\langle y_{i,k}\rangle)}{\sum_{k=1}^{n}(\gamma_{i,k})^m}$$

where the expectation of the latent variables is $\langle y_{i,k}\rangle = M_i^{-1}W_i^T(x_k - v_i^x)$, the q×q matrix $M_i = \sigma_{i,x}^2 I + W_i^T W_i$ and the fuzzy weighting component $m = 2$. The new value of W_i can be computed from

$$\tilde{W}_i = F_i W_i (\sigma_{i,x}^2 I + M_i^{-1}W_i^T F_i W_i)^{-1}$$

where the covariance matrix F_i can be computed by

$$F_i = \frac{\sum_{k=1}^{n}(\gamma_{i,k})^m (x_k - v_i^x)(x_k - v_i^x)^T}{\sum_{k=1}^{n}(\gamma_{i,k})^m}$$

The new value of $\sigma_{i,x}^2$ is

$$\sigma_{i,x}^2 = \frac{1}{q}tr(F_i - F_i W_i M_i^{-1}\tilde{W}_i^T).$$

The fuzzy covariance matrix is

$$A_i = \sigma_{i,x}^2 I + \tilde{W}_i \tilde{W}_i^T.$$

The square distance measurement, D^2, for the fuzzy k-means is defined as the product of three terms: prior probability of the cluster, the distance between the k-th data point and the centroid v_i of cluster i, and the distance between the cluster prototype and the data in the subspace. The objective function, J, of the clustering is

$$J = \sum_{i=1}^{c}\sum_{k=1}^{n}(\gamma_{i,k})^m D^2 + \sum_{k=1}^{n}\lambda_i(\sum_{i=1}^{c}\gamma_{i,k} - 1).$$

4. Experimental evaluation

We compare our proposed projective clustering (PKM) with the fuzzy k-means (FKM) and k-means (KM) algorithms. The experiments are performed on the Pentium IV 1.0 GHz machine with 512 MB of main memory. We generate the synthetic data sets of 50,000 points with varied dimensions up to 100. To assess the quality of a clustering algorithm we use the distance metric

$$\sqrt{\sum_{i=1}^{n}\min_{j\in\{1..k\}}\| x_i - c_j \|^2}.$$

The performance is evaluated on the clustering quality, the number of iteration toward convergence criteria, and the running time. The results are shown in Figures 1 through 3.

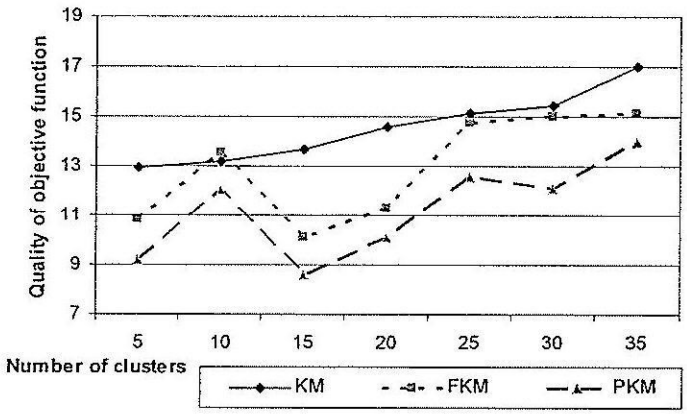


Figure 1. A comparison on clustering quality

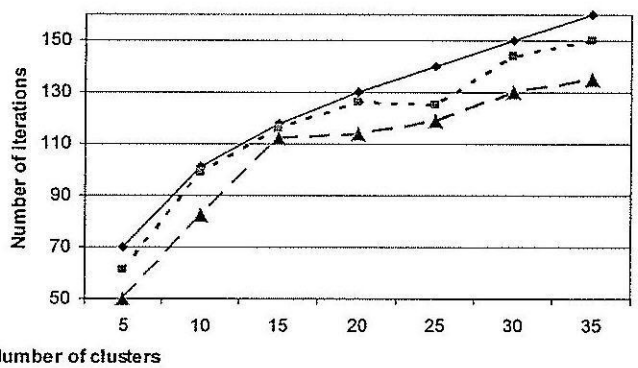


Figure 2. A comparison on number of iteration

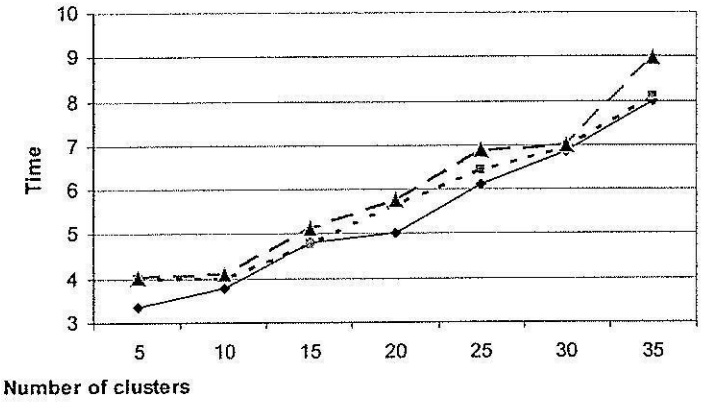


Figure 3. A comparison on running time

5. Conclusions

We have introduced a new method for clustering high dimensional data using the concept of projective clustering. The algorithm is based on the fuzzy partitional clustering algorithm. The key to the effectiveness of finding clusters on different subspaces is due to the power of PCA. The main justification of powerful dimension reduction is that PCA uses singular value decomposition (SVD) which gives the best low rank approximation to original data. We perform multiple PCA to project data into different subspaces for the effectiveness of discovering clusters embedded in different layers. The experimental results confirm the efficiency of our proposed method. Running experiments on real data is an essential step toward the practicality improvement of the method.

