รหัสโครงการ **SUT7-705-47-24-01**

รายงานการวิจัย

การพัฒนาซอฟต์แวร์สร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทนต่อข้อมูลรบกวน

(Software development for noise-tolerant decision-tree induction)

ผู้วิจัย

หัวหน้าโครงการ

รองศาสตราจารย์ ดร.นิตยา เกิดประสพ

สาขาวิชาวิศวกรรมคอมพิวเตอร์

สำนักวิชาวิศวกรรมศาสตร์

ผู้วิจัยร่วม

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ

ได้รับทุนอุดหนุนการวิจัยจากมหาวิทยาลัยเทคโนโลยีสุรนารี ปีงบประมาณ พ.ศ. 2547

ผลงานวิจัยเป็นความรับผิดชอบของหัวหน้าโครงการวิจัยแต่เพียงผู้เดียว

มิถุนายน 2552

กิตติกรรมประกาศ

ผู้วิจัยขอขอบคุณมหาวิทยาลัยเทคโนโลยีสุรนารีและสำนักงานคณะกรรมการวิจัยแห่งชาติ ที่ได้จัดสรรงบประมาณให้ในปีงบประมาณ 2547 และ 2548 เพื่อสนับสนุนโครงการวิจัยนี้ ขอขอบคุณผู้ทรงคุณวุฒิที่ได้เสียสละเวลาทำหน้าที่ตรวจข้อเสนอโครงการ และตรวจร่างรายงานการวิจัยฉบับสมบูรณ์ งานวิจัยนี้สำเร็จได้อย่างดีด้วยการมีส่วนร่วมจากนักศึกษาสาขาวิชาวิศวกรรมคอมพิวเตอร์ทั้งในระดับปริญญาบัณฑิตและบัณฑิตศึกษา ที่ได้ทำหน้าที่ผู้ช่วยวิจัยในโครงการวิจัยนี้ โดยเฉพาะนายนฤพนต์ ว่องประชานุกูล นักศึกษาปริญญาโทสาขาวิศวกรรมคอมพิวเตอร์ รุ่นที่ 1 ที่ได้นำแนวคิดของโครงการวิจัยนี้ไปประยุกต์เป็นหัวข้อวิทยานิพนธ์

บทคัดย่อภาษาไทย

การสร้างต้นไม้ตัดสินใจเชิงอุปนัยเป็นงานที่สำคัญงานหนึ่งของการทำเหมืองข้อมูล และการเรียนรู้ของเครื่อง การทำเหมืองข้อมูลเป็นกระบวนการคัดแยกสารสนเทศที่เป็นประโยชน์และยังไม่เคยปรากฏมาก่อน เช่นรูปแบบร่วม หรือ ความสัมพันธ์ที่ซ่อนอยู่ในกลุ่มข้อมูล เทคนิคที่ใช้ในการค้นหารูปแบบร่วมที่น่าสนใจนี้มีอยู่หลากหลายเทคนิค ต้นไม้ตัดสินใจเป็นเครื่องมือชนิดหนึ่งที่นิยมใช้กันมากในงานทำเหมืองข้อมูล เทคนิคทุกชนิดที่ใช้ในงานทำเหมืองข้อมูลล้วนถูกขับเคลื่อนด้วยตัวข้อมูล แต่ข้อมูลที่ใช้ก็มักจะมีขนาดใหญ่และมีข้อผิดพลาดปะปนอยู่ ข้อผิดพลาดคือความบกพร่องที่ปรากฏแบบสุ่มที่ตำแหน่งใดก็ได้ ข้อมูลที่ผิดพลาดมีได้หลายรูปแบบ ได้แก่ การระบุค่าของคลาสผิด ค่าของแอททริบิวต์ผิด หรือข้อมูลที่มีรายละเอียดของทุกแอททริบิวต์เหมือนกันแต่มีค่าของคลาสต่างกันทำให้เกิดเป็นกรณีขัดแย้ง การมีข้อผิดพลาดทำให้ข้อมูลกลายเป็นข้อมูลรบกวน ข้อมูลรบกวนทุกประเภทล้วนมีผลต่อประสิทธิภาพการเรียนรู้ ผลกระทบที่ร้ายแรงที่สุดคือทำให้อัลกอริทึมการเรียนรู้สร้างผลลัพธ์ที่ซับซ้อนและผิดเพี้ยน ผลลัพธ์ที่มีขนาดใหญ่และซับซ้อนเกิดจากความพยายามที่จะสร้างโมเดลที่อธิบายทั้งข้อมูลที่ถูกและผิด สิ่งนี้ทำให้เกิดปัญหาที่เรียกว่า โมเดลที่จำเพาะเจาะจงมากเกินไป

อัลกอริทึมการเรียนรู้มักจะได้รับการออกแบบให้หลีกเลี่ยงปัญหาของการสร้างโมเดลที่จำเพาะเจาะจงมากเกินไป เทคนิคที่มักจะใช้จัดการไม่ให้เกิดการสร้างต้นไม้ตัดสินใจขนาดใหญ่ที่จะไปครอบคลุมข้อมูลรบกวนคือการตัดกิ่งของต้นไม้ ทั้งที่เป็นการตัดก่อนที่ต้นไม้จะมีขนาดใหญ่หรือตัดหลังจากพบว่าต้นไม้ใหญ่เกินไป เทคนิคทั้งสองประเภทมีข้อเหมือนกันคือผนวกการตัดกิ่งไว้ในขั้นตอนการสร้างต้นไม้ งานวิจัยนี้พิจารณาวิธีการจัดการกับข้อมูลรบกวนในแง่มุมที่ต่างออกไป โดยจะแยกขั้นตอนการจัดการกับข้อมูลรบกวนออกจากขั้นตอนการสร้างต้นไม้ ข้อมูลทั้งหมดที่มีทั้งข้อมูลที่ดีปะปนกับข้อมูลรบกวนจะถูกจัดกลุ่มและคัดเลือกด้วยฮิวริสติก ก่อนที่จะถูกส่งต่อไปยังขั้นตอนการสร้างต้นไม้ จากผลการทดลองพบว่าในข้อมูลปกติวิธีการใหม่ที่เสนอขึ้นนี้สามารถสร้างโมเดลที่มีความแม่นตรงสูงเทียบเท่ากับวิธี ID3 และเมื่อข้อมูลมีข้อรบกวนมากขึ้นเทคนิคที่พัฒนาขึ้นนี้ยังสามารถสร้างโมเดลที่มีความแม่นตรงสูงในขณะที่ ID3 จะให้โมเดลที่มีความแม่นตรงลดลง

บทคัดย่อภาษาอังกฤษ

Decision tree induction is a major task in data mining and machine learning. Data mining is the process of extracting useful and yet unknown information such as patterns or association hidden in stored data. Among various existing techniques applied to search for interesting patterns, decision tree is one of the most popular tools used for data mining. Most data mining techniques are data-driven, however, data repositories of interest in data mining applications can be very large and noisy. Noise is a random error in data. Noise in a data set can happen in different forms: misclassification or wrong labeled instances, erroneous or distorted attribute values, contradictory or duplicate instances having different labels. All kinds of noise can more or less affect the learning performance. The most serious effect of noise is that it can confuse the learning algorithms to produce complex and distorted results. The long and complex results are due to the attempt to fit every training data instance, including noisy ones, into the concept descriptions. This is a major cause of overfitting problem.

Most learning algorithms are designed with the awareness of overfitting problem due to noisy data. Prepruning and postprocessing are two major techniques applied to avoid growing a decision tree too deep down to cover the noisy training data. These techniques are tightly coupled to the tree induction phase. We, on the contrary, design a loosely coupled approach to deal with noisy data. Our noise-handling feature is in a separate phase from the tree induction. Both corrupted and uncorrupted data are clustered and heuristically selected prior to the application of tree induction engine. We observe from our experimental study that tree models produced from our approach are as accurate as the models generated by conventional ID3 approach. Moreover, upon highly corrupted data our approach shows a better performance than the ID3 approach.

สารบัญ

หน้า

กิตติกรรมประกาศ ..... ก

บทคัดย่อภาษาไทย ..... ข

บทคัดย่อภาษาอังกฤษ ..... ค

สารบัญ ..... ง

สารบัญตาราง ..... ฉ

สารบัญภาพ ..... ช

บทที่ 1 บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย 1

1.2 วัตถุประสงค์ของการวิจัย 8

1.3 ขอบเขตของการวิจัย 8

1.4 ประโยชน์ที่ได้รับจากการวิจัย 8

บทที่ 2 การสร้างต้นไม้ตัดสินใจเชิงอุปนัย

2.1 โครงสร้างและการใช้งานต้นไม้ตัดสินใจ 10

2.2 ขั้นตอนการสร้างต้นไม้ตัดสินใจจากข้อมูลฝึก 12

2.3 การตรวจสอบประสิทธิภาพของต้นไม้ตัดสินใจ 19

2.4 งานวิจัยที่เกี่ยวข้องกับการสร้างต้นไม้ตัดสินใจเชิงอุปนัย 23

บทที่ 3 วิธีการสร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทนต่อข้อมูลรบกวน

3.1 กรอบของงานวิจัย 26

3.2 การพัฒนาซอฟต์แวร์สร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทนต่อข้อมูลรบกวน 28

3.2.1 รูปแบบของข้อมูล 28

3.2.2 ผลลัพธ์ในลักษณะของโมเดล 29

3.2.3 โมดูล Main 31

3.2.4 โมดูล Clustering 34

3.2.5 โมดูล Data selection 36

3.2.6 โมดูล Tree induction 36

3.2.7 การทดสอบโมเดลด้วยโมดูล Testing 38

บทที่ 4 การทดสอบความแม่นตรงและความทนทานของต้นไม้ตัดสินใจ

4.1 ข้อมูลที่ใช้ในการทดสอบ 40

4.2 วิธีการทดสอบความแม่นตรงและความทนทาน 41

หน้า

4.3 ผลการทดสอบ 43

บทที่ 5 บทสรุป

5.1 สรุปผลการวิจัย 52

5.2 ข้อจำกัดของซอฟต์แวร์และข้อเสนอแนะ 56

บรรณานุกรม 57

ภาคผนวก

ก บทความวิจัยนำเสนอในการประชุมวิชาการ 59

1. N. Kerdprasop and K. Kerdprasop (2008). A declarative programming paradigm and the development of knowledge mining agents. *Proceedings of IADIS Multi Conference on Computer Science and Information Systems*, Amsterdam, Netherlands, July 22-27, pp. 45-52. 60
2. N. Wongprachanukul, N. Kerdprasop, and K. Kerdprasop (2005). A comparative study of method for pruning decision trees. *Proceedings of 31st Congress on Science and Technology of Thailand*, Suranaree University of Technology, Nakhon Ratchasima, Thailand, October 18-20. 68
3. K. Kerdprasop, N. Kerdprasop and N. Wongprachanukul (2005). การศึกษาเปรียบเทียบวิธีการลดความซับซ้อนของโมเดลข้อมูล. *Proceedings of the Research Network Development of Higher Education Alliance in Nakhon Ratchasima*, Suranaree University of Technology, Nakhon Ratchasima, Thailand, June 24, pp.68-70..... 71
4. N. Kerdprasop, K. Kerdprasop, L. Khomnotai and T. Thianniwet (2004). The impact of noise at different data attributes. *Proceedings of 1st KMITL International Conference on Integration of Science and Technology for Sustainable Development*, Bangkok, Thailand, August 25-26, pp.275-278. 74
5. N. Kerdprasop and K. Kerdprasop (2003). Data Partitioning for incremental data mining. *Proceedings of 1st International Forum on Information and Computer Science (IFICT 2003)*, Shizuoka University, Japan, January 9-10, pp.114-118. 78
6. N. Kerdprasop, K. Kerdprasop, Y. Saiveaw, and P. Pumrungreong (2003). A comparative study of techniques to handle missing values in the classification task of data mining. *Proceedings of 29th Congress on Science and Technology of Thailand*, Khon Kaen University, Thailand, October 20-22. 83

ข รหัสต้นฉบับภาษาโปรล็อกของซอฟต์แวร์สร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทนต่อข้อมูลรบกวน 88

ประวัติผู้วิจัย 99

สารบัญตาราง

หน้า

ตารางที่ 1.1 ข้อมูลฝึกที่ใช้สำหรับสร้างต้นไม้ตัดสินใจเชิงอุปนัย 4
ตารางที่ 1.2 ข้อมูลฝึกที่มีข้อมูลรบกวนในแอททริบิวต์คลาสของรายการที่หนึ่งและสอง 6
ตารางที่ 2.1 ข้อมูลสภาพอากาศประกอบการตัดสินใจเล่น/ไม่เล่นกอล์ฟ 15
ตารางที่ 2.2 ข้อมูลทดสอบโมเดลการตัดสินใจเล่น/ไม่เล่นกอล์ฟ 21
ตารางที่ 4.1 รายละเอียดของข้อมูลที่ใช้ทดสอบประสิทธิภาพของโมเดล 40
ตารางที่ 4.2 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Monk 44
ตารางที่ 4.3 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Audiology 45
ตารางที่ 4.4 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Breast cancer 46
ตารางที่ 4.5 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Vote 47

สารบัญภาพ

หน้า

รูปที่ 1.1 ข้อมูลฝึกซึ่งเป็นข้อมูลของคนไข้ที่ได้รับการวินิจฉัยโรคแล้วรวม 10 คน 2
รูปที่ 1.2 โมเดลของอาการคนไข้แต่ละโรคแสดงในลักษณะของต้นไม้ตัดสินใจ 3
รูปที่ 1.3 ต้นไม้ตัดสินใจประกอบการพิจารณาเล่นหรือไม่เล่นกอล์ฟ..... 4
รูปที่ 1.4 โครงสร้างต้นไม้ตัดสินใจที่เปลี่ยนโหนดรากเป็นแอททริบิวต์ outlook 5
รูปที่ 1.5 เปรียบเทียบโครงสร้างต้นไม้ตัดสินใจกรณีที่ข้อมูลถูกต้องและกรณีที่มี
ข้อมูลรบกวน 7
รูปที่ 2.1 ต้นไม้ตัดสินใจอธิบายรูปแบบของลูกค้าสองประเภทที่กู้เงินเพื่อซื้อ/ไม่ซื้อ
คอมพิวเตอร์ 11
รูปที่ 2.2 การจำแนกข้อมูลของแต่ละแอททริบิวต์ที่ทำหน้าที่เป็นโหนดตัดสินใจ 14
รูปที่ 2.3 โครงสร้างต้นไม้ตัดสินใจหลังการเลือกโหนดราก 17
รูปที่ 2.4 เปรียบเทียบค่า gain ของแอททริบิวต์ในระดับที่สอง 18
รูปที่ 2.5 ต้นไม้ตัดสินใจที่จำแนกข้อมูลได้สมบูรณ์ 19
รูปที่ 2.6 การวัดความแม่นตรงแบบ Holdout 20
รูปที่ 2.7 การวัดความแม่นตรงแบบ k-fold cross-validation 22
รูปที่ 3.1 โครงสร้างของซอฟต์แวร์ Robust-tree 27
รูปที่ 3.2 ตัวอย่างไฟล์ข้อมูลอินพุทที่จะนำเข้ายังโปรแกรม Robust-tree 29
รูปที่ 3.3 ข้อความที่ปรากฏเมื่อเรียกใช้โปรแกรม Robust-tree 30
รูปที่ 3.4 ผลลัพธ์ของโมเดลในลักษณะของต้นไม้ตัดสินใจ 30
รูปที่ 3.5 อัลกอริทึม Main ที่ทำหน้าที่เป็นส่วนติดต่อกับผู้ใช้ 32
รูปที่ 3.6 อัลกอริทึมสร้าง Robust-tree ที่ทนทานต่อข้อมูลรบกวน 33
รูปที่ 3.7 แสดงการจัดกลุ่มข้อมูลสองมิติด้วยอัลกอริทึม k-means 35
รูปที่ 3.8 อัลกอริทึม clustering ที่ทำหน้าที่จัดกลุ่มข้อมูล 35
รูปที่ 3.9 อัลกอริทึม Data selection สำหรับคัดเลือกข้อมูลตัวแทนในกลุ่ม 36
รูปที่ 3.10 อัลกอริทึม Tree induction ที่ทำหน้าที่สร้างต้นไม้ตัดสินใจ 37
รูปที่ 3.11 จอภาพแสดงการโต้ตอบกับผู้ใช้เพื่อทดสอบโมเดล 39
รูปที่ 4.1 เปรียบเทียบโครงสร้างโมเดลที่สร้างโดย ID3 และ Robust tree เมื่อข้อมูลมี noise 41