Acknowledgements

This research has been supported by grants from the Thailand Research Fund (TRF, MRG4780170), and the National Research Council. The Data Engineering and Knowledge Discovery Research Unit is fully supported by the research grants from Suranaree University of Technology.

References

[1] C. Aggarwal, J. Wolf, P. Yu, C. Procopiuc, and J. Park, "Fast algorithms for projected clustering", *Proceedings of ACM SIGMOD Int. Conf. on Management of Data*, 1999, pp. 61-72.

[2] C. Aggarwal and P. Yu, "Finding generalized projected clusters in high dimensional spaces", *Proceedings of ACM SIGMOD Int. Conf. on Management of Data*, 2000, pp.70-81.

[3] C. Aggarwal and P.Yu, "Redefining clustering for high-dimensional applications", *IEEE Transactions on Knowledge and Data Engineering*, 14(2), 2002, pp.210-225.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications", *Proceedings of ACM SIGMOD Int. Conf. on Management of Data*, 1998, pp.94-105.

[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is nearest neighbor meaningful?", *Proceedings of 7th Int. Conf. on Database Theory*, 1999, pp.217-235.

[6] J. Bezdek and J. Dunn, "Optimal fuzzy partitions: A heuristic for estimating the parameters in a mixture of normal distributions", *IEEE Transactions on Computers*, 1975, pp.835-838.

[7] P. Bradley and U. Fayyad, "Refining initial points for k-means clustering", *Proceedings of 15th Int. Conf. on Machine Learning*, 1988, pp.91-99.

[8] A. Hinneburg, C. Aggarwal, and D. Keim, "What is the nearest neighbor in high dimensional spaces?", *Proceedings of 26th Int. Conf. on Very Large Data Bases*, 2000, pp.506-515.

[9] A. Hinneburg and D. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high dimensional clustering", *Proceedings of 25th Int. Conf. on Very Large Data Bases*, 1999, pp.506-517.

[10] A. Jaine and R. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[11] I. Jolliffe, *Principal Component Analysis*, Springer Verlag, 1986.

[12] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: A review", *SIGKDD Explorations*, 6(1), 2004, pp.90-105.

[13] M. Tipping and C. Bishop, "Mixtures of probabilistic principal component analyses", *Neural Computation*, 11(2), 1999, pp.443-482.

[14] K. Yeung and W. Ruzzo, "Principal component analysis for clustering gene expression data", *Bioinformatics*, 17(9), 2001, pp.763-774.

Proceedings of the 2nd PSU Phuket Conference, Phuket, Thailand, 18-20 November 2009.

A KNOWLEDGE DISCOVERY FROM DATABASES WITH HIGHER-ORDER PREDICATES AND LOGIC PROGRAMMING

Kittisak Kerdprasop and Nittaya Kerdprasop

Data Engineering and Knowledge Discovery (DEKD) Research Unit,
School of Computer Engineering, Suranaree University of Technology,
Nakhon Ratchasima, Thailand
kerdpras@sut.ac.th

ABSTRACT

Modern organizations generate huge amount of data in electronic form and store in heterogeneous databases. These data are a valuable resource for automatic discovering of useful knowledge, known as knowledge discovery in databases – KDD or data mining, to support high-level decisions. KDD is the process of deriving new and useful knowledge from vast volumes of data and information stored as background knowledge. Derived knowledge may be patterns of data represented in summarized form, relationships among data represented as rules, or representatives of data subgroups represented by mean values. During the past decades there has been an increasing interest in devising database and machine learning technologies to automatically induce knowledge from stored data using imperative and object-oriented programming styles. In this paper we propose a KDD system based on a logical framework. The proposed system includes three major knowledge induction components: data classification, association mining, and data clustering. In order to derive useful knowledge efficiently from both underlying raw data and background knowledge, we consider employing the concepts of logic programming and higher-order predicates. Higher-order logic programming can greatly reduce the burden of programmers as it is a very high-level programming scheme suitable for the development of knowledge intensive tasks. We have shown in this paper tree-based classification, frequent pattern mining, and k-means clustering implemented with higher-order predicates using Prolog programming language. Our design and implementation of logic-based knowledge discovery system is intended to support higher-order and constraint mining that will be the next step of our research work.

KEYWORDS

Knowledge engineering, Data mining, Logic programming, Higher-order predicates

1. INTRODUCTION

Knowledge is a valuable asset to most organizations as a substantial source to enhance understanding of data relationships and support better decisions to increase organizational competency. Automatic knowledge acquisition can be achieved through the availability of the knowledge discovery system. The discovered knowledge facilitates expert decision support, data exploration and explanation, estimation of future trends, and prediction of future outcomes based on present data.

In this paper we present the design and implementation of a knowledge discovery system, called SUT-Miner, to be applicable to any domain that requires a knowledge-based decision support. A rapid prototyping of the proposed system is provided in a declarative programming style. The intuitive idea of our design is that for such a complicated knowledge-based system program coding should be done declaratively at a high level to alleviate the burden of programmers. The advantages of declarative style are thus the decrease in program development time and the increases in expressiveness of knowledge representation and efficiency of knowledge utilization.

A logic-based approach to the development of knowledge discovery system has long been an interesting research topic among data mining and machine learning researchers. For the classification task, FOIL [17] and PROGOL [12] were two examples of successful logic-based systems. Tree-based concept induction [10, 17]

and rule induction [12] are major approaches normally adopted for the classification task. Association mining task was mostly based on the well-known APRIORI algorithm [1]. WARMR system [4] upgraded APRIORI algorithm to discover frequent patterns. Its extension [5] was developed to discover frequent Datalog patterns and relational association rules. Data clustering based on logic programming and first-order logic was mainly an extension of k-means clustering algorithm [11]. K-prototypes [9] extended k-means clustering to work on first-order representation. KBG [2], TIC [3], COLA-2 [6], and RDBC [8] were all first-order clustering systems.