หน้า

รูปที่ 4.2 เปรียบเทียบความแม่นตรงของโมเดล ID3 และ Robust tree เมื่อสร้างจาก
ข้อมูลที่มี noise 42
รูปที่ 4.3 กราฟเปรียบเทียบค่าความแม่นตรงของ ID3 model และ Robust-tree model..... 48
รูปที่ 4.4 กราฟเปรียบเทียบขนาดของ ID3 model และ Robust-tree model 49

บทที่ 1
บทนำ

การสร้างต้นไม้ตัดสินใจเป็นวิธีที่นิยมใช้มากในงานทำเหมืองข้อมูลและงานการเรียนรู้ของเครื่อง เนื่องจากเป็นเทคนิคที่ให้ความถูกต้อง มีความแม่นตรงในการจำแนกสูงและมีขั้นตอนที่สามารถตรวจสอบและติดตามการทำงานได้ รวมถึงให้ผลลัพธ์อยู่ในรูปแบบของภาพที่ทำความเข้าใจได้ง่าย ผลลัพธ์นี้สามารถแปลงเป็นลักษณะของกฎเพื่อให้ทำหน้าที่เป็นฐานความรู้ในระบบผู้เชี่ยวชาญหรือระบบสนับสนุนการตัดสินใจได้ แต่การนำเทคนิคการสร้างต้นไม้ตัดสินใจมาใช้วิเคราะห์หารูปแบบจากข้อมูลปริมาณมาก โดยเฉพาะเมื่อข้อมูลนั้นมีความบกพร่องในลักษณะของการมีข้อมูลรบกวนปะปนอยู่ ยังคงเป็นปัญหาหลักของการวิเคราะห์หารูปแบบข้อมูล งานวิจัยนี้จึงมุ่งศึกษาในประเด็นเทคนิคการสร้างต้นไม้ตัดสินใจที่มีความทนทาน สามารถรองรับข้อมูลขนาดใหญ่ที่มีข้อมูลรบกวนปะปนอยู่โดยยังคงให้ผลลัพธ์ที่ดีมีความแม่นตรงสูง

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

ในปัจจุบันเครื่องคอมพิวเตอร์และอุปกรณ์นำเข้าข้อมูล เช่น สแกนเนอร์ (scanner), เครื่องอ่านรหัสบาร์ (bar-code reader) มีใช้อย่างแพร่หลาย ประกอบกับอุปกรณ์เก็บข้อมูล เช่น ฮาร์ดดิสก์มีราคาถูกลง ทำให้ข้อมูลที่ถูกบันทึกอยู่ในรูปแบบอิเล็กทรอนิกส์มีปริมาณมหาศาล การใช้เฉพาะแรงงานผู้เชี่ยวชาญวิเคราะห์ข้อมูลเพื่อจะนำความรู้จากข้อมูลมาใช้ให้เกิดประโยชน์เป็นสิ่งที่แทบจะเป็นไปไม่ได้ ปัญหานี้จะเห็นได้ชัดเจนในกรณีของข้อมูลที่ได้รับจากดาวเทียมสำรวจสภาพอากาศและพื้นผิวโลก ที่มีขนาดของข้อมูลเป็นเทอราไบต์ (1 Terabyte = 1024 Gigabyte) และยังมีข้อมูลใหม่ส่งมาจากดาวเทียมที่โคจรรอบโลกทุกวัน

การใช้แรงงานผู้เชี่ยวชาญวิเคราะห์ข้อมูลด้วยโปรแกรมทางสถิติ เช่น SPSS เป็นงานที่ใช้เวลามากจนไม่สามารถนำผลการวิเคราะห์มาใช้ประโยชน์ทันเวลาได้ แนวทางที่จะช่วยแก้ไขปัญหาความล่าช้านี้ ก็คือ ทำให้กระบวนการวิเคราะห์ข้อมูลเป็นอัตโนมัติมากขึ้น ลดขั้นตอนการควบคุมและสั่งงานจากผู้เชี่ยวชาญให้น้อยลง โดยให้ระบบคอมพิวเตอร์ทำหน้าที่ค้นหาแนวโน้มต่างๆ ที่น่าสนใจจากข้อมูล หรือวิเคราะห์ความสัมพันธ์ระหว่างข้อมูลได้ด้วยความสามารถของระบบเอง และเรียกกระบวนการวิเคราะห์ข้อมูลอัตโนมัตินี้ว่า *การทำเหมืองข้อมูล*

การทำเหมืองข้อมูล (data mining) หรือในบางครั้งเรียกว่าการค้นหาความรู้จากฐานข้อมูล (knowledge discovery in databases -- KDD) คือ กระบวนการค้นหาแนวโน้มของข้อมูล รูปแบบร่วมของกลุ่มข้อมูล ความสัมพันธ์ระหว่างข้อมูล ข้อมูลที่มีลักษณะผิดปกติ หรือ

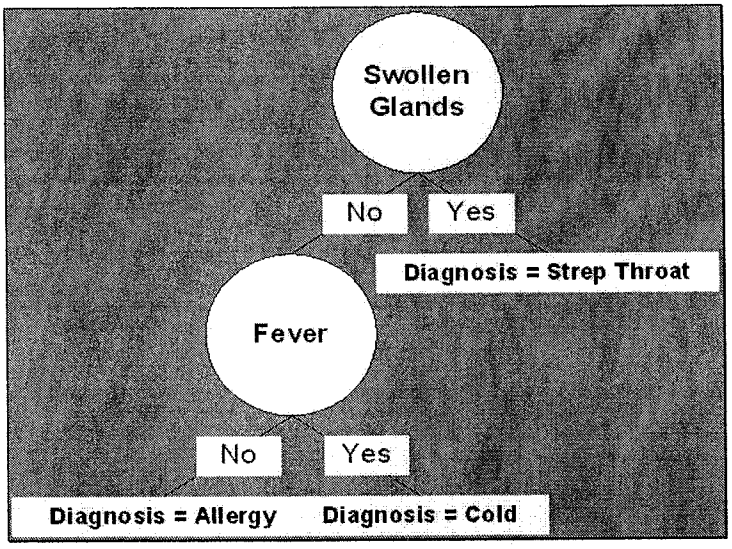
ความรู้ใหม่อื่นๆ จากข้อมูลจำนวนมาก การทำเหมืองข้อมูลจัดได้ว่าเป็นเทคโนโลยีใหม่ที่ช่วยให้การวิเคราะห์ข้อมูลทำได้โดยอัตโนมัติและมีประสิทธิภาพสูงขึ้นกว่าที่เคยเป็นมา จึงได้รับความสนใจนำไปใช้อย่างแพร่หลายในทุกวงการ โดยเฉพาะในกรณีที่ข้อมูลมีขนาดใหญ่มาก เช่นข้อมูลสำมะโนประชากร ข้อมูลที่ได้รับจากดาวเทียมสำรวจสภาพอากาศและพื้นผิวโลก

กระบวนการวิเคราะห์ข้อมูลอัตโนมัติถูกเรียกว่า*การทำเหมืองข้อมูล* เนื่องจากการทำงานของโปรแกรมเปรียบเสมือนการทำเหมืองแร่ ที่เราใช้เครื่องจักรคัดแยกแร่ที่เป็นที่ต้องการออกจากกองหิน กรวด ดินที่ปะปนมากับสายแร่ เพียงแต่ในกระบวนการทำเหมืองข้อมูลสิ่งที่เราได้จากกองข้อมูลมหาศาล คือ ความรู้ (knowledge) ที่ซ่อนอยู่ในกองข้อมูล ความรู้นี้จะช่วยให้เราเข้าใจลักษณะของข้อมูล และเข้าใจปัจจัยที่ทำให้เกิดลักษณะบางอย่างขึ้นในข้อมูลบางกลุ่ม ซึ่งจะช่วยให้เราสามารถทำนายแนวโน้มของข้อมูลใหม่ที่จะเกิดขึ้นในอนาคตได้ รวมถึงเข้าใจความสัมพันธ์ที่เชื่อมโยงข้อมูลแต่ละกลุ่มย่อยเข้าด้วยกัน ตัวอย่างเช่น จากบันทึกประวัติการตรวจรักษาคนไข้ในอดีตจำนวน 10 คน (รูปที่ 1.1) ที่มีกลุ่มอาการพื้นฐานคล้ายกันซึ่งแพทย์ได้วินิจฉัยแล้วว่าคนไข้แต่ละรายป่วยด้วยโรคหวัด (cold), ภูมิแพ้ (allergy) หรือเป็นการติดเชื้อ (strep throat) ข้อมูลในอดีตเหล่านี้จะทำหน้าที่เป็นข้อมูลฝึก (training data) เพื่อช่วยให้โปรแกรมทำเหมืองข้อมูลสามารถเรียนรู้รูปแบบของข้อมูล และสร้างผลลัพธ์ในลักษณะของโมเดล (รูปที่ 1.2) ที่แสดงรูปแบบกลุ่มอาการคนไข้ที่ป่วยด้วยไข้หวัด ภูมิแพ้ และการติดเชื้อ

ความรู้ที่ได้จากการทำเหมืองข้อมูลกับข้อมูลฝึกชุดนี้คือ การค้นพบว่ามีปัจจัยหลักเพียงสองปัจจัย (ได้แก่ swollen glands และ fever) ที่แพทย์ใช้ประกอบการวินิจฉัยว่าคนไข้เป็นหวัด, ภูมิแพ้ หรือ ติดเชื้อ

Patient ID#	Sore Throat	Fever	Swollen Glands	Congestion	Headache	Diagnosis
1	Yes	Yes	Yes	Yes	Yes	Strep throat
2	No	No	No	Yes	Yes	Allergy
3	Yes	Yes	No	Yes	No	Cold
4	Yes	No	Yes	No	No	Strep throat
5	No	Yes	No	Yes	No	Cold
6	No	No	No	Yes	No	Allergy
7	No	No	Yes	No	No	Strep throat
8	Yes	No	No	Yes	Yes	Allergy
9	No	Yes	No	Yes	Yes	Cold
10	Yes	Yes	No	Yes	Yes	Cold

รูปที่ 1.1 ข้อมูลฝึกซึ่งเป็นข้อมูลของคนไข้ที่ได้รับการวินิจฉัยโรคแล้วรวม 10 คน

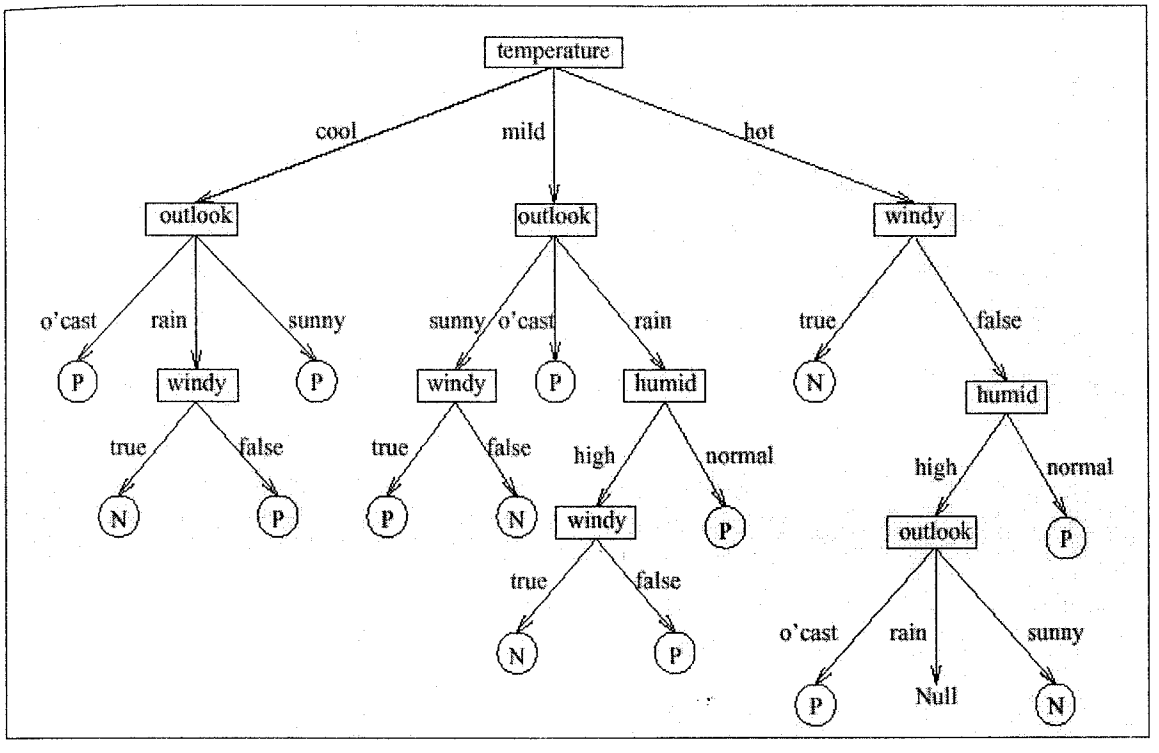


รูปที่ 1.2 โมเดลของอาการคนไข้แต่ละโรคแสดงในลักษณะของต้นไม้ตัดสินใจ

การทำเหมืองข้อมูลแบบนี้เรียกว่า การจำแนก หรือ classification โดยมีคลาสของข้อมูล 3 คลาส คือ cold, allergy, strep throat ผลลัพธ์จะเป็นโมเดลที่อธิบายลักษณะของข้อมูลแต่ละคลาส และสามารถใช้โมเดลนี้ทำนายการวินิจฉัยโรคให้กับคนไข้ใหม่ในอนาคตได้ ถ้ามีคนไข้รายใหม่มาพบแพทย์ด้วยอาการต่อมที่ลำคอบวม (Swollen Glands = Yes) โมเดลนี้สามารถทำนายได้ทันทีว่าคนไข้ป่วยเนื่องจากมีการติดเชื้อ (Diagnosis = Strep Throat)

การวิเคราะห์ข้อมูลอัตโนมัติด้วยวิธีการทำเหมืองข้อมูล มีได้หลากหลายเทคนิคเช่น การใช้โครงข่ายประสาทเทียม (neural network) การวิเคราะห์จากข้อมูลใกล้เคียง (nearest neighbor) หรือการใช้ทฤษฎีความน่าจะเป็น (Bayes theorem) แต่เมื่อต้องการผลลัพธ์ที่เข้าใจได้ง่ายส่วนใหญ่จะใช้วิธีการสร้างต้นไม้ตัดสินใจเชิงอุปนัย (decision-tree induction) เพื่อจำแนกกลุ่มข้อมูลและอธิบายลักษณะข้อมูลในแต่ละกลุ่ม

การสร้างต้นไม้ตัดสินใจใช้วิธีค้นหาปัจจัยหลักของการตัดสินใจจากข้อมูลฝึก จึงเรียกวิธีนี้ว่า การสร้างต้นไม้ตัดสินใจเชิงอุปนัย แสดงตัวอย่างได้ดังรูปที่ 1.3 (Quinlan 1993) ซึ่งเป็นภาพต้นไม้ตัดสินใจที่แสดงเงื่อนไขสภาพอากาศประกอบการตัดสินใจว่าจะออกไปเล่นกอล์ฟ (Play -- P) หรือไม่ออกไปเล่น (Don't play -- N) การสร้างต้นไม้ตัดสินใจดังภาพใช้วิธีอุปนัยหรือสังเคราะห์ขึ้นจากข้อมูลตามตารางที่ 1.1 (Quinlan 1993)



รูปที่ 1.3 ต้นไม้ตัดสินใจประกอบการพิจารณาเล่น (P)หรือไม่เล่นกอล์ฟ (N)

ตารางที่ 1.1 ข้อมูลฝึกที่ใช้สำหรับสร้างต้นไม้ตัดสินใจเชิงอุปนัย

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

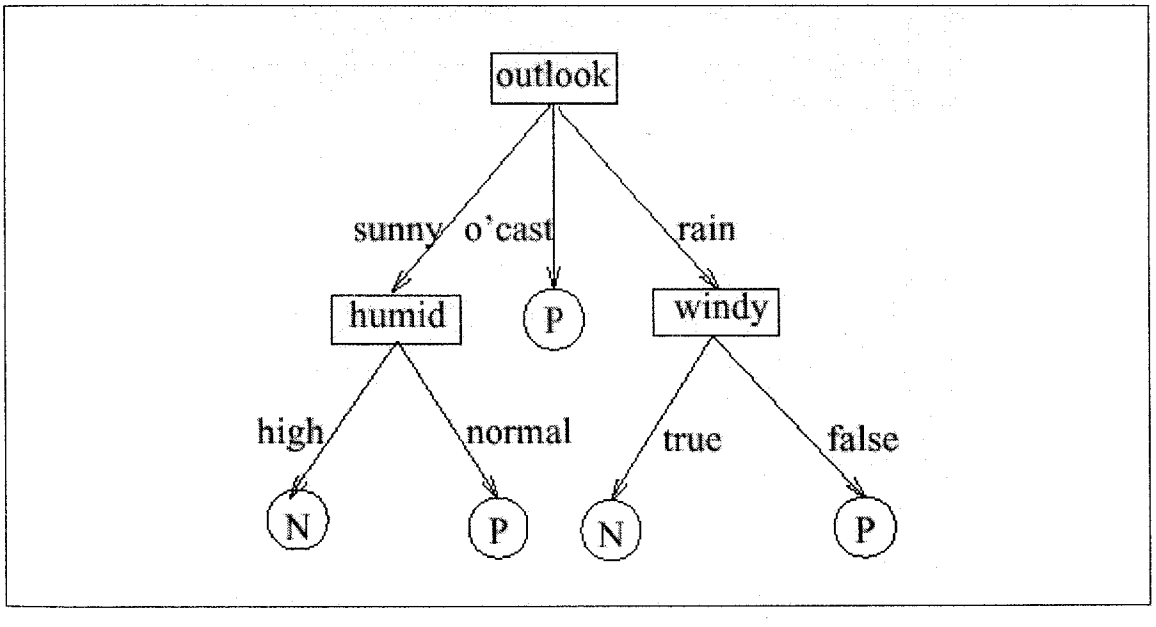
วิธีการสร้างต้นไม้ตัดสินใจเชิงอุปนัย มีขั้นตอนโดยทั่วไปดังนี้

(1) เลือกแอททริบิวต์ (attribute) ที่ทำหน้าที่เป็นโหนดรากของต้นไม้ (root node)

(2) สร้างเส้นทางเชื่อมจากโหนดรากไปยังโหนดลูก จำนวนเส้นทางเชื่อมจะเท่ากับจำนวนค่าที่เป็นไปได้ทั้งหมดของแอททริบิวต์ที่เป็นโหดนราก

(3) ถ้าโหนดลูก เป็นกลุ่มของข้อมูลที่อยู่ในคลาสเดียวกันทั้งหมด ให้หยุดการสร้างต้นไม้ แต่ถ้าโหนดลูกมีข้อมูลของหลายคลาสปะปนกันอยู่ ต้องสร้าง subtree เพื่อจำแนกข้อมูลต่อไป โดยเลือกแอททริบิวต์มาทำหน้าที่เป็นโหนดรากของ subtree และทำซ้ำในขั้นตอนที่ 2 และ 3

การเลือกแอททริบิวต์เพื่อใช้เป็นโหนดในโครงสร้างต้นไม้ตัดสินใจมีความสำคัญ และให้ผลลัพธ์ที่แตกต่างกัน จากข้อมูลในตารางที่ 1.1 ถ้าเปลี่ยนแอททริบิวต์ที่ทำหน้าที่เป็นโหนดราก จาก temperature เป็น outlook จะได้โครงสร้างต้นไม้ที่ซับซ้อนน้อยลง ดังในรูปที่ 1.4 (Quinlan 1993)



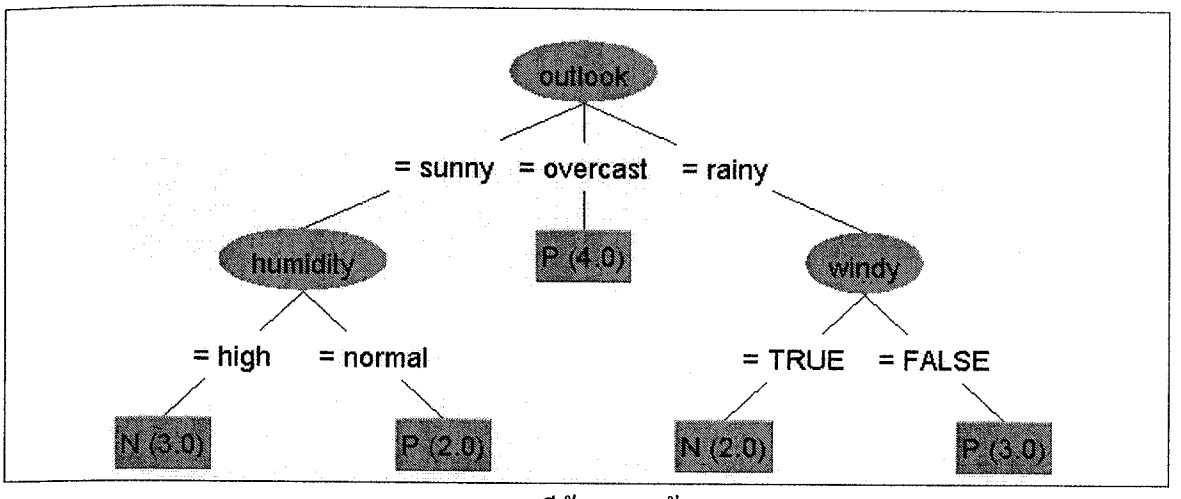
รูปที่ 1.4 โครงสร้างต้นไม้ตัดสินใจที่เปลี่ยนโหนดรากเป็นแอททริบิวต์ outlook

ต้นไม้ตัดสินใจดังในรูปที่ 1.3 และ 1.4 สร้างมาจากข้อมูลสมมุติที่มีข้อมูลครบถ้วนและถูกต้อง แต่การนำเทคนิคการวิเคราะห์ข้อมูลอัตโนมัติแบบนี้ไปใช้กับข้อมูลจริง มักจะพบกับปัญหาข้อมูลผิดเพี้ยน ซึ่งอาจจะเกิดจากการบันทึกข้อมูลผิดหรือเหตุผิดพลาดอื่นๆ ข้อมูลที่ผิดพลาดแบบนี้ เรียกว่า **ข้อมูลรบกวน** (noise) ผลกระทบที่สำคัญจากข้อมูลรบกวน คือ ต้นไม้ตัดสินใจที่สังเคราะห์ขึ้นจะวิเคราะห์ข้อมูลผิดพลาด และอาจจะสร้างต้นไม้ที่มีขนาดใหญ่และ

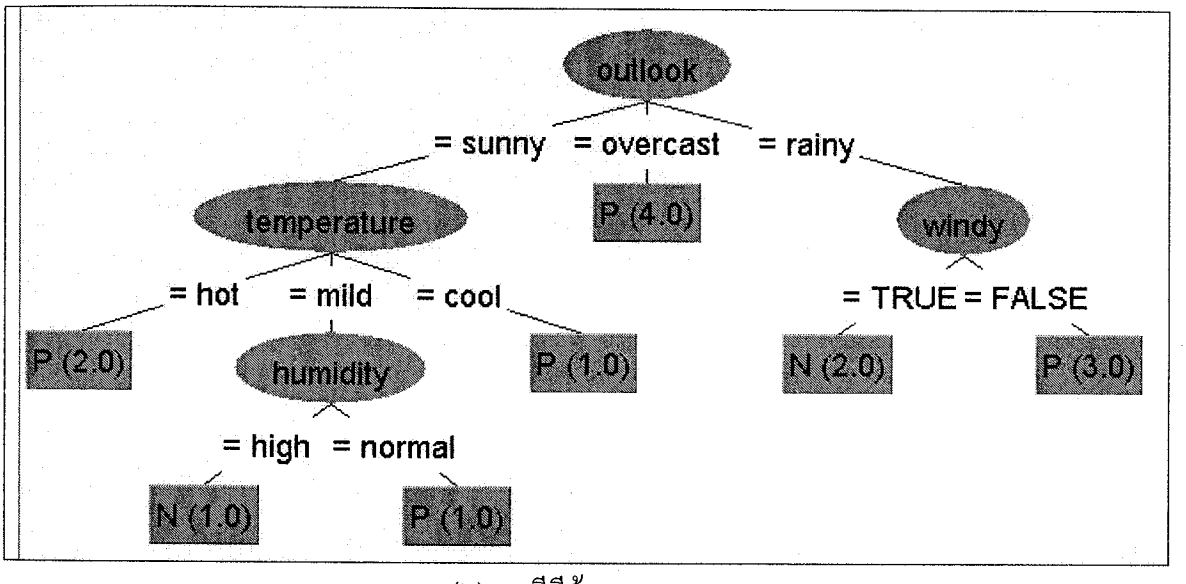
ซับซ้อนเกินไป เนื่องมาจากการพยายามที่จะขยายโครงสร้างให้สามารถอธิบายข้อมูลรบกวนเหล่านั้นให้ได้ ข้อมูลในตารางที่ 1.2 แสดงตัวอย่างของการเกิดข้อมูลรบกวน โดยข้อมูลการพิจารณาสภาพอากาศเพื่อประกอบการตัดสินใจเล่นกอล์ฟ ในข้อมูลรายการที่หนึ่งและสอง ค่าของคลาสที่ถูกต้องจะต้องเป็นค่า N แต่เกิดความผิดพลาดของข้อมูลทำให้ค่าของคลาสเปลี่ยนเป็น P (ค่าที่ล้อมรอบด้วยวงรีเส้นประ) ผลที่เกิดขึ้นจากข้อมูลรบกวนเพียงสองค่านี้คือ โมเดลในลักษณะของต้นไม้ตัดสินใจจะผิดไปจากที่ควรจะเป็น แสดงได้ดังรูปที่ 1.5 ที่เป็นภาพโครงสร้างต้นไม้เปรียบเทียบระหว่างการสร้างโครงสร้างต้นไม้จากข้อมูลที่ถูกต้อง กับการสร้างโครงสร้างต้นไม้ที่มีข้อมูลรบกวนในสองรายการแรก จากภาพจะเห็นได้ว่ากรณีมีข้อมูลรบกวน (รูปที่ 1.5b) ต้นไม้จะมีความลึกมากขึ้น มีกิ่งเพิ่มขึ้น และมีจำนวนโหนดโดยรวมมากขึ้น ซึ่งจะทำให้การนำโมเดลนี้ไปใช้เพื่อการตัดสินคลาสหรือประเภทของข้อมูลในอนาคตไม่สะดวก และการใช้งานไม่ง่ายเท่าโมเดลที่ซับซ้อนน้อยกว่า (ดังรูปที่ 1.5a)

ตารางที่ 1.2 ข้อมูลฝึกที่มีข้อมูลรบกวนในแอททริบิวต์คลาสของรายการที่หนึ่งและสอง

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	P
2	sunny	hot	high	true	P
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N



(a) กรณีข้อมูลถูกต้อง



(b) กรณีมีข้อมูลรบกวน

รูปที่ 1.5 เปรียบเทียบโครงสร้างต้นไม้ตัดสินใจกรณีที่ข้อมูลถูกต้องและกรณีที่มีข้อมูลรบกวน

นอกจากนี้ในกรณีที่ข้อมูลมีขนาดใหญ่มาก การสร้างต้นไม้ตัดสินใจที่มีขนาดใหญ่เกินไปอาจส่งผลให้โปรแกรมหยุดทำงานได้ (freeze) ซึ่งจะทำให้ไม่สามารถสร้างโมเดลและไม่ปรากฏผลลัพธ์เป็นภาพต้นไม้ตัดสินใจ โปรแกรมวิเคราะห์ข้อมูลที่ดีจึงจะต้องมีเทคนิคที่ตรวจจับและจัดการกับข้อมูลรบกวนได้อย่างมีประสิทธิภาพ นอกจากนั้นยังต้องสามารถรองรับข้อมูลขนาดใหญ่ได้ งานวิจัยนี้จึงเสนอขึ้นเพื่อที่จะพัฒนาเทคนิคในการตรวจจับข้อมูลรบกวน สามารถแยกแยะข้อมูลที่ดีออกจากข้อมูลรบกวน และสร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทำงานกับข้อมูลได้ทั้งข้อมูลปกติและข้อมูลขนาดใหญ่

1.2 วัตถุประสงค์ของการวิจัย

เพื่อพัฒนาโปรแกรมคอมพิวเตอร์สร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทนต่อข้อมูลรบกวน โดยพัฒนาเทคนิคที่จะช่วยให้การสร้างต้นไม้ตัดสินใจเชิงอุปนัย (decision-tree induction) ทำงานกับข้อมูลที่มีข้อมูลรบกวนปะปนอยู่มากได้โดยไม่ทำให้โปรแกรมหยุดทำงานกลางคัน เทคนิคการสร้างต้นไม้ตัดสินใจที่พัฒนาขึ้นนี้จะใช้วิธีลดขนาดข้อมูล (data reduction) โดยตัดทิ้งข้อมูลที่แตกต่างจากข้อมูลส่วนใหญ่ในกลุ่มเพื่อให้โปรแกรมทำงานกับข้อมูลขนาดใหญ่ได้

1.3 ขอบเขตของการวิจัย

โครงการวิจัยนี้มีจุดมุ่งหมายที่จะพัฒนาซอฟต์แวร์สร้างต้นไม้ตัดสินใจเชิงอุปนัย ที่ทนต่อข้อมูลรบกวน โดยเน้นความสนใจไปที่งานวิเคราะห์ข้อมูลประเภท การจำแนกข้อมูล (data classification) โดยจะไม่พิจารณาการทำเหมืองข้อมูลประเภทอื่น เช่น การหาความสัมพันธ์ในข้อมูล (association mining) การจัดกลุ่มข้อมูล (data clustering) ในกระบวนการสร้างต้นไม้ตัดสินใจ โหนดต่างๆ ในต้นไม้ตัดสินใจจะเป็นแอททริบิวต์เดี่ยว เพื่อให้การพิจารณาความซับซ้อนของโครงสร้างต้นไม้สามารถวัดได้จากจำนวนโหนดในต้นไม้ ค่าในแต่ละแอททริบิวต์ของชุดข้อมูลอาจจะปรากฏค่าเป็นจำนวนเลข (numeric) หรือเป็นค่าสัญลักษณ์ (nominal or categorical) ก็ได้ ในกรณีที่แอททริบิวต์เป็นค่าต่อเนื่อง (continuous) หรือตัวเลขที่มีการกระจายของค่าจำนวนมาก ก่อนจะมีการนำเข้าข้อมูลจะต้องมีการจัดช่วงของค่าด้วยวิธีการทำ discretization ซึ่งเป็นขั้นตอนที่อยู่นอกเหนือขอบเขตของงานวิจัยนี้

1.4 ประโยชน์ที่ได้รับจากการวิจัย

ซอฟต์แวร์ที่พัฒนาขึ้นนี้มีจุดมุ่งหมายเพื่อใช้ประโยชน์ในงานวิเคราะห์ข้อมูลอัตโนมัติ สามารถใช้งานได้จริงกับข้อมูลที่รวบรวมจากงานประเภทต่างๆ ถึงแม้ข้อมูลที่รวบรวมมาจะมีข้อมูลที่บันทึกผิดพลาดซึ่งจัดเป็นข้อมูลรบกวนปะปนอยู่ ซอฟต์แวร์นี้ก็ยังสามารถสังเคราะห์โครงสร้างต้นไม้ที่นำไปใช้จำแนกประเภทข้อมูล และโครงสร้างต้นไม้นี้ใช้อธิบาย (description) ลักษณะเด่นของข้อมูลในแต่ละประเภทได้ ผล (output) ที่ได้จากซอฟต์แวร์ จะมุ่งประโยชน์ไปที่การทำนายและคาดหมาย (prediction) ประเภทหรือคลาสของข้อมูลที่จะเกิดขึ้นในอนาคต