All logic-based knowledge discovery systems proposed in the literature considered a single task. Our work, on the contrary, is a proposal of a knowledge discovery system designed as an integrated environment storing a repertoire of tools for discovering various kinds of knowledge. The outline of this paper is as follows. Section 2 is preliminaries on three main knowledge discovery tasks: classification, association mining, and clustering. Section 3 presents the basic of logic programming concept and the notation of higher-order predicates. Section 4 discusses the conceptual design of SUT-Miner, which is our proposed logic-based knowledge discovery system. Section 5 demonstrates its implementation using Prolog language. Section 6 concludes the paper.

2. PRELIMINARIES

2.1 TREE-BASED CLASSIFICATION

Decision tree induction [16] is a popular method for inducing knowledge from data. Popularity is due to the fact that mining result in a form of decision tree is interpretability, which is more concern among practitioners than a sophisticated method but lack of understandability. A decision tree is a hierarchical structure with each node contains decision attribute and node branches corresponding to different attribute values of the decision node. The goal of building decision tree is to partition data with mixing classes down the tree until the leaf nodes contain pure class.

In order to build a decision tree, we need to choose the best attribute that contributes the most towards partitioning data to the purity groups. The metric to measure attribute's ability to partition data into pure class is *Info*, which is the number of bits required to encode a data mixture. To choose the best attribute, we have to calculate information gain, which is the yield we obtained from choosing that attribute. The information gain calculates yield on data set before splitting and after choosing attribute with two or more splits. The gain value of each candidate attribute is calculated. Then choose the maximum one to be the decision node. The process of data partitioning continues until the data subset has the same class label

2.2 ASSOCIATION MINING

Association mining is the discovery of relationships or correlations between items in a database. Let $I = \{i_1, i_2, i_3, ..., i_m\}$ be a set of m items and $DB = \{ C_1, C_2, C_3, ..., C_n\}$ be a database of n cases or observations and each case contains items in I. A *pattern* is a set of items that occur in a case. The number of items in a pattern is called the length of the pattern. To search for all valid patterns of length 1 up to m in large database is computational expensive. For a set I of m different items, the search space for all distinct patterns can be as huge as 2^m-1. To reduce the size of the search space, the *support* measurement has been introduced [1]. The function *support*(P) of a pattern P is defined as a number of cases in DB containing P. Thus, *support*(P) = $|\{T | T \in DB, P \subseteq T\}|$. A pattern P is called *frequent pattern* if the support value of P is not less than a predefined minimum support threshold *minS*. It is the *minS* constraints that help reducing the computational complexity of frequent pattern generation. The *minS* metric has an anti-monotone property and is applied as a basis for reducing search space of mining frequent patterns in algorithm Apriori [1].

2.3 K-MEANS CLUSTERING

Clustering refers to the iterative process of automatic grouping of data based on their similarity. There exist a large number of clustering techniques, but the most classical and popular one is the k-means algorithm [11]. Given a data set containing n objects, k-means partitions these objects into k groups. Each group is

represented by the centroid, or central point, of the cluster. Once cluster means or representatives are selected, data objects are assigned to the nearest centers. The algorithm iteratively selects new better representatives and reassigns data objects until the stable condition has been reached. The stable condition can be observed from cluster assigning that each data object does not change its cluster.

3. THE BASIC OF LOGIC PROGRAMMING AND HIGHER-ORDER PREDICATES

In logic programming, a clause is a disjunction of literals (atomic symbols or their negations) such as $p \vee q$ and $\neg p \vee r$. A statement is in clausal form if it is a conjunction of clauses such as $(p \vee q) \wedge (\neg p \vee r)$. Logic programming is a subset of first order logic in which clauses are restricted to Horn clauses. A Horn clause, named after the logician Alfred Horn [15], is a clause that contains at most one positive literal such as $\neg p \vee \neg q \vee r$. Horn clauses are widely used in logic programming because their satisfiability property can be solved by resolution algorithm (an inference method for checking whether the formula can be evaluated to true).

A Horn clause with no positive literal, such as $\neg p \vee \neg q$, which is equivalent to $\neg (p \wedge q)$, is called *query* in Prolog and can be interpreted as ':- *p, q*' in which its value (true/false) to be proven by resolution method. A clause that contains exactly one positive literal such as *r* is called a *fact* representing a true statement, written in clausal form as '*r* :-' in which the condition part is empty and that means *r* is unconditionally true. Therefore, facts are used to represent data. A Horn clause that contains one positive literal and one or more negative literals such as $\neg p \vee \neg q \vee r$ is called a *definite clause* and such clause can equivalently written as $(p \wedge q) \rightarrow r$ which in turn can be represented as a Prolog *rule* as *r :- p, q*. The symbol ':-' is intended to mean '$\leftarrow$', which is implication in first-order logic (it stands for 'if'), and the symbol ',' represents the operator $\wedge$ (or 'AND'). In Prolog, rules are used to define procedures and a Prolog program is normally composed of facts and rules. Running a Prolog program is nothing more than posing queries to obtain true/false answers. The advantages of using logic programming are the flexible form of query posing and the additional information regarding variable instantiation obtained from the Prolog system once the query is evaluated to be true.

The symbols *p, q, r* are called *predicates* in first-order logic programming and they can be quantified over variables such as *r(X) :- p(X,Y),q(Y)*. This clause has the same meaning as $\forall X\,(p(X,Y) \wedge q(Y) \rightarrow r(X))$.The scope of variables is within a clause (delimit the end of clause with a period). Horn clauses are thus the fundamental concept of logic programming.

Higher-order predicate is a predicate in a clause that can quantify over other predicate symbols [13, 14]. As an example, besides the rule *r(X):- p(X,Y), q(Y)*, if we are also given the following five Horn clauses (or facts):

p(1, 2). *p(1, 3).* *p(5, 4).* *q(2).* *q(4).*

By asking the query: *?- r(X)*, we will get the response as '*true*' and also the first instantiation information as *X=1*. If we want to know all instantiations that make *r(X)* to be true, we may ask the query: *?- findall(X, r(X), Answer)*.We will get the response: *Answer = [1,5]*, which is a set of all answers obtained from the predicate *r(X)* according to the given facts. The predicate symbol *findall* quantifies over the variables *X*, *Answer*, and the predicate *r*. The predicate *findall* is thus called a higher-order predicate.

Meta-level programming is also another powerful feature of Prolog. Meta-programs treat other programs as their input data. Data and program in Prolog take the same representational format, i.e. clausal form. Therefore, it is very natural to write meta-program in Prolog. The following example illustrates the procedure *map* that takes a list of integers [1,2,3,4,5] and another procedure *square* as its input arguments and produce a list of square values as its output. If we pose the query: *?- map(square, [1,2,3,4,5], L)*, then we will get the answer: *L = [1,4,9,16,25]*.

```
square(X, Y) :- Y is X*X.
map(ProcedureName, [H|T], [NewH|NewT]) :- Procedure=.. [ProcedureName,H,NewH],
                                          call(Procedure),
                                          map(ProcedureName, T, NewT).
map(_, [], []).
```

4. SYSTEM ARCHITECTURE

The process of knowledge discovery is complex and iterative in its nature. We design the system (Figure 1) to be composed of two phases: knowledge induction and knowledge inferring.

Knowledge induction is the back-end of the system responsible for acquiring and discovering new and useful knowledge. Usefulness is to be validated at the final step by human experts. Discovered knowledge is stored in the knowledge base to be applied to solve new cases or create new knowledge in the knowledge inferring phase, which is the front-end of the proposed system.

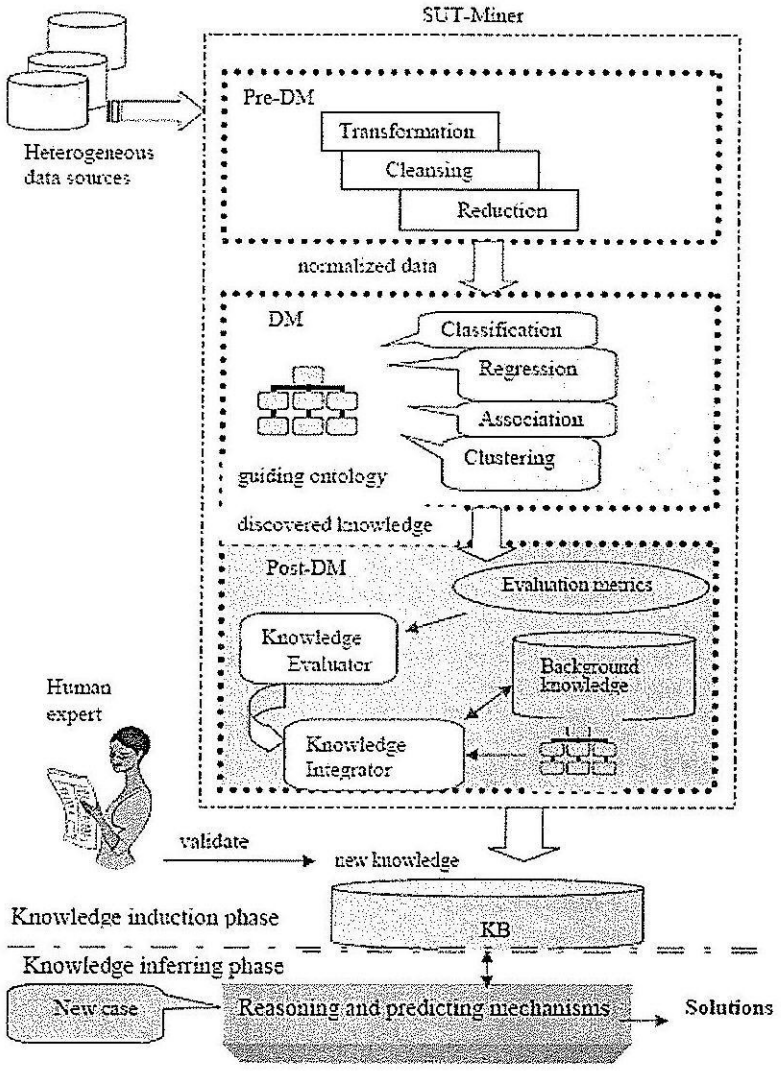


Figure 1. Architecture of the SUT-Miner system

The SUT-Miner system obtains input from heterogeneous data sources. Therefore, redundancy, incompleteness, noise can be expected from the input data. The Pre-DM component has been designed to clean, transform the format, and select only relevant data. The DM component is for performing various mining tasks. Currently, we design and implement three different mining modules, i.e. classification, association, and clustering. We adopt the ontology concept at this step to guide the mining methodology selection. A simple form of ontology to select appropriate mining method is shown in Figure 2.

The Post-DM component composed of two main features: knowledge evaluator and knowledge integrator. These features perform functionality aiming at a feasible knowledge deployment. Knowledge evaluator involves evaluation, based on corresponding measurement metrics, of the mining results. Knowledge integrator examines the induced patterns to remove redundant knowledge. Ontology has also been applied at this step to provide essential semantics regarding the domain problems.