ดังนั้นซอฟต์แวร์ที่พัฒนาขึ้นจึงใช้ประโยชน์ได้กับทุกวงการที่เกี่ยวข้องกับงานที่มีการรวบรวมและวิเคราะห์ข้อมูล โดยโปรแกรมที่พัฒนาขึ้นนี้จะช่วยให้งานวิเคราะห์ข้อมูลทำได้รวดเร็วขึ้นและผู้ใช้ไม่จำเป็นต้องเป็นผู้มีความเชี่ยวชาญด้านการวิเคราะห์ข้อมูล ประโยชน์จึงเกิด

กับทุกหน่วยงานที่มีการเก็บรวบรวมข้อมูล และวิเคราะห์ข้อมูล โดยเฉพาะในกรณีที่การรวบรวมข้อมูลใช้วิธีการส่งเจ้าหน้าที่ไปสอบถามและจดบันทึก เช่น การสำรวจสำมะโนประชากร การจดบันทึกอาจเกิดข้อผิดพลาดทำให้มีข้อมูลรบกวนเกิดขึ้น เทคนิคการจัดการกับข้อมูลรบกวนที่พัฒนาขึ้นใช้จะช่วยให้รับมือกับข้อมูลเช่นนี้ได้

นอกจากนี้องค์ความรู้เกี่ยวกับเทคนิคในการจัดการกับข้อมูลรบกวน และวิธีการพัฒนาโปรแกรมสร้างต้นไม้ตัดสินใจที่มีความทนทานต่อข้อมูลรบกวน ยังสามารถเผยแพร่ในวงวิชาการทางด้านการวิเคราะห์ข้อมูลอัตโนมัติและการค้นหาความรู้จากฐานข้อมูล งานวิจัยนี้ยังมีประโยชน์ในด้านการพัฒนาผู้ช่วยวิจัยให้มีความรู้และประสบการณ์ สามารถพัฒนาตนเองเป็นบุคลากรที่มีศักยภาพสูงในด้านการวิจัยในเชิงทฤษฎีข้อมูลและการพัฒนาซอฟต์แวร์ขนาดใหญ่ ซึ่งจะเป็นประโยชน์ต่องานวิจัยและพัฒนาด้านคอมพิวเตอร์ซอฟต์แวร์ของประเทศได้ต่อไป

บทที่ 2
การสร้างต้นไม้ตัดสินใจเชิงอุปนัย

ต้นไม้ตัดสินใจ (decision tree) เป็นโครงสร้างข้อมูลชนิดเป็นลำดับชั้น (hierarchy) ใช้ช่วยให้คำแนะนำในการตัดสินใจ โหนดรากและโหนดภายในเป็นองค์ประกอบที่ใช้ในการพิจารณาตัดสินใจแต่ละขั้น โหนดใบเป็นผลลัพธ์สุดท้ายของการตัดสินใจ โครงสร้างข้อมูลชนิดนี้ง่ายต่อการทำความเข้าใจจึงเป็นที่นิยมใช้มากกว่าโครงสร้างข้อมูลประเภทอื่นๆ การทำเหมืองข้อมูลประเภทการจำแนก (classification) ที่ใช้ข้อมูลเป็นอินพุทให้กับโปรแกรมเพื่อฝึกโปรแกรมให้สามารถสร้างโครงสร้างต้นไม้ตัดสินใจที่อธิบายข้อมูลได้อย่างถูกต้องจึงเรียกว่า การสร้างต้นไม้ตัดสินใจเชิงอุปนัย (decision-tree induction)

2.1 โครงสร้างและการใช้งานต้นไม้ตัดสินใจ

ต้นไม้ตัดสินใจมีโครงสร้างข้อมูลพื้นฐานเป็นลักษณะต้นไม้ (tree) ประกอบด้วยโหนด (node) และกิ่ง (branch) ที่แยกออกจากโหนด โครงสร้างต้นไม้ตัดสินใจจะประกอบด้วยโหนด 3 ประเภท คือ โหนดราก (root node) โหนดภายใน (internal node) และโหนดใบ (leaf node) โหนดรากและโหนดภายในบรรจุชื่อแอททริบิวต์ของข้อมูล ที่ใช้เป็นปัจจัยประกอบการตัดสินใจในแต่ละขั้น โหนดรากเป็นการตัดสินใจขั้นแรก กิ่งที่แยกออกมาจากโหนดจะเป็นแต่ละเส้นทางเลือกของการตัดสินใจในแต่ละขั้น แต่ละกิ่งจะมีค่ากำกับ และจำนวนกิ่งจะขึ้นอยู่กับค่าที่เป็นไปได้ทั้งหมดของแอททริบิวต์ในโหนดนั้น ส่วนของโหนดใบจะแสดงค่าที่เป็นผลลัพธ์ของการตัดสินใจ

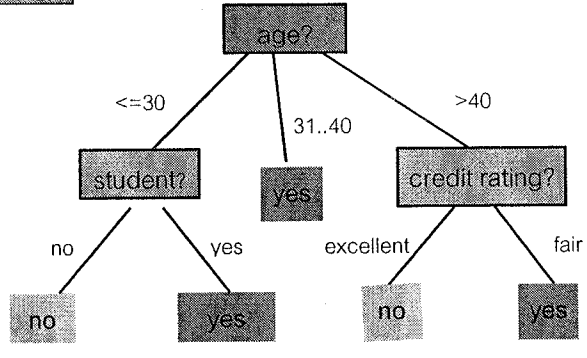
จากตัวอย่างในรูปที่ 2.1 (Han and Kamber, 2006) ที่แสดงต้นไม้ตัดสินใจของวัตถุประสงค์การกู้เงินของลูกค้าสองประเภท (หรือเรียกว่าสองคลาส) คือลูกค้าที่กู้เงินเพื่อซื้อคอมพิวเตอร์ (buys computer = yes) และลูกค้าที่กู้เงินเพื่อซื้อสินค้าอื่นที่ไม่ใช่คอมพิวเตอร์ (buys computer = no) ดังนั้นเป้าหมายของการสร้างต้นไม้ตัดสินใจในตัวอย่างนี้คือ ต้องการหาโมเดลเพื่อการจำแนกประเภท (classification) ของลูกค้าเงินกู้ โดยจะมีผลลัพธ์ของการตัดสินใจเป็นหนึ่งในสองค่าที่เป็นไปได้ คือ yes หรือ no และสองค่านี้จะปรากฏในโหนดใบของต้นไม้ตัดสินใจ

ในการเริ่มสร้างต้นไม้ตัดสินใจถ้าเลือกใช้แอททริบิวต์ age เป็นปัจจัยแรก (นั่นคือ age จะเป็นโหนดราก) ในการตัดสินใจว่าลูกค้ารายนั้นจะกู้เงินเพื่อซื้อคอมพิวเตอร์หรือไม่ จำนวนกิ่งที่แตกออกมาจากโหนดนี้จะมีจำนวนสามกิ่ง เท่ากับค่าที่เป็นไปได้สามค่าหรือสามกรณีของ age คือ

กรณีที่ลูกค้ามีอายุน้อยกว่าหรือเท่ากับ 30 กรณีที่ลูกค้าอายุอยู่ระหว่าง 31 ถึง 40 ปี และกรณีที่ลูกค้าอายุมากกว่า 40 ปี

ในกรณีที่ลูกค้าอายุอยู่ระหว่าง 31 ถึง 40 ปีสามารถจำแนกหรือตัดสินใจได้ทันทีว่าลูกค้ารายนั้นจะกู้เงินเพื่อซื้อคอมพิวเตอร์ (คลาส yes) ในกรณีที่ลูกค้ามีอายุน้อยกว่าหรือเท่ากับ 30 ปียังจำแนกคลาสทันทีไม่ได้ จะต้องพิจารณาเงื่อนไขต่อไปว่าลูกค้ารายนั้นยังเป็นนักศึกษาหรือไม่ ถ้าไม่ใช่เขาจะกู้เงินเพื่อซื้อสินค้าอื่นที่ไม่ใช่คอมพิวเตอร์ (คลาส no) แต่ถ้าใช่เขาจะกู้เงินเพื่อซื้อคอมพิวเตอร์ ส่วนกรณีที่ลูกค้าอายุมากกว่า 40 ปีจะต้องพิจารณาจากระดับของเครดิต ถ้าเครดิตดีเยี่ยมเขาจะกู้เงินเพื่อซื้อสินค้าอื่นที่ไม่ใช่คอมพิวเตอร์ แต่ถ้าเครดิตอยู่ในระดับปานกลางเขาจะกู้เงินเพื่อซื้อคอมพิวเตอร์

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



รูปที่ 2.1 ต้นไม้ตัดสินใจอธิบายรูปแบบของลูกค้าสองประเภทที่กู้เงินเพื่อซื้อ/ไม่ซื้อคอมพิวเตอร์

ต้นไม้ตัดสินใจที่สร้างขึ้นมีวัตถุประสงค์เพื่อให้ทำหน้าที่เป็นโมเดลอธิบายรูปแบบของข้อมูลในแต่ละคลาส เรียกว่า โมเดลเพื่อการอธิบาย (descriptive model) การใช้โมเดลเพื่อเป็นตัวแทนของข้อมูล จะช่วยให้การทำความเข้าใจกับข้อมูลทำได้ง่ายกว่าการพิจารณาจากตัวข้อมูลโดยตรง นอกจากประโยชน์ในด้านการเป็นตัวแทนอธิบายข้อมูลแล้วต้นไม้ตัดสินใจยังสามารถใช้เป็น โมเดลเพื่อการทำนาย (predictive model) เช่นถ้ามีลูกค้ารายใหม่อายุ 35 ปีแจ้งความจำนงขอกู้เงิน จากโมเดลในรูปที่ 2.1 สามารถทำนายได้ทันทีว่าลูกค้ารายนี้ขอกู้เงินเพื่อซื้อคอมพิวเตอร์ หรือ

ถ้ามีลูกค้าอายุ 25 ปีและมีสถานภาพเป็นนักศึกษา ก็สามารถทำนายได้ว่าลูกค้ารายนี้ขอกู้เงินเพื่อซื้อคอมพิวเตอร์

2.2 ขั้นตอนการสร้างต้นไม้ตัดสินใจจากข้อมูลฝึก

การสร้างต้นไม้ตัดสินใจเชิงอุปนัยจะเกิดขึ้นได้เมื่อมีข้อมูล และข้อมูลที่ใช้นี้เรียกว่าข้อมูลฝึก (training data) ข้อมูลนี้มีลักษณะเป็นเรคคอร์ด ในหนึ่งเรคคอร์ดจะประกอบขึ้นจากหลายแอททริบิวต์ที่ใช้บรรยายลักษณะของเรคคอร์ดนั้น เช่นข้อมูลลูกค้าเงินกู้ในรูปที่ 2.1 ข้อมูลของลูกค้าแต่ละคนจะบรรยายด้วยห้าลักษณะหรือห้าแอททริบิวต์ ได้แก่ age, income, student, credit rating, buys computer การสร้างต้นไม้ตัดสินใจเพื่อให้ทำหน้าที่จำแนกข้อมูลออกเป็นแต่ละคลาส จะต้องมีการระบุเป้าหมายว่าในข้อมูลฝึกนั้นจะกำหนดให้แอททริบิวต์ใดเป็นเป้าหมายของการจำแนกคลาสของข้อมูล ในตัวอย่างตามรูปที่ 2.1 กำหนดให้ buys computer เป็นเป้าหมายของการจำแนก แอททริบิวต์เป้าหมายจะเรียกว่า goal attribute หรือ class ดังนั้นแอททริบิวต์ที่เหลือ คือ age, income, student, credit rating จะทำหน้าที่เป็นแอททริบิวต์เพื่อสร้างต้นไม้ตัดสินใจเรียกว่า predictive attributes ในงานจำแนกข้อมูลมีข้อกำหนดว่า แอททริบิวต์เป้าหมายหรือคลาสจะต้องมีค่าของข้อมูลเป็นชนิดข้อความ (nominal or categorical attribute) เท่านั้น ส่วนแอททริบิวต์อื่นที่ใช้เพื่อการตัดสินใจมีค่าเป็นข้อความหรือตัวเลข (numeric) ก็ได้ ในกรณีที่เป็นตัวเลขและมีค่าต่อเนื่อง (continuous) หรือเป็นค่าที่ไม่ต่อเนื่อง (discrete) แต่มีค่าที่แตกต่างกันเป็นจำนวนมาก เช่น ค่าของอายุที่มีค่าตั้งแต่ 15, 16, 17, ..., 59, 60 จะต้องมีการจัดช่วงของค่าเรียกว่า ดิสครีไทเซชัน (discretization) เช่นแบ่งช่วงอายุเป็น 15-20, 21-30, 31-40, 41-50, 51-60 เป็นต้น

กระบวนการในการสร้างต้นไม้ตัดสินใจจากข้อมูลฝึก มีลักษณะเป็นการทำงานซ้ำและใช้เทคนิคกรีดี (greedy) โดยในแต่ละรอบของการทำงานจะเลือกสร้างต้นไม้ที่จำแนกข้อมูลที่ดีที่สุด ณ ขณะนั้น ขั้นตอนการทำงานอธิบายได้ดังนี้

(1) ถ้าข้อมูลที่มีอยู่ขณะนั้นเป็นคลาสเดียวกันทั้งหมด หยุดการสร้างต้นไม้ และแสดงภาพโหนดใบของต้นไม้ที่มีชื่อคลาสของข้อมูลกลุ่มนี้เป็นชื่อของโหนดใบ

(2) ถ้าข้อมูลมีหลายคลาสปะปนกันอยู่ ดำเนินการแยกข้อมูลด้วยการเลือกแอททริบิวต์เพื่อทำหน้าที่เป็นโหนดตัดสินใจ (decision node) ของต้นไม้ ข้อมูลจะถูกกระจายไปในแต่ละกิ่งตามค่าของแอททริบิวต์ที่ถูกเลือกเป็นโหนดตัดสินใจ จำนวนกิ่งจะเท่ากับจำนวนค่าที่เป็นไปได้ทั้งหมดของแอททริบิวต์นั้น ทำซ้ำในขั้นตอนที่ 1 และ 2 กับกลุ่มข้อมูลโหนดลูกแต่ละโหนด

ปัญหาที่ต้องพิจารณาในการสร้างต้นไม้ตัดสินใจ คือ ในแต่ละขั้นตอนของการสร้างต้นไม้ควรจะเลือกแอททริบิวต์ใดมาทำหน้าที่เป็นโหนดตัดสินใจ วิธีการที่นิยมใช้เพื่อการพิจารณาเลือกแอททริบิวต์ คือ ทดลองเลือกแต่ละแอททริบิวต์ มาทำหน้าที่เป็นโหนดตัดสินใจ แล้ววัดค่า gain ซึ่งเป็นค่าที่ชี้ว่าแอททริบิวต์นั้นจะช่วยจำแนกคลาสของข้อมูลได้ดีเพียงใด แอททริบิวต์ที่ให้ค่า gain สูง คือแอททริบิวต์ที่แยกข้อมูลแล้วได้กลุ่มข้อมูลในแต่ละโหนดลูกเป็นคลาสเดียวกันทั้งหมด หรือมีข้อมูลต่างคลาสปะปนมาบ้างเพียงเล็กน้อย ค่า gain ที่สูงที่สุด หมายถึง การจำแนกคลาสที่ดีที่สุด นั่นคือให้โหนดลูกที่เป็นข้อมูลคลาสเดียวกันทั้งหมด

ถ้าให้ T แทนเซตของข้อมูลฝึก และ X แทนแอททริบิวต์ที่ถูกเลือกให้เป็นตัวตรวจสอบเพื่อแยกกลุ่มข้อมูล ค่า gain คำนวณได้ดังนี้

$$\text{gain (X)} = \text{info(T)} - \text{info}_X\text{ (T)}$$

เมื่อ info(T) คือ ฟังก์ชันที่ระบุปริมาณข้อมูลที่ต้องการเพื่อให้สามารถจำแนกคลาสของข้อมูลได้

info_X (T) คือ ฟังก์ชันที่ระบุปริมาณข้อมูลที่ต้องการเพื่อการจำแนกคลาสของข้อมูล โดยใช้แอททริบิวต์ X เป็นตัวตรวจสอบเพื่อจำแนกกลุ่มของข้อมูล

ค่าของ info(T) และ info_X (T) มีหน่วยเป็นบิต คำนวณได้จากสูตรต่อไปนี้

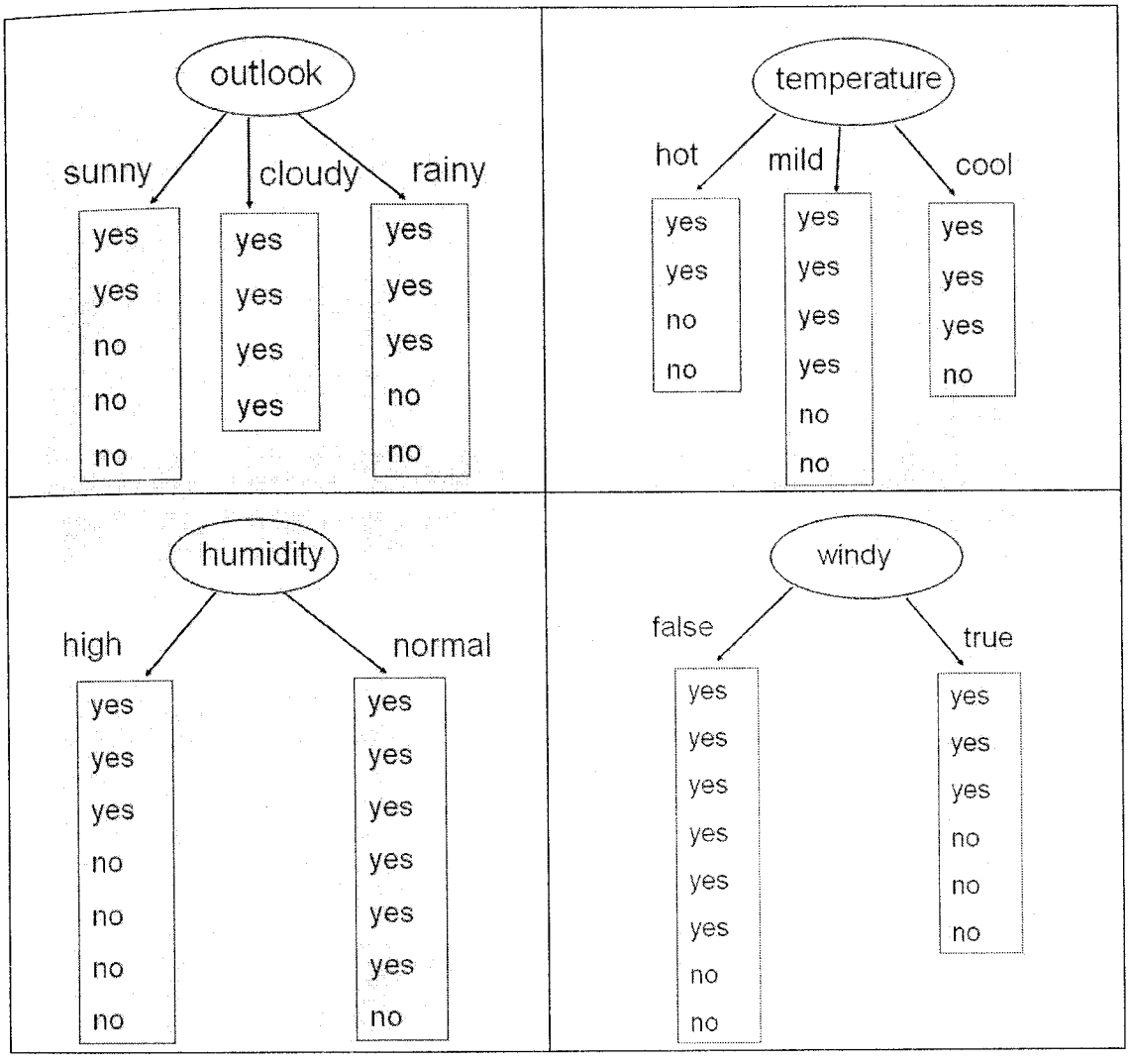
$$\text{info(T)} = -\sum_{j=1}^{k} \frac{freq(C_j,T)}{|T|} \times log_2\left(\frac{freq(C_j,T)}{|T|}\right) \quad \text{bits}$$

โดยที่ |T| คือ จำนวนข้อมูลทั้งหมดในเซตของข้อมูลฝึก และ freq (C_j, T) คือ ความถี่ที่ข้อมูลใน T ปรากฏเป็นคลาส C_j

$$\text{info}_X\text{ (T)} = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times inf\,o(T_i) \quad \text{bits}$$

เมื่อ i คือ จำนวนค่าที่เป็นไปได้ของแอททริบิวต์ X และ $|C_i|$ คือ จำนวนข้อมูลที่มีค่า X = i

การคำนวณค่า gain อธิบายได้ด้วยตัวอย่างข้อมูลตามตารางที่ 2.1 (Quinlan 1993) ที่เป็นข้อมูลการตัดสินใจเล่น/ไม่เล่นกอล์ฟโดยพิจารณาจากสภาพอากาศ ข้อมูลที่ระบุสภาพอากาศประกอบด้วย 4 แอททริบิวต์คือ outlook, temperature, humidity และ windy โดยมีแอททริบิวต์ play เป็นเป้าหมายของการทำ classification ดังนั้นมี 4 แอททริบิวต์ที่มีโอกาสทำหน้าที่เป็นโหนดรากได้ แสดงการแยกกลุ่มข้อมูลของแต่ละแอททริบิวต์ได้ดังรูปที่ 2.2



รูปที่ 2.2 การจำแนกข้อมูลของแต่ละแอททริบิวต์ที่ทำหน้าที่เป็นโหนดตัดสินใจ

ตารางที่ 2.1 ข้อมูลสภาพอากาศประกอบการตัดสินใจเล่น/ไม่เล่นกอล์ฟ

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	No
sunny	hot	high	true	No
cloudy	hot	high	false	Yes
rainy	mild	high	false	Yes
rainy	cool	normal	false	Yes
rainy	cool	normal	true	No
cloudy	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rainy	mild	normal	false	Yes
sunny	mild	normal	true	Yes
cloudy	mild	high	true	Yes
cloudy	hot	normal	false	Yes
rainy	mild	high	true	No

จากตัวอย่างข้อมูลในตารางที่ 2.1 เซตของข้อมูลฝึก T ประกอบด้วยข้อมูล 2 คลาส คือ play = yes และ play = no การจะระบุว่าข้อมูลหนึ่งเรคคอร์ดอยู่ในคลาส yes หรือ no ต้องการปริมาณข้อมูลประกอบการตัดสินใจจำแนกคลาสดังนี้

info(play) = [- (ความถี่ของการปรากฏข้อมูลเป็นคลาส yes / จำนวนข้อมูลทั้งหมด)
x $\log_2$ (ความถี่ของการปรากฏข้อมูลเป็นคลาส yes / จำนวนข้อมูลทั้งหมด)]
+ [- (ความถี่ของการปรากฏข้อมูลเป็นคลาส no / จำนวนข้อมูลทั้งหมด)
x $\log_2$ (ความถี่ของการปรากฏข้อมูลเป็นคลาส no / จำนวนข้อมูลทั้งหมด)]

$= - (9/14) \times \log_2 (9/14) - (5/14) \times \log_2 (5/14)$

$= 0.940$ bits

การจะสร้างต้นไม้ตัดสินใจเพื่อจำแนกคลาสของข้อมูลออกเป็น play = yes หรือ play = no ต้องใช้ข้อมูลจากแอททริบิวต์มาสร้างโหนดประกอบการตัดสินใจ ถ้าเลือกแอททริบิวต์ outlook จะต้องการปริมาณข้อมูลเพิ่มเพื่อประกอบการเลือกคลาส ดังนี้

$$\text{info}_{\text{outlook}}(T) = (5/14) \times [- (2/5) \times \log_2(2/5) - (3/5) \times \log_2(3/5)]$$
$$+ (4/14) \times [- (4/4) \times \log_2(4/4) - (0/4) \times \log_2(0/4)]$$
$$+ (5/14) \times [- (3/5) \times \log_2(3/5) - (2/5) \times \log_2(2/5)]$$
$$= 0.693 \text{ bits}$$

ค่า info(T) นี้เรียกได้อีกอย่างว่า ค่า *entropy* ซึ่งเป็นค่าของ impurity หมายถึงความไม่บริสุทธ์ของกลุ่มข้อมูล หรือการคละกันของข้อมูลต่างคลาส ถ้ามีค่าสูงแสดงว่าข้อมูลต่างคลาสปะปนกันมากทำให้การจำแนกข้อมูลไม่ดี จากค่า info ของแอททริบิวต์ outlook สามารถคำนวณค่า gain ได้ดังนี้

$$\text{gain (outlook)} = \text{info(play)} - \text{info}_{\text{outlook}}(T)$$
$$= 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

นั่นคือ ถ้ามีข้อมูลใหม่เข้ามา เมื่อพิจารณาจากค่า outlook ของข้อมูลใหม่นี้ จะต้องใช้ข้อมูลเพิ่มอีก 0.693 bits จึงจะบอกคลาสที่ถูกต้องของข้อมูลใหม่นี้ได้ ดังนั้น gain หรือประโยชน์ที่ได้รับจากการพิจารณาแอททริบิวต์ outlook คือ 0.247 บิต ค่า gain ของอีกสามแอททริบิวต์ได้แก่ temperature, humidity และ windy คำนวณได้ดังนี้

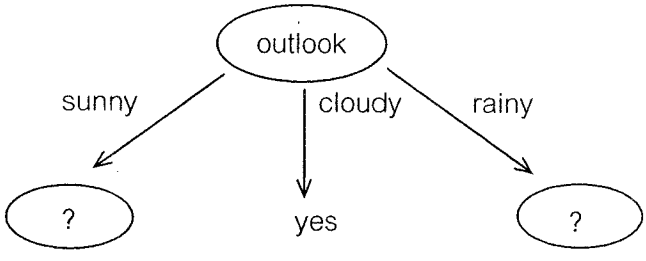
$$\text{gain (temperature)} = \text{info (T)} - \text{info}_{\text{temperature}}(T)$$
$$= 0.940 - 0.911$$
$$= 0.029 \text{ bits}$$

$$\text{gain (humidity)} = \text{info (T)} - \text{info}_{\text{humidity}}(T)$$
$$= 0.940 - 0.788$$
$$= 0.152 \text{ bits}$$

$$\text{gain (windy)} = \text{info (T)} - \text{info}_{\text{windy}}(T)$$
$$= 0.940 - 0.892$$
$$= 0.048 \text{ bits}$$

แอททริบิวต์ที่ให้ค่า gain สูงที่สุด คือ outlook ดังนั้นแอททริบิวต์ outlook จึงถูกเลือกเป็นโหนดรากของต้นไม้ตัดสินใจ แต่เนื่องจากแอททริบิวต์ outlook ยังไม่สามารถจัดกลุ่มข้อมูลให้เป็นคลาสเดียวกันทั้งหมด (พิจารณาได้จากรูปที่ 2.2 ที่กรณี outlook = sunny จัดกลุ่มข้อมูลที่เป็นคลาส yes จำนวน 2 เรคคอร์ด และคลาส no จำนวน 3 เรคคอร์ด กรณี outlook = cloudy จัดข้อมูลได้เป็นคลาส yes ทั้งสี่เรคคอร์ด และ outlook = rainy จัดกลุ่มข้อมูลที่เป็นคลาส yes จำนวน 3 เรคคอร์ด และคลาส no จำนวน 2 เรคคอร์ด) จึงต้องสร้างต้นไม้ตัดสินใจในระดับต่อไปเพื่อให้จำแนกข้อมูลคลาส yes ออกจากคลาส no ได้สมบูรณ์

ขั้นตอนในลำดับต่อไปจะพิจารณาเลือกแอททริบิวต์ที่จะมาเป็นโหนดในระดับที่ 2 ต่อจากโหนดราก ในกรณี outlook = cloudy ไม่จำเป็นต้องสร้างโหนดเพิ่มเติม เนื่องจากสามารถจำแนกกลุ่มข้อมูลที่เป็นคลาส yes ได้ทั้งหมด (รูปที่ 2.3)



รูปที่ 2.3 โครงสร้างต้นไม้ตัดสินใจหลังการเลือกโหนดราก

แอททริบิวต์ที่สามารถถูกเลือกเป็นโหนดในระดับที่ 2 ประกอบด้วย temperature, humidity และ windy (แอททริบิวต์ outlook จะไม่ถูกใช้อีก เพราะสภาพอากาศจะไม่มีโอกาสเกิดเหตุการณ์ outlook = sunny AND outlook = cloudy)

พิจารณาการสร้างโหนดลูกทางด้านซ้ายมือ (outlook = sunny) ถ้าเลือกแอททริบิวต์ temperature จะคำนวณค่า gain ได้ดังนี้

$$\text{gain (temperature)} = \text{info (outlook = sunny)} - \text{info}_{\text{temperature}}\text{ (outlook = sunny)}$$

เนื่องจาก outlook = sunny จัดกลุ่มข้อมูลที่เป็นคลาส yes 2 เรคคอร์ด และข้อมูลที่เป็นคลาส no 3 เรคคอร์ด ดังนั้น

$$\text{info (outlook = sunny)} = -\frac{2}{5}\times\log_2\left(\frac{2}{5}\right)-\frac{3}{5}\times\log_2\left(\frac{3}{5}\right)$$

$$= 0.971 \text{ bits}$$

$$\text{info}_{\text{temperature}}\text{(outlook = sunny)} = \text{info ([0,2] , [1, 1] , [1, 0])}$$

$$= \frac{2}{5}\times\left[-\frac{0}{2}\times\log_2\left(\frac{0}{2}\right)-\frac{2}{2}\times\log_2\left(\frac{2}{2}\right)\right]$$

$$+ \frac{2}{5}\times\left[-\frac{1}{2}\times\log_2\left(\frac{1}{2}\right)-\frac{1}{2}\times\log_2\left(\frac{1}{2}\right)\right]$$

$$+ \frac{1}{5}\times\left[-\frac{1}{1}\times\log_2\left(\frac{1}{1}\right)-\frac{0}{1}\times\log_2\left(\frac{0}{1}\right)\right]$$

$$= 0.4 \text{ bits}$$

$\therefore$ gain (temperature) = 0.971 – 0.4 bits

= 0.571 bits

เมื่อ outlook = sunny แล้วทดลองจำแนกกลุ่มข้อมูลต่อไปด้วยแอททริบิวต์ humidity และ windy จะคำนวณค่า gain ได้ดังนี้ (แสดงภาพการเปรียบเทียบค่า gain ได้ดังรูปที่ 2.4)

gain (humidity) = info (outlook = sunny) – $info_{humidity}$ (outlook = sunny)

= 0.971 – info ([0,3] , [2, 0])