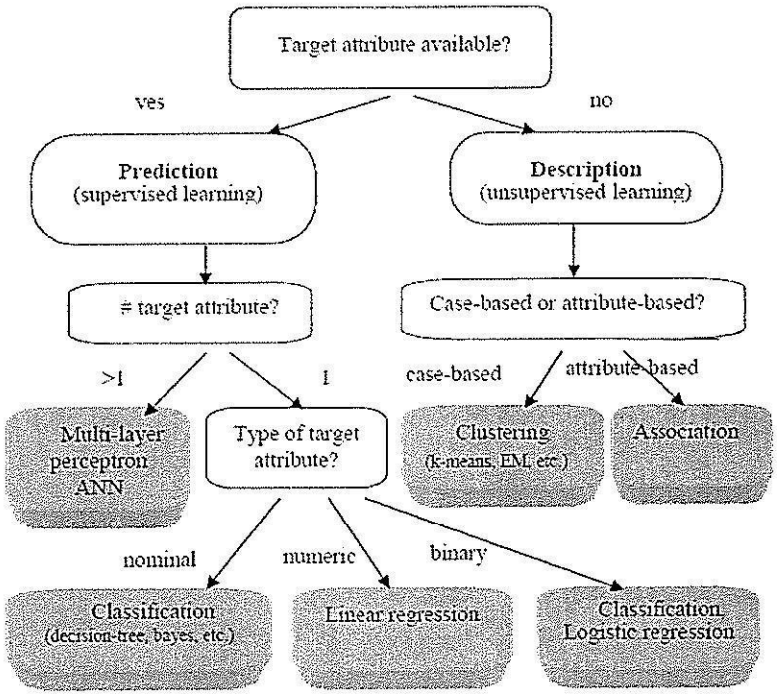


Figure 2. Ontology for guiding mining-method selection at the DM step

5. THE IMPLEMENTATION OF SUT-MINER

The SUT-Miner system has been implemented using the logic programming paradigm with extensive use of higher-order predicates. In first-order logic given a general rule $\forall X\ man(X) \rightarrow mortal(X)$ and a known fact *man(socretes)*, we can deduce new fact that *mortal(socretes)*. First-order logic poses restriction on the type of quantified variable X appeared in the predicates *man* and *mortal* to be instantiated by either atom (such as *socretes*), other variable, or functor, but not a predicate. Higher-order logic, on the other hand, allows variables to quantify over predicates. With such relaxation, higher-order logic facilitates the efficient implementation of a knowledge discovery system that requires other background knowledge to be its input in a clausal form.

Classification. We present in Figure 3 the coding of data classification module based on decision tree induction (ID3) algorithm [16]. Prolog code is based on the syntax of SWI Prolog (www.swi-prolog.org).
The *main* module in Figure 3 calls the *init* procedure and starts creating edges and nodes of the decision tree via the predicates *getNode* and *create_edge*, respectively. The ID3 implementation is in a separate file (id3.pl) and the source code is provided in Figure 4. The data to be used by main module to create decision tree is also in another Prolog file (data.pl). A small data set of ten instances is shown in Figure 5.
It can be noticed that program and data take the same format, i.e. all are in Prolog clausal form. Knowledge induction modules shown in Figures 3 and 4 are to induce data model of two classes: positive (class=yes) and negative (class=no). Binary classification is a typical task performed in many domains. The code can be easily modified to classify data with more than two classes.

```
:- include('data.pl').
:- include('id3.pl').
:- dynamic current_node/1,node/2,edge/3.

main :-     init(AllAttr,EdgeList),
            getNode(N),   % get node sequence number
            create_edge(N,AllAttr,EdgeList),
            print_model.

init(AllAttr,[root-nil/PB-NB]) :-  retractall(node(_,_)),
                                   retractall(current_node(_)),
                                   retractall(edge(_,_,_)),
                                   assert(current_node(0)) ,
                                   findall(X, attribute(X,_), AllAttr1),
                                   delete(AllAttr1, class, AllAttr),
                                   findall(X2,instance(X2,class=yes,_),PB),
                                   findall(X3,instance(X3,class=no,_),NB).

getNode(X) :- current_node(X),
              X1 is X+1,
              retractall( current_node(_)),
              assert( current_node(X1)).

create_edge(_,_,[]) :- !.
create_edge(_,[],_) :- !.
create_edge(N, AllAttr, EdgeList) :-  create_nodes(N, AllAttr, EdgeList).

create_nodes(_,_,[]) :- !.
create_nodes(_,[],_) :- !.
create_nodes(N, AllAttr, [H1-H2/PB-NB|T]) :- getNode(N1),   % get node seq num N1
                                             assert(edge(N,H1-H2,N1)),
                                             assert(node(N1,PB-NB)),
                                             append(PB, NB, AllInst),
                                             ( (PB \== [], NB \== []) ->
                                                   (cand_node(AllAttr, AllInst, AllSplit),
                                                   best_attribute(AllSplit, [V, MinAttr, Split]),
                                                   delete(AllAttr, MinAttr, Attr2),
                                                   create_edge( N1, Attr2, Split))
                                             ;  true ),
                                             create_nodes(N, AllAttr, T).
```