= 0.971 – 0 bits

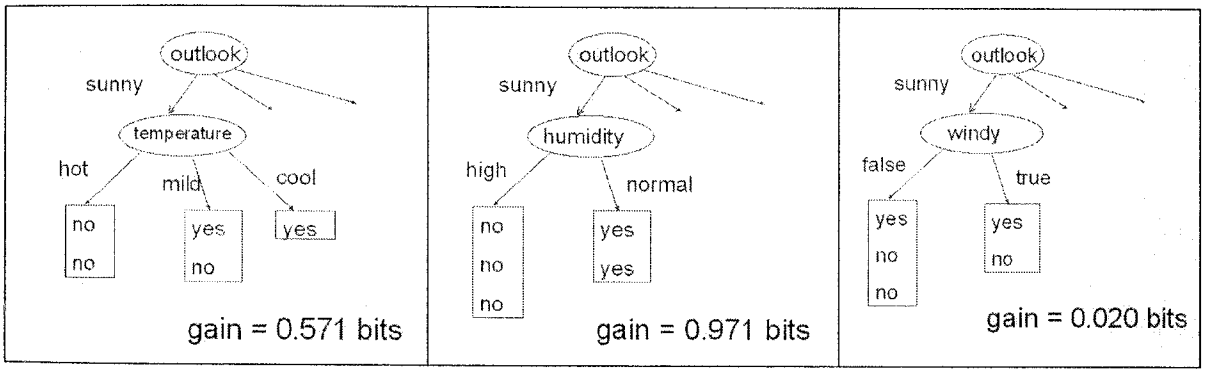
= 0.971 bits

gain (windy) = info (outlook = sunny) – $info_{windy}$ (outlook = sunny)

= 0.971 – info ([1,2] , [1, 1])

= 0.971 – 0.951 bits

= 0.020 bits

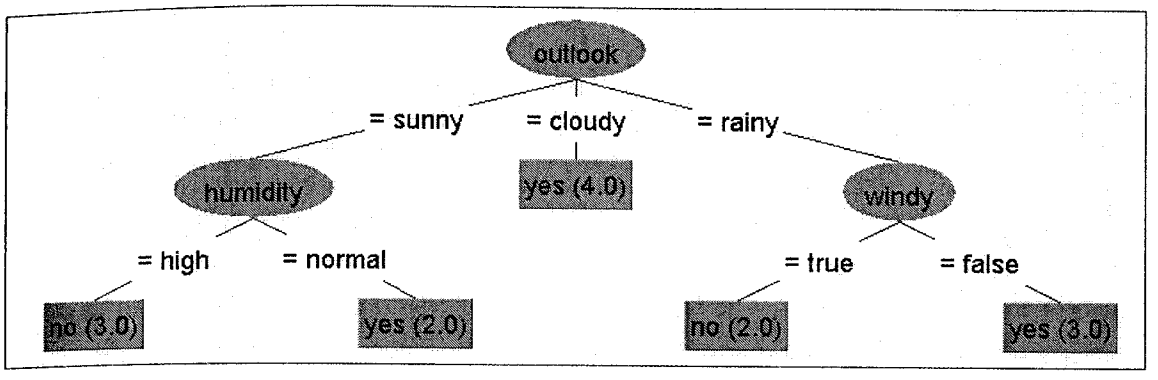


รูปที่ 2.4 เปรียบเทียบค่า gain ของแอททริบิวต์ในระดับที่สอง

จากการเปรียบเทียบค่า gain ของโหนดในระดับที่สองต่อจากกรณี outlook = sunny พบว่าแอททริบิวต์ humidity ให้ค่าสูงที่สุด จึงพิจารณาเลือกแอททริบิวต์ humidity เป็นโหนดในระดับที่สองต่อจากโหนด outlook

ในการพิจารณากิ่งทางขวาของโหนด outlook ที่ต้องพิจารณาเลือกแอททริบิวต์ และจากวิธีการคำนวณค่า gain ที่แสดงด้วยตัวอย่างก่อนหน้านี้ สามารถเลือกได้ว่าแอททริบิวต์ windy จะให้ค่า gain ที่สูงที่สุด (จากกลุ่มของแอททริบิวต์ temperature, humidity และ windy)

กระบวนการสร้างต้นไม้ตัดสินใจจะสิ้นสุดเมื่อโหนดใบ เป็นกลุ่มของข้อมูลคลาสเดียวกันทั้งหมด ซึ่งแสดงต้นไม้ตัดสินใจที่สมบูรณ์ได้ดังรูปที่ 2.5



รูปที่ 2.5 ต้นไม้ตัดสินใจที่จำแนกข้อมูลได้สมบูรณ์

ในกรณีที่มีข้อมูลใหม่ที่ยังไม่ทราบคลาส

"outlook = sunny, temperature = cool, humidity = high, windy = true"

สามารถใช้ต้นไม้ตัดสินใจนี้ทำนายคลาสของข้อมูลได้ว่า play = no โดยพิจารณาจากเพียงสองแอททริบิวต์ คือ outlook = sunny และ humidity = high

ต้นไม้ตัดสินใจสามารถแปลงเป็นกฎเรียกว่า กฎการจำแนกข้อมูล (classification rules) กฎนี้จะอยู่ในลักษณะของการทำนาย ถ้า..*เงื่อนไข*..แล้ว..*ผลการทำนาย*.. หรือ IF ... THEN ... การแสดงเงื่อนไขของกฎจะเริ่มจากส่วนของโหนดราก แอททริบิวต์ในลำดับถัดไปจะเป็นเงื่อนไขที่นำมา AND กับเงื่อนไขในส่วนโหนดราก โครงสร้างของโหนดที่อยู่ในแนวกิ่งเดียวกันจะเป็นกฎข้อเดียวกัน จำนวนของกฎจะเท่ากับจำนวนของโหนดใบ และจากตัวอย่างในรูปที่ 2.5 แสดงกฎการจำแนกข้อมูลได้ดังนี้

rule 1 : IF (outlook = sunny) AND (humidity = high) THEN play = no

rule 2 : IF (outlook = sunny) AND (humidity = normal) THEN play = yes

rule 3 : IF (outlook = cloudy) THEN play = yes

rule 4 : IF (outlook = rainy) AND (windy = false) THEN play = yes

rule 5 : IF (outlook = rainy) AND (windy = true) THEN play = no

2.3 การตรวจสอบประสิทธิภาพของต้นไม้ตัดสินใจ

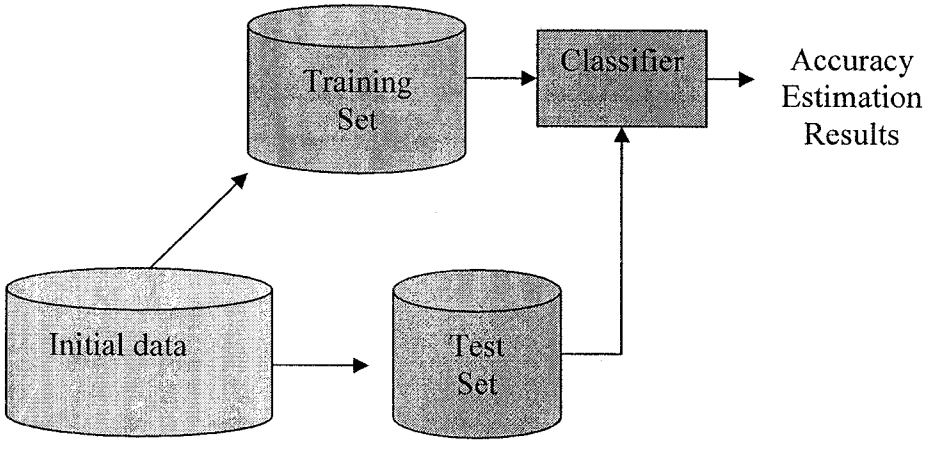
การทำเหมืองข้อมูลประเภทการจำแนก (classification) เป็นการสร้างตัวจำแนกหรือ classifier เพื่อใช้เป็นโมเดลในการอธิบายลักษณะข้อมูลในแต่ละคลาส หรือใช้ทำนายประเภท (class) ของข้อมูลที่จะเกิดขึ้นในอนาคต ในกรณีของการจำแนกข้อมูลด้วยต้นไม้ตัดสินใจ ตัวจำแนกหรือโมเดลจะมีลักษณะเป็นโครงสร้างต้นไม้ การจำแนกข้อมูลด้วยเทคนิคอื่นเช่นใช้โครงข่ายประสาทเทียม ใช้ทฤษฎีทางสถิติหรือใช้เทคนิคอื่นๆ โมเดลอาจจะอยู่ในลักษณะที่ต่าง

ออกไปเช่น เป็นสมการเชิงเส้น สมการโพลิโนเมียล แต่ไม่ว่าโมเดลที่ได้จะถูกแสดงผลในรูปแบบใดโมเดลที่ใช้ประโยชน์ได้ดีจะต้องมีความถูกต้องในการอธิบายรูปแบบข้อมูล และมีความแม่นตรง (accurate) ในการทำนายสูง วิธีการที่ใช้วัดความแม่นตรงของโมเดลมีหลายวิธีดังนี้

วิธี Holdout

วิธีการนี้จะแบ่งข้อมูลออกเป็นสองส่วน ส่วนแรกเรียกว่า ข้อมูลฝึก (training data) จะมีจำนวนประมาณสองในสาม หรือประมาณ 66% ของข้อมูลทั้งหมด ส่วนที่สองเรียกว่า ข้อมูลทดสอบ (test data) มีจำนวนประมาณหนึ่งในสาม หรือ 34% ของข้อมูลทั้งหมด

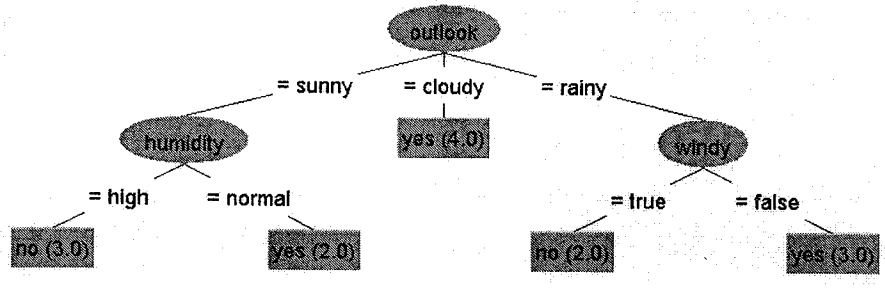
ข้อมูลฝึกจะถูกส่งเป็น input ให้อัลกอริทึม classification เพื่อใช้สร้างโมเดลของข้อมูลที่เรียกว่า classifier จากนั้นจะใช้ข้อมูลทดสอบวัดความถูกต้องในการจำแนกคลาสของ classifier วิธีการวัดความแม่นตรงนี้แสดงเป็นแผนภาพได้ดังรูปที่ 2.6



รูปที่ 2.6 การวัดความแม่นตรงแบบ Holdout

ข้อมูลที่ใช้ทดสอบจะมีโครงสร้างเหมือนข้อมูลฝึกทุกประการ รวมทั้งมีแอททริบิวต์ที่ระบุคลาสของข้อมูล การทดสอบจะเป็นการนำข้อมูลทดสอบแต่ละเรคคอร์ดมาตรวจสอบกับโมเดลว่าโมเดลทำนายคลาสของข้อมูลเป็นค่าใด จากนั้นนำคลาสที่โมเดลทำนายเปรียบเทียบกับคลาสที่แท้จริง ถ้าตรงกันแสดงว่าโมเดลทำนายได้ถูกต้อง แต่ถ้าไม่ตรงกันแสดงว่าโมเดลทำนายผิด ตรวจสอบเช่นนี้กับข้อมูลทดสอบทุกเรคคอร์ดและบันทึกผลการทดสอบของแต่ละเรคคอร์ด เมื่อเสร็จสิ้นการทดสอบ จะรายงานผลการตรวจสอบว่าโมเดลทำนายข้อมูลได้ถูกต้องคิดเป็นกี่เปอร์เซนต์ โดยเทียบสัดส่วนกับจำนวนข้อมูลทดสอบทั้งหมด ค่าที่ได้นี้จะเรียกว่าความแม่นตรง (accuracy) ของโมเดล

จากตัวอย่างข้อมูลสภาพอากาศที่ใช้พิจารณาประกอบการตัดสินใจเล่น/ไม่เล่นกอล์ฟ ซึ่งนำมาสร้างโมเดลได้ดังรูปที่ 2.5 (แสดงซ้ำอีกครั้งด้วยแผนภาพด้านล่างนี้) ถ้าทดสอบด้วยข้อมูลทดสอบตามตารางที่ 2.2 จะได้ค่าความแม่นตรงเป็น 90 เปอร์เซนต์ เนื่องจากทำนายข้อมูลถูกต้อง 9 เรคคอร์ดและทำนายผิด 1 เรคคอร์ด (เรคคอร์ดที่สอง จากค่าที่แท้จริงเป็น No แต่โมเดลทำนายว่า Yes) ความถูกต้องคิดเป็นสัดส่วน 9/10 = 0.9 = 90%



ตารางที่ 2.2 ข้อมูลทดสอบโมเดลการตัดสินใจเล่น/ไม่เล่นกอล์ฟ

Outlook	Temperature	Humidity	Windy	Play	Predicted by model
sunny	mild	high	false	No	No
sunny	hot	normal	true	No	Yes
cloudy	mild	high	false	Yes	Yes
rainy	mild	high	false	Yes	Yes
rainy	cool	normal	false	Yes	Yes
rainy	cool	normal	true	No	No
cloudy	cool	normal	true	Yes	Yes
sunny	hot	high	false	No	No
cloudy	hot	normal	false	Yes	Yes
rainy	mild	high	true	No	No

วิธี holdout นี้จะเหมาะสมกับกรณีที่ข้อมูลมีจำนวนมาก เช่นมีจำนวนเรคคอร์ดมากกว่า 1,000 เรคคอร์ด ทั้งนี้เนื่องจากจะต้องมีการแบ่งข้อมูลออกเป็นข้อมูลฝึกและข้อมูลทดสอบ ถ้าชุดข้อมูลมีปริมาณน้อย เช่นต่ำกว่า 100 เรคคอร์ด เมื่อแบ่งเป็นข้อมูลฝึกจะมีข้อมูลน้อยกว่า 66 เรคคอร์ดซึ่งจะส่งผลให้โมเดลที่ได้มีความถูกต้องต่ำ วิธีการสร้างและทดสอบโมเดลที่ให้ผลดีกว่าคือวิธีการทดสอบไขว้ หรือ cross-validation

วิธี k-fold cross-validation

วิธีการวัดประสิทธิภาพของ classifier แบบนี้เรียกว่าการทดสอบไขว้เนื่องจากข้อมูลทุกตัวจะถูกสลับบทบาทให้มีโอกาสทำหน้าที่เป็นทั้งข้อมูลฝึกและข้อมูลทดสอบ การฝึกและการ

ทดสอบจะกระทำหลายรอบขึ้นกับการระบุค่า k เช่นถ้าระบุ k=10 แสดงว่าจะต้องทำกระบวนการฝึกเพื่อสร้างโมเดลและทดสอบโมเดลจำนวน 10 รอบ

ในตอนเริ่มต้นจะแบ่งข้อมูลออกเป็น k ส่วน และจะวัดประสิทธิภาพ k รอบ โดย k เป็นเลขจำนวนเต็ม เช่น 5, 10, 24

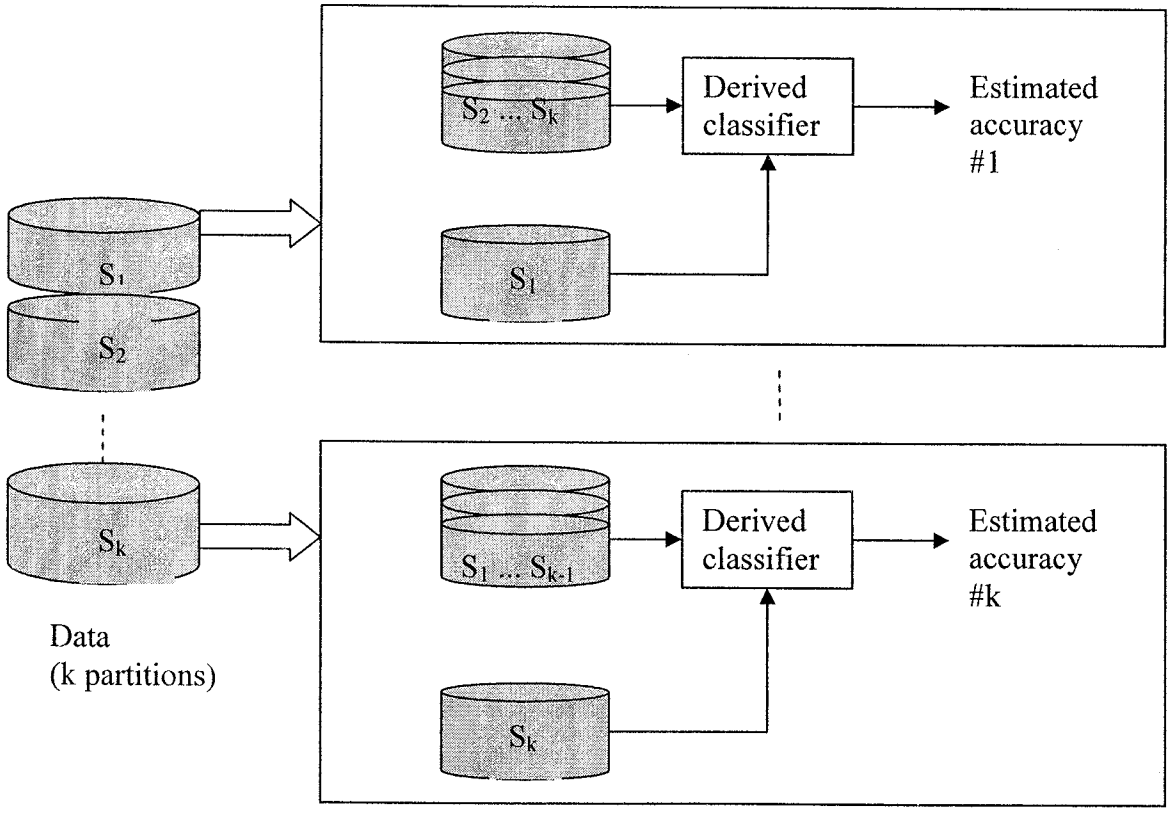
รอบที่ 1 จะใช้ข้อมูลส่วนที่ 1 เป็นข้อมูลทดสอบ ข้อมูลส่วนที่ 2 ถึงส่วนที่ k ถูกใช้เป็นข้อมูลฝึก ผลการวัดประสิทธิภาพ จะได้ค่า accuracy#1

รอบที่ 2 จะใช้ข้อมูลส่วนที่ 2 เป็นข้อมูลทดสอบ ข้อมูลส่วนที่ 1 และข้อมูลส่วนที่ 3 ถึงส่วนที่ k ถูกใช้เป็นข้อมูลฝึก ผลการวัดประสิทธิภาพ จะได้ค่า accuracy#2

...

รอบที่ k จะใช้ข้อมูลส่วนที่ k เป็นข้อมูลทดสอบ ข้อมูลส่วนที่ 1 ถึงส่วนที่ k-1 ถูกใช้เป็นข้อมูลฝึก ผลการวัดประสิทธิภาพ จะได้ค่า accuracy#k

ค่าความแม่นตรงของ classifier จะเป็นค่าเฉลี่ยของ accuracy#1, accuracy#2, ..., accuracy#k วิธีการนี้แสดงเป็นแผนภาพได้ดังรูปที่ 2.7 วิธีวัดประสิทธิภาพ classifier แบบนี้จะเหมาะกับกรณีข้อมูลมีน้อย



รูปที่ 2.7 การวัดความแม่นตรงแบบ k-fold cross-validation

ในกรณีที่การแบ่งส่วนของข้อมูลแบ่งจำนวนส่วนเท่ากับจำนวนข้อมูล เช่น ข้อมูลมี 50 เรคคอร์ด แบ่งข้อมูลเป็น 50 ส่วน (k=50) เพื่อทำการทดสอบ 50-fold cross-validation จะเรียกการทดสอบนี้ได้อีกชื่อหนึ่งว่า leave-one-out เนื่องจากในแต่ละรอบจะมีข้อมูลเพียงเรคคอร์ดเดียวเท่านั้นที่ทำหน้าที่เป็นข้อมูลทดอบ วิธีการนี้จะใช้ในกรณีที่ข้อมูลมีน้อยมาก

วิธี stratified cross-validation

วิธีการทดสอบประสิทธิภาพแบบนี้ปรับปรุงเพิ่มเติมขึ้นมาจากวิธี k-fold cross-validation โดยให้ข้อมูลที่แบ่งออกเป็น k ส่วน แต่ละส่วนมีข้อมูลครบทุกคลาสด้วยสัดส่วนเดียวกับข้อมูลตั้งต้น ตัวอย่างเช่น ถ้าข้อมูลตั้งต้นมี 1,000 เรคคอร์ด ในจำนวนนี้เป็นคลาส A 600 เรคคอร์ด และคลาส B 400 เรคคอร์ด เมื่อแบ่งข้อมูลออกเป็น 10 ส่วน (นั่นคือ k = 10) แต่ละส่วนจะมีข้อมูล 100 เรคคอร์ด และในจำนวนนี้ 60 เรคคอร์ดเป็นข้อมูลคลาส A และ 40 เรคคอร์ดเป็นข้อมูลคลาส B

2.4 งานวิจัยที่เกี่ยวข้องกับการสร้างต้นไม้ตัดสินใจเชิงอุปนัย

โครงสร้างข้อมูลประเภทต้นไม้ตัดสินใจ เริ่มได้รับความนิยมนำมาใช้อย่างแพร่หลายในงานด้านการเรียนรู้ของเครื่อง (machine learning) ดังปรากฏในเอกสารของ Breiman และคณะ (Breiman et al. 1984) และงานที่เป็นจุดเริ่มต้นสร้างความนิยมให้กับการใช้โครงสร้างต้นไม้ตัดสินใจ ได้แก่ อัลกอริทึม ID3 (iterative dichotomiser) ของ John Ross Quinlan (Quinlan 1986) และพัฒนาต่อมาเป็นโปรแกรม C4.5 (Quinlan 1993)

การสร้างต้นไม้ตัดสินใจเชิงอุปนัยเพื่อวิเคราะห์และจำแนกประเภทข้อมูล ผู้ออกแบบอัลกอริทึมจะต้องคำนึงถึงกรณีข้อมูลรบกวน เนื่องจากข้อมูลรบกวนจะทำให้โครงสร้างต้นไม้ผิดเพี้ยนไปจากที่ควรจะเป็น ซึ่งโดยทั่วไปจะส่งผลให้โครงสร้างต้นไม้มีขนาดใหญ่เกินไปเพราะพยายามจะขยายกิ่งออกไปให้สามารถอธิบายข้อมูลรบกวน การกระทำเช่นนี้จะนำไปสู่ลักษณะที่เรียกว่า *overfitting* หรือการพยายามสร้างโมเดลให้เจาะจงอธิบายได้กับข้อมูลฝึกทุกรายการมากเกินไปเพื่อให้ได้ความถูกต้องสูงที่สุด โดยที่เมื่อนำโมเดลนี้ไปอธิบายหรือทำนายชุดข้อมูลข้อมูลอื่นความถูกต้องลดลงอย่างมาก ปัญหาเรื่อง overfitting ได้มีการศึกษาและนำเสนอไว้ในเอกสารจำนวนมาก (Schaffer 1993; Talmon and McNair 1992; Wolpert 1992)

ปัญหาสำคัญของการมีข้อมูลรบกวนคือ จะทำให้โครงสร้างต้นไม้ที่สังเคราะห์ได้มีขนาดใหญ่และซับซ้อนเกินไป แนวทางการแก้ไขปัญหาโดยทั่วไปคือ (1) ใช้วิธีการควบคุมขนาดของต้นไม้ไม่ให้มีจำนวนโหนดมากเกินไป และ(2) ใช้วิธีการตัดกิ่งย่อยของต้นไม้ (pruning) ที่คาดว่ากำลังพยายามสร้างกิ่งเหล่านั้นขึ้นเพื่อให้อธิบายครอบคลุมข้อมูลรบกวน

แนวทางแก้ปัญหาแบบแรก นิยมใช้ในโปรแกรมสร้างต้นไม้ตัดสินใจเชิงอุปนัยในยุคแรกๆ (Freidman 1977) แต่ภายหลังมีการพบว่าแนวทางนี้ไม่เสถียรเท่าที่ควร จึงเปลี่ยนมาใช้วิธีการตัดกิ่งหรือ pruning เทคนิคการตัดกิ่งถูกเสนอขึ้นโดย Breiman และคณะ (1984) ภายหลัง Kim and Koehler (1994) ได้ศึกษาวิเคราะห์เงื่อนไขที่จะพิจารณาว่าการตัดกิ่งในระดับใดจึงจะให้ต้นไม้ตัดสินใจที่เหมาะสมและมีความแม่นตรงสูงที่สุด เทคนิคการตัดกิ่งที่ Quinlan ใช้ (Quinlan 1987) จะกันข้อมูลส่วนหนึ่งไว้เพื่อประโยชน์ในการตรวจสอบและตัดบางส่วนของโครงสร้างต้นไม้ที่ไม่ก่อประโยชน์ออกไป โดยการตรวจสอบจะใช้วิธีหาความสัมพันธ์ในเชิงสถิติ

Gelford และคณะ (1993) ได้เสนออัลกอริทึมในการสร้างต้นไม้และมีการตัดกิ่งบางส่วนทิ้งด้วยการแบ่งข้อมูลออกเป็นสองส่วนเช่นกัน ส่วนแรกใช้ในการสร้างต้นไม้ ส่วนที่เหลือใช้ในการตรวจสอบเพื่อตัดส่วนที่ไม่ก่อประโยชน์ทิ้ง จากนั้นสลับบทบาทของข้อมูล (cross-validation) ให้ส่วนตรวจสอบถูกนำมาใช้ในการสร้างต้นไม้และส่วนที่ใช้สร้างไปทำหน้าที่ตรวจสอบ ทำซ้ำการสร้างและการตัดกิ่งโครงสร้างต้นไม้จนกว่าต้นไม้ที่ได้ไม่มีความแม่นตรงเพิ่มขึ้น ผู้พัฒนารับรองว่าอัลกอริทึมจะลู่เข้าสู่ผลลัพธ์สุดท้าย คือได้ต้นไม้ตัดสินใจที่มีความแม่นตรงและขนาดไม่ใหญ่ซับซ้อนเกินไป

เทคนิคการตัดกิ่งอีกแนวทางหนึ่งที่ Quinlan and Rivest (1989) ใช้คือ ใช้หลักการของ minimum descriptive length (MDL) ทั้งในการสร้างต้นไม้และตัดกิ่งของต้นไม้ แต่เทคนิคนี้ยังมีข้อบกพร่องที่ถูกค้นพบในภายหลังโดย Wallace and Pattrick (1993) อีกแนวทางหนึ่งที่ใช้ได้ผลในการตัดกิ่งต้นไม้ คือ ใช้เทคนิค dynamic programming (Bohanec and Bratko 1994)

จากการที่เทคนิคการตัดกิ่งมีได้หลายวิธี ทำให้มีผู้สนใจศึกษาวิเคราะห์เปรียบเทียบประสิทธิภาพของเทคนิคการตัดกิ่ง ดังปรากฏในงานวิจัยของ Esposito et al. (1995); Cohen (1993); Mingers (1989) ในภายหลังได้มีนักวิจัยพยายามคิดค้นเทคนิคการตัดกิ่งที่มีประสิทธิภาพมากขึ้น เช่นในงานวิจัยของ Cohen and Jensen (1997) ที่ประยุกต์ใช้ทฤษฎี multiple testing (MT) โดยพยายามให้หลีกเลี่ยงลักษณะ overfitting มากที่สุด

โครงการวิจัยนี้แตกต่างจากงานวิจัยอื่นที่ได้กล่าวมา ตรงที่ไม่ได้มุ่งจัดการกับข้อมูลรบกวนในขั้นตอนของการสร้างต้นไม้ตัดสินใจ แต่พิจารณาแยกขั้นตอนการตรวจจับและจัดการกับข้อมูลรบกวนออกจากขั้นตอนของการสร้างต้นไม้ โดยอาจพิจารณาได้ว่ากรอบแนวคิดของงานวิจัยนี้แยกขั้นตอนจัดการกับข้อมูลรบกวนเป็นขั้นดำเนินการก่อน (pre-process) และขั้นตอนการสร้างต้นไม้ตัดสินใจเป็นขั้นตอนหลัก (core-process) การแยกการจัดการกับข้อมูลรบกวนออกเป็นขั้นตอนต่างหากจะส่งผลให้ลดเวลาในช่วงการสังเคราะห์โครงสร้างต้นไม้เชิงอุปนัย นอกจากนี้ยังช่วยลดเนื้อที่หน่วยความจำที่ต้องใช้ในการเก็บแต่ละกิ่งของโครงสร้างต้นไม้ ที่สร้างขึ้นเนื่องจาก

พยายามที่จะอธิบายข้อมูลรบกวน ประโยชน์โดยตรงของการลดความต้องการหน่วยความจำคือช่วยให้ซอฟต์แวร์สังเคราะห์ต้นไม้ตัดสินใจเชิงอุปนัยทำงานกับข้อมูลขนาดใหญ่ได้ และการแยกขั้นตอนจัดการกับข้อมูลรบกวนออกจากขั้นการสร้างต้นไม้ตัดสินใจ ยังจะช่วยให้การประมวลผลกับข้อมูลที่มีปริมาณไม่จำกัดเช่นข้อมูลสตรีมกระทำได้อย่างมีประสิทธิภาพมากขึ้น เนื่องจากสามารถผสานเทคนิคการสุ่มเข้ากับเทคนิคการตรวจจับข้อมูลรบกวน

บทที่ 3
วิธีดำเนินการวิจัย

โครงการวิจัยนี้มีวัตถุประสงค์ที่จะพัฒนาแนวทางและโปรแกรมต้นแบบเพื่อการจัดการกับข้อมูลรบกวนในกระบวนการสร้างต้นไม้ตัดสินใจ การจัดการกับข้อมูลรบกวนจะใช้วิธีการวิเคราะห์ข้อมูลเพื่อแยกกลุ่มข้อมูลรบกวนออกจากข้อมูลปกติด้วยเทคนิคการจัดกลุ่ม (clustering) จากนั้นข้อมูลที่มีลักษณะเกาะกลุ่มกันมากจะถูกคัดเลือก (ด้วย heuristics)ไปใช้ในขั้นตอนของการสร้างต้นไม้ตัดสินใจ รายละเอียดของอัลกอริทึมทั้งในขั้นการคัดเลือกข้อมูลและการสร้างต้นไม้ตัดสินใจจะนำเสนอในรูปแบบของภาษาเชิงตรรกะโดยใช้ภาษาโปรล็อก (Prolog) เนื่องจากเป็นภาษาระดับสูงและมีลักษณะเชิงประกาศทำให้การเขียน specification เพื่ออธิบายอัลกอริทึมทำได้สะดวก และโปรแกรมที่พัฒนาขึ้นจะสั้นกว่าการพัฒนาด้วยภาษาเชิงกระบวนคำสั่ง เช่น ภาษา C หรือภาษาเชิงวัตถุ เช่น ภาษา Java