Figure 3. Main module of tree-based classifier

```
best_attribute([], Min, Min).
best_attribute([H|T], Min) :-     best_attribute(T, H, Min).
best_attribute([H|T], Min0, Min) :-   H =[V,_, _ ],
                                      Min0 = [V0, _, _ ],
                                      ( V < V0 -> Min1 = H; Min1 = Min0),
                                      best_attribute(T, Min1, Min).

  % generate candidate decision node
cand_node([],_,[]) :- !.
cand_node(_,[],[]).
cand_node([H|T],CurIns,[[Val,H,SplitL]|OtherAtt]) :- info(H, CurIns, Val, SplitL),
                                                     cand_node(T,CurIns,OtherAtt).

  % compute Info of each candidate node
concat3(A,B,C,R) :-  atom_concat(A,B,R1),
                     atom_concat(R1,C,R).
info(A, CurInstL, R, Split) :-    attribute(A,L),
                                  maplist( concat3(A,=), L, L1),
                                  suminfo(L1, CurInstL, R, Split).
suminfo([],_,0,[]).
suminfo([H|T], CurInstL, R, [Split | ST]) :-
       AllBag=CurInstL,
      term_to_atom(H1,H),
       findall(X1, (instance(X1,_,L1), member(X1, CurInstL), member(H1,L1)), BagGro),
       findall(X2,(instance(X2,class=yes, L2), member(X2, CurInstL),
                  member(H1,L2)), BagPos),
       findall(X3,(instance(X3,class=no, L3), member(X3, CurInstL),
                  member(H1,L3)), BagNeg),
      (H11=H22) =H1, length(AllBag, Nall),
       length(BagGro, NGro),
       length(BagPos, NPos),
       length(BagNeg, NNeg),
       Split = H11-H22/BagPos-BagNeg,
       suminfo(T, CurInstL, R1,ST),
      ( NPos is 0 *->L1 = 0;   L1 is (log(NPos/NGro)/log(2)) ),
      ( 0 is NNeg *->L2 = 0;   L2 is (log(NNeg/NGro)/log(2)) ),
      ( NGro is 0 -> R= 999;   R is (NGro/Nall)*(-(NPos/NGro)*L1- (NNeg/NGro)*L2)+R1 ).
```

Figure 4. ID3 algorithm implemented with Prolog using higher-order predicates

```
%% Data: Allergy diagnosis

 % patients' symptoms and their possible values
attribute( soreThroat,  [yes, no]).
attribute( fever, [yes, no]).
attribute( swollenGlands, [yes, no]).
attribute( congestion, [yes, no]).
attribute( headache, [yes, no]).
attribute( class, [yes, no]).

   % data instances
instance(1, class=no, [soreThroat=yes, fever=yes, swollenGlands=yes, congestion=yes,
          headache=yes]).
instance(2, class=yes, [soreThroat=no,  fever=no,  swollenGlands=no,  congestion=yes,
          headache=yes]).
instance(3, class=no, [soreThroat=yes, fever=yes, swollenGlands=no,  congestion=yes,
          headache=no]).
instance(4, class=no,  [soreThroat=yes, fever=no,  swollenGlands=yes, congestion=no,
          headache=no]).
instance(5, class=no,  [soreThroat=no,  fever=yes, swollenGlands=no,  congestion=yes,
          headache=no]).
instance(6, class=yes, [soreThroat=no,  fever=no,  swollenGlands=no,  congestion=yes,
          headache=no]).
instance(7, class=no,  [soreThroat=no,  fever=no,  swollenGlands=yes, congestion=no,
          headache=no]).
instance(8, class=yes, [soreThroat=yes, fever=no, swollenGlands=no,  congestion=yes,
          headache=yes]).
instance(9, class=no,  [soreThroat=no,  fever=yes, swollenGlands=no,  congestion=yes,
          headache=yes]).
instance(10, class=no, [soreThroat=yes, fever=yes, swollenGlands=no,  congestion=yes,
          headache=yes]).
```

Figure 5. Data of ten patients in which three are suffering from allergy (class=yes)

Data shown in Figure 5 are patient records suffering from allergy (class = yes). The possible indicative symptoms are sore throat, fever, swollen glands, congestion, and headache. Some patients had some of these symptoms but are not suffering from allergy (class = no). To induce the common symptoms (or model) of allergy patients from this data, we have to save this data set as a Prolog file (data.pl) and include this file at the header declaration of the main program (in Figure 3). The source code in Figure 3 does not provide detail for *print_model* predicate. Interested readers are suggested to simply add a rule *print_model :- true*. Then run the program by calling predicate *main*. Prolog will respond *true* with no other information because we simply add the always-true condition in the *print_model* predicate. At this moment we can view the tree model by calling *listing(node)* and *listing(edge)* predicates. The results will be as follows:

```
1 ?- main.
true.

2 ?- listing(node).
:- dynamic user:node/2.
user:node(1, [2, 6, 8]-[1, 3, 4, 5, 7, 9, 10]).
user:node(2, []-[1, 3, 5, 9, 10]).
user:node(3, [2, 6, 8]-[4, 7]).
user:node(4, []-[4, 7]).
user:node(5, [2, 6, 8]-[]).
true.

3 ?- listing(edge).
:- dynamic user:edge/3.
user:edge(0, root-nil, 1).
user:edge(1, fever-yes, 2).
user:edge(1, fever-no, 3).
user:edge(3, swollenGlands-yes, 4).
user:edge(3, swollenGlands-no, 5).
true.
```

The node and edge structures have the following formats:

node(nodeID, [Positive_Cases]- [Negative_Cases])
edge(ParentNode, EdgeLabel, ChildNode)

The node structure is composed of two parts: node-id and the mixture of positive and negative cases in that node. For instance, node number 3 contains a mixture of three positive cases (i.e. case numbers 2, 6, 8) and two negative cases (i.e. case numbers 4 and 7). The edge is a link from parent node to child node. Each edge contains three pieces of information; that is, id of parent node, the edge label, and id of child node. Node id 0 is a special node representing a root node and it links to node number 1. From node number 1, the edge with label fever-yes (representing attribute fever with a value yes) links to node number 2. Node 1 contains all ten cases of patients suffering and not suffering from allergy, whereas node 2 contains the information []-[1,3,5,9,10] to infer none of positive cases and five negative cases. Therefore, the results in the above node and edge structures represent the following data model:

class(allergy) :- fever=no, swollenGlands=no.

Association mining. We implement the association mining module based on the algorithm APRIORI [1]. The implementation (Figure 6) shows only the first pass of algorithm; that is, the generation of frequent itemsets. The second pass, which is the generation of association rules from frequent itemsets, can be easily extended from the given code. Main predicate of this module is *association_mining*. Upon invocation, this predicate obtains input data from the predicate *input(Data)*, and get the minimum support value through the predicate *min_support(V)*. Then the main predicate starts the process by making candidate and large itemsets of length one, two, three, and so on (through the predicates *makeC1*, *makeL*, and *apriori_loop*, respectively). All highlighted terms in Figure 6 are higher-order predicates. These predicates are *maplist*, *include*, and *setof*. The predicate *maplist* takes three arguments; therefore, it may be written as *maplist/3*. This predicate applies its first argument, which is also a predicate, to each element of a list appeared in the second argument. The result is a list in the third argument.

The predicate *include/3* takes another predicate as its first argument and adds the result obtained from the first argument to the list in second argument. The result appears as a list in the third argument. The predicate *setof/3* also works with other predicate to collect each answer as a list in its third argument.

```
association_mining :-   input(Data), min_support(V),
                        makeC1(C), makeL(C,L),
                        apriori_loop(L,1).
apriori_loop(L,N) :- length(L) is 1,!.
apriori_loop(L,N) :-    N1 is N+1,
                        makeC(N1,L,C), makeL(C, Res),
                        apriori_loop(Res, N1).
makeC1(Ans) :-  input(D),
                allComb(1, ItemSet, Ans2),
                maplist(countSS(D), Ans2, Ans).
makeC(N,ItemSet,Ans) :-     input(D),
                            allComb(2,ItemSet, Ans1),
                            maplist(flatten, Ans1, Ans2),
                            maplist(list_to_ord_set, Ans2, Ans3) ,
                            list_to_set(Ans3,Ans4),
                            include(len(N), Ans4, Ans5),
                            maplist(countSS(D), Ans5, Ans).
        %scan database to find: List+N
makeL(C,Res) :- include(filter, C, Ans), maplist(head, Ans, Res).
filter(_+N) :- input(A), length(A, I),  min_support(V), N>=(V/100)*I.
head(H+_,H).
   % arbitrary subset of the set containing
   % given number of elements
comb(0, _, []).
comb(N, [X|T], [X|Comb]) :- N>0, N1 is N-1,  comb(N1,T,Comb).
comb(N,[_|T],Comb) :- N>0, comb(N,T,Comb).
allComb(N,I,Ans) :- setof( L, comb(N, I, L), Ans).
countSubset(A,[],0).
countSubset(A,[B|X],N) :- not(subset(A,B)),  countSubset(A,X,N).
countSubset(A,[B|X],N) :- subset(A,B),  countSubset(A,X,N1), N is N1+1.
countSS(SL,S,S+N) :- countSubset(S,SL,N).
len(N,X) :- length(X,N1), N is N1.
```

Figure 6. Association mining in SUT-Miner implemented with higher-order predicates

```
clustering(K) :-  makeInitCluster(K, AllClust),
                  assignPoint(AllClust, Data, Start, AllPoint),
                  OldClust=AllClust,
                  repeatCompute(K, AllPoint, OldClust).
makeInitCluster(K, AllClust) :-  initClust(K, 1, AllClust).
initClust(K, L0, []) :- L0>K , ! .
initClust(K, L0, [L0*L | T]) :- instance(L0,_,L),  L1 is L0+1, initClust(K, L1, T).
assignPoint(_, U, M, []) :- M>U, !.
assignPoint(AllClust, U, M, [M-V-A | T]) :- maplist(freq(M), AllClust, Res),
                                            cmax(Res, A*V),  M1 is M+1,
                                            assignPoint(AllClust, U, M1, T).
freq(X, N*Y, N*F) :- instance(X, _, L1),
                     intersection(L1, Y, I), length(I, F).
cmax(L, A*V) :- maplist(cvalue, L, L2),
                max_list(L2, V), member(A*V, L), !.
cvalue(_*V, V).
reComputeCenter(K, S, AllPoint, []) :- S>K, !.
reComputeCenter(K, S, AllPoint, [S*NewC | T]) :-  findall(P, member(P-_-S, AllPoint), Z),
                                                  allPointAtAllAttr(Z, NewC),
                                                  S1 is S+1,
                                                  reComputeCenter(K, S1, AllPoint, T).
allPointAtAllAttr(AllP, NewClusters) :-  findall(AttName, (attribute(AttName,_),
                                         AttName\==class), AttNameL),
                                         maplist(allPoint(AllP), AttNameL, NewClusters).
allPoint(AllP, Att, A) :-  findall(Att=V, (instance(X, _, K),
                           member(X, AllP), member(Att=V, K)) , Z),
                           maxFreq(Z, A*V).
maxFreq(L, A*V) :-  findall(X*C, (member(X,L), count(X,L,C)), Z),  cmax(Z,A*V).
repeatCompute(K, AllPoint, OldClust) :-  reComputeCenter(K,Start,AllPoint,NewClus),
                                         ( OldClust==NewClus ->
                                         writeln('-No-cluster-changes***EndProcess*');
                                         ( writeln(newClust-NewClus),
                                           assignPoint(NewClus,Data,Start,AllPoint2),
                                           writeln(allNewPoint-AllPoint2),
                                           repeatCompute(K, AllPoint2, NewClus) ) ).
```

Figure 7. Clustering with k-means algorithm

Clustering. Figure 7 demonstrates the implementation of k-means clustering [11]. The main predicate is *clustering* in which the number of clusters (k) has to be specified and data are to be included. The predicate *makeInitCluster* creates initial k clusters with randomized k centroids, then assign each data to the closest centroid through the predicate *assignPoint*. Note that the symbol '*', such as those appear in the predicate *cmax(Res, A*V)* and *freq(X, N*Y, N*F)*, refers to the data format to represent *Attribute*Value*; it does not mean multiplication. In Prolog, numerical computation will occur in a clause with the predicate *'is'*, such as *S1 is S + 1* in the *reComputeCenter* procedure.

The iteration step, *repeatCompute* predicate, re-computes the new k centroids and then re-assign each data point to the new closest centroid. Iteration stops when all data do not change their clusters. The source code presented in Figure 7 works with categorical data. For numerical or data with mixing types, the distance measurement has to be modified.

6. CONCLUSIONS

In this paper we propose the design and implementation of SUT-Miner, a logic-based knowledge discovery system. The system is intended to support automatic knowledge acquisition in any domains that require new knowledge to support better decisions as well as to enhance comprehension of stored data. The proposed knowledge discovery environment is composed of tools and methods suitable for various kinds of knowledge discovery tasks including data classification, association discovery, and data clustering.

The implementation of the proposed system is based on the concept of logic programming using some higher-order predicates such as *maplist*, *findall*, *setoff*, and *include*. These predicates take other predicates as its argument. With such expressive power of higher-order predicates, program coding of the designed system is very concise as demonstrated in the paper. Program conciseness contributes directly to program verification and validation, which are important issues in software engineering. The declarative style of programming also eases the extension of the present system towards the constraint higher-order mining, which is our future research plan.

ACKNOWLEDGMENT

This research has been funded by grants from the National Research Council and the Thailand Research Fund (TRF, grant number RMU5080026). Data Engineering and Knowledge Discovery (DEKD) Research Unit is fully supported by research grants from Suranaree University of Technology.

REFERENCES

[1] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. (1996). Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp.307-328. AAAI Press.

[2] Bisson, G. (1992). Conceptual clustering in a first-order logic representation. *Proceedings of the 10th European Conference on Artificial Intelligence*, pp.458-462. John Wiley.

[3] Blockeel, H., De Raedt, L., & Ramon, J. (1998). Top-down induction of clustering trees. *Proceedings of the 5th International Conference on Machine Learning (ICML-98)*, pp.55-63. Morgan Kaufmann.

[4] Dehaspe, L. & Toivonen, H. (1999). Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1): 7-36.

[5] Dehaspe, L. & Toivonen, H. (2001). Discovery of relational association rules. In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, pp.189-212. Springer.

[6] Emde, W. (1994). Inductive learning of characteristic concept descriptions from small sets to classified examples. *Proceedings of the 7th European Conference on Machine Learning*, LNAI 784, pp.103-121. Springer.

[7] Han, J. & Kamber, M. (2006). *Data Mining: Concepts and Techniques, 2nd edition*. Morgan Kaufmann.

[8] Kirsten, M. & Wrobel, S. (1998). Relational distance-based clustering. *Proceedings of the 8th International Conference on Inductive Logic Programming*, LNAI 1446, pp.261-270. Springer.

[9] Kirsten, M. & Wrobel, S. (2000). Extending k-means clustering to first-order representations. *Proceedings of the 10th International Conference on Inductive Logic Programming*, LNAI 1866, pp.112-129. Springer.

[10] Kramer, S. & Widmer, G. (2001). Inducing classification and regression trees in first order logic. In S. Dzeroski and N. Lavrac (Eds.), *Relational Data Mining*, pp.140-159. Springer.

[11] MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1*, pp.281-297. University of California Press.

[12] Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13:245-286.

[13] Nadathur, G. & Miller, D. (1990). Higher-order Horn clauses. *Journal of the ACM*, 37:777-814.

[14] Naish, L. (1996). Higher-order logic programming in Prolog. *Technical Report 96/2*, Department of Computer Science, University of Melbourne, Australia.

[15] Nienhuys-Cheng, S.-H. & Wolf, R. (1997). *Foundations of Inductive Logic Programming*. Springer.

[16] Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1:81-106.

[17] Quinlan, J. & Cameron-Jones, R. (1993). FOIL: A midterm report. *Proceedings of the 6th European Conference on Machine Learning*, LNAI 667, pp.3-20. Springer.

ประวัติผู้วิจัย

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ สำเร็จการศึกษาในระดับปริญญาเอกสาขา Computer Science จาก Nova Southeastern University เมือง Fort Lauderdale รัฐฟลอริดา ประเทศสหรัฐอเมริกา เมื่อปีพุทธศักราช 2542 (ค.ศ. 1999) ด้วยทุนการศึกษาของทบวงมหาวิทยาลัย (หรือสำนักงานคณะกรรมการการอุดมศึกษาในปัจจุบัน) โดยทำวิทยานิพนธ์ระดับปริญญาเอกในหัวข้อเรื่อง "Active database rule set reduction by knowledge discovery" หลังสำเร็จการศึกษาได้ปฏิบัติงานในตำแหน่งอาจารย์ ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ปัจจุบันดำรงตำแหน่งหัวหน้าหน่วยปฏิบัติการวิจัยด้านวิศวกรรมข้อมูลและการค้นหาความรู้ (Data Engineering and Knowledge Discovery Research Unit – DEKD) สำนักวิชาวิศวกรรมศาสตร์ ดำเนินการวิจัยประยุกต์เกี่ยวกับการออกแบบและพัฒนาระบบเหมืองข้อมูลประสิทธิภาพสูงที่สามารถทนต่อข้อมูลรบกวน และการวิจัยพื้นฐานเกี่ยวกับเทคนิคการจัดกลุ่มข้อมูล และเทคนิคการวิเคราะห์ข้อมูลขนาดใหญ่ด้วยฮิวริสติก โดยมีผลงานวิจัยตีพิมพ์ในวารสารวิชาการและเอกสารการประชุมวิชาการ จำนวนมากกว่า 30 เรื่อง ในสาขาระบบฐานความรู้ ฐานข้อมูลแอคทีฟ ฐานข้อมูลนิรนัย การทำเหมืองข้อมูลและการค้นหาความรู้