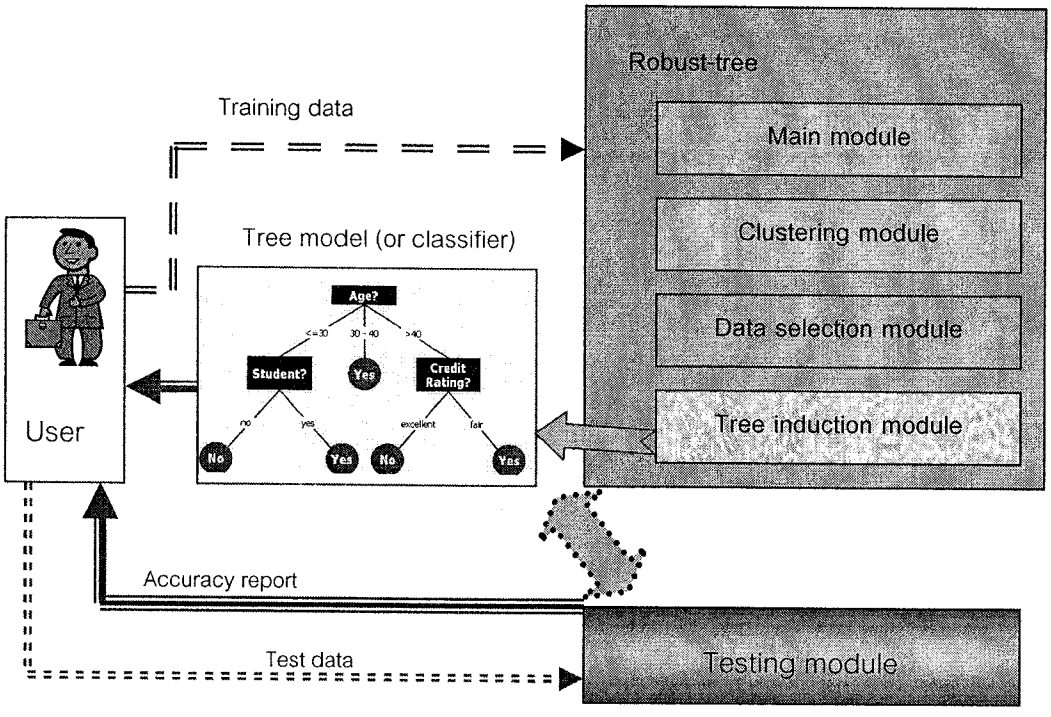
3.1 กรอบของงานวิจัย

วัตถุประสงค์หลักของงานวิจัยนี้คือ การพัฒนาซอฟต์แวร์เพื่อสร้าง classifier หรือตัวจำแนก หรืออาจเรียกได้ว่าเป็นโมเดลที่จะนำไปใช้จำแนกประเภทข้อมูล การสร้างโมเดลใช้เทคนิคของต้นไม้ตัดสินใจเชิงอุปนัย เงื่อนไขสำคัญของซอฟต์แวร์นี้คือการทำงานของโปรแกรมส่วนที่สร้างโมเดลจะต้องทนทานต่อข้อมูลรบกวน นั่นคือเมื่อข้อมูลฝึกที่รับเข้ามาเป็นอินพุทของโปรแกรมมีข้อมูลที่ผิดพลาด หรือ noise ปะปนอยู่มาก โปรแกรมจะยังต้องสามารถสร้างโมเดลที่มีค่าความแม่นตรงค่อนข้างสูง ชุดของโปรแกรมหรือซอฟต์แวร์ที่พัฒนาขึ้นนี้ให้ชื่อว่า *Robust-tree* (หมายเหตุ โดยปกติคำว่าซอฟต์แวร์ จะใช้สื่อความหมายถึงชุดของโปรแกรมหรือโปรแกรมขนาดใหญ่ที่ประกอบด้วยโปรแกรมย่อยหลายโปรแกรม ในเนื้อหาของบทนี้จะใช้ทั้งคำว่าซอฟต์แวร์และโปรแกรมตามความเหมาะสมของบริบทเมื่ออธิบาย Robust-tree)

ซอฟต์แวร์ Robust-tree ประกอบด้วยโมดูลหลัก 4 โมดูล ได้แก่

- ☐ Main module ทำหน้าที่รับข้อมูลฝึกจากผู้ใช้และส่งข้อมูลนั้นไปยังส่วนของโมดูล Clustering นอกจากนี้ยังทำหน้าที่เชื่อมโยงโมดูลย่อยอื่นๆให้เป็นชุดโปรแกรมเดียวกัน

- Clustering module ทำหน้าที่จัดกลุ่มข้อมูล หาลักษณะกลางของกลุ่ม(ค่า means ในกรณีแอททริบิวต์เป็นตัวเลข หรือค่า modes ในกรณีที่แอททริบิวต์เป็นข้อความ) จากนั้นรายงานผลการจัดกลุ่มไปยังส่วน Data selection
- Data selection module ทำหน้าที่คัดเลือกไว้เฉพาะข้อมูลที่เกาะกลุ่มอยู่ใกล้จุดกึ่งกลางหรือค่าส่วนใหญ่ของกลุ่ม ในขั้นนี้ต้องใช้ฮิวริสติก (heuristics) ประกอบการเลือก
- Tree induction module ทำหน้าที่รับข้อมูลที่คัดเลือกแล้วมาสังเคราะห์ต้นไม้ตัดสินใจด้วยการใช้วิธีคำนวณค่า gain ของแอททริบิวต์
- Testing module ทำหน้าที่วิเคราะห์ค่าความแม่นตรงของโมเดล โดยใช้ข้อมูลทดสอบที่รับมาจากผู้ใช้



รูปที่ 3.1 โครงสร้างของซอฟต์แวร์ Robust-tree

ส่วนประกอบทั้งหมดของซอฟต์แวร์ Robust-tree แสดงดังรูปที่ 3.1 จากทิศทางของลูกศรในภาพจะเห็นว่าข้อมูลฝึกหรือ training data จะถูกส่งจากผู้ใช้เข้าไปยัง main module ซึ่งทำหน้าที่เป็น user interface ผลผลิตของซอฟต์แวร์ Robust-tree คือ โมเดลในลักษณะต้นไม้ตัดสินใจ หรือ tree model ซึ่งผู้ใช้สามารถสั่งให้ Testing module ตรวจสอบความแม่นตรงของ

โมเดล โดยผู้ใช้จะต้องป้อนข้อมูลทดสอบหรือ test data ให้กับโมดูลนี้ เมื่อการตรวจสอบเสร็จสิ้น โมดูลจะรายงานผลการทดสอบเป็นค่าความแม่นตรงพร้อมทั้งเวลาทั้งหมดที่ใช้ไปในการตรวจสอบ

ซอฟต์แวร์สร้างต้นไม้ตัดสินใจเชิงอุปนัย หรือ Robust-tree ที่พัฒนาขึ้นนี้แตกต่างจากต้นไม้ตัดสินใจเชิงอุปนัยโดยทั่วไป หรือ ID3 (Quinlan, 1986) ตรงที่มี Clustering module ทำหน้าที่จัดกลุ่มข้อมูล โดยใช้ข้อสมมุติฐานว่าข้อมูลที่ไม่ถูกต้องหรือข้อมูลรบกวน รวมถึงข้อมูลที่มีลักษณะแตกต่างจากข้อมูลอื่น (เรียกว่า outlier) จะมีลักษณะโดยรวมต่างจากข้อมูลส่วนใหญ่ของกลุ่ม ซึ่งเมื่อใช้มาตรวัด similarity จะได้ระยะที่ห่างจากจุดกึ่งกลางกลุ่ม (หรือ mean) มาก

เมื่อใช้จุดกึ่งกลางกลุ่มนี้เป็นค่าอ้างอิงเพื่อส่งต่อให้กับ Data selection module ที่ทำหน้าที่คัดเลือกเฉพาะข้อมูลที่เป็นข้อมูลส่วนใหญ่ของกลุ่ม จะทำให้ข้อมูลรบกวน และ outlier บางส่วนถูกกรองออกไปจากข้อมูลฝึก จากนั้นนำข้อมูลฝึกนี้ไปใช้ในกระบวนการสร้างต้นไม้ตัดสินใจจะทำให้ได้โมเดลที่ไม่มีลักษณะ overfitting (นั่นคือ การสร้างโมเดลที่ซับซ้อนเพื่อให้ครอบคลุมข้อมูลทุกรายการ รวมถึงข้อมูลที่อาจจะเป็นข้อมูลรบกวน) การมีโมดูล Clustering และ Data selection จึงเป็นลักษณะเด่นของซอฟต์แวร์ Robust-tree ที่แตกต่างจาก ID3 ที่นิยมใช้โดยทั่วไป

3.2 การพัฒนาซอฟต์แวร์สร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทนต่อข้อมูลรบกวน

ซอฟต์แวร์ Robust-tree นี้พัฒนาด้วย SWI Prolog เวอร์ชัน 5.6.55 (ดาวน์โหลดได้จาก www.swi-prolog.org) การอธิบายรายละเอียดของซอฟต์แวร์จะเริ่มด้วยการอธิบายรูปแบบของข้อมูลที่จะนำเข้าเพื่อทำหน้าที่เป็น training data โดยตัวอย่างที่ใช้ประกอบการอธิบายจะเป็นชุดข้อมูลการพิจารณาสภาพอากาศประกอบการตัดสินใจเล่น/ไม่เล่นกอล์ฟ จากนั้นจะอธิบายรูปแบบผลลัพธ์ที่ได้จาก Robust-tree ส่วนสุดท้ายจะเป็นการอธิบายรายละเอียดโมดูลต่างๆของซอฟต์แวร์ Robust-tree

3.2.1 รูปแบบของข้อมูล

ข้อมูลฝึกที่จะเป็นอินพุทของโปรแกรม จะมีลักษณะเป็นข้อความที่อยู่ในรูปแบบของโปรแกรม Prolog เช่นเดียวกับรูปแบบของตัวโปรแกรมเอง ทั้งนี้เพื่อความสะดวกในการปรับปรุงโปรแกรมในอนาคตให้สามารถเพิ่ม background knowledge เข้าไปเป็นอินพุทอีกส่วนหนึ่ง ตัวอย่างของข้อมูลแสดงได้ดังรูปที่ 3.2

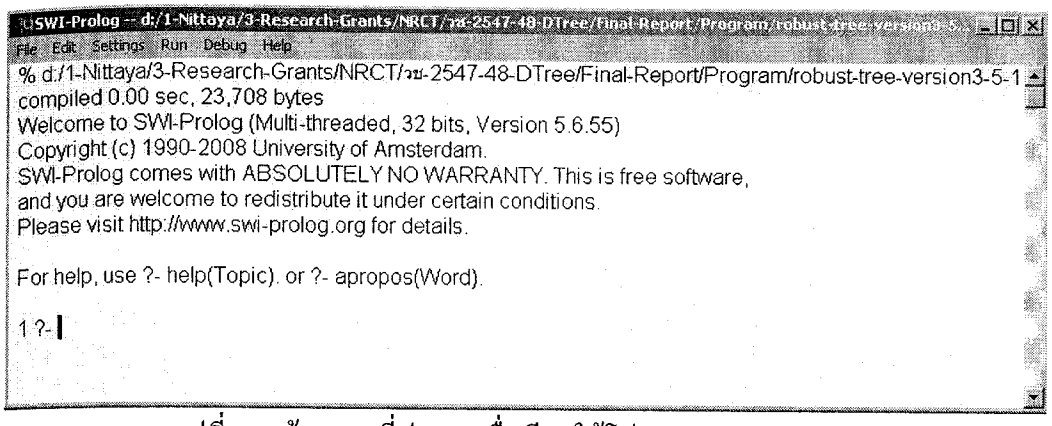
```
%% Data weather
%
% attribute detail including names and their possible values
%
attribute( outlook,  [sunny, overcast, rainy]).
attribute( temperature, [hot, mild, cool]).
attribute( humidity, [high, normal]).
attribute( windy, [true, false]).
attribute( class, [yes, no]).
%
% data detail
%
instance(1, 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 3.2 ตัวอย่างไฟล์ข้อมูลอินพุทที่จะนำเข้ายังโปรแกรม Robust-tree

บรรทัดแรกของข้อมูลเริ่มต้นด้วยเครื่องหมาย % หมายถึง comment โครงสร้างของไฟล์ข้อมูลจะแยกส่วนประกอบออกเป็นสองส่วนคือ ส่วนคำอธิบายแอททริบิวต์ (ได้แก่ส่วนข้อความ attribute ...) และส่วนแสดงรายละเอียดของข้อมูลแต่ละเรคคอร์ด (ได้แก่ส่วนข้อความ instance ...) ในส่วนที่อธิบายแอททริบิวต์ ภายในจะประกอบด้วยสองอาร์กิวเม้นต์ อาร์กิวเม้นต์แรกจะบอกชื่อแอททริบิวต์ อาร์กิวเม้นต์ที่สองเป็นลิสต์ของค่าที่เป็นไปได้ทั้งหมดของแอททริบิวต์นั้น ในส่วนของข้อมูลหรือ instance จะประกอบด้วยสามอาร์กิวเม้นต์ คือ ลำดับที่ของข้อมูลรายการนั้น, คลาสของข้อมูล, ลิสต์ที่ระบุค่าของแต่ละแอททริบิวต์ โดยวิธีการระบุค่าจะใช้รูปแบบชื่อแอททริบิวต์=ค่าของแอททริบิวต์ เมื่อสร้างข้อมูลในรูปแบบที่กำหนดเสร็จแล้วจะต้องบันทึกไฟล์ให้อยู่ในรูปแบบของโปรแกรม Prolog ที่มีนามสกุล (file extension) เป็น .pl เช่นตัวอย่างไฟล์ข้อมูลในรูปที่ 3.2 บันทึกอยู่ในชื่อ data-weather.pl

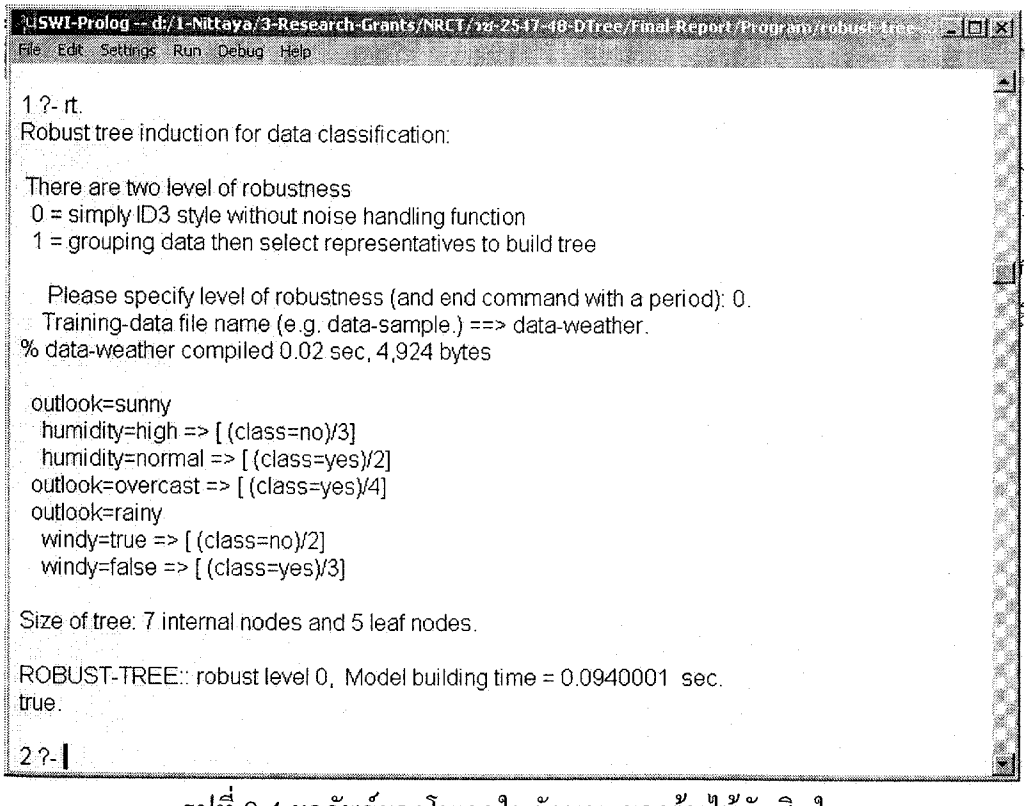
3.2.2 ผลลัพธ์ในลักษณะของโมเดล

เมื่อเตรียมไฟล์ข้อมูลแล้ว เริ่มการสร้างโมเดลด้วยการเรียกใช้โปรแกรม Robust-tree (เวอร์ชันปัจจุบันของโปรแกรมนี้คือ เวอร์ชัน 3.5.1) หน้าต่างแรกที่ปรากฏแสดงได้ดังรูปที่ 3.3



รูปที่ 3.3 ข้อความที่ปรากฏเมื่อเรียกใช้โปรแกรม Robust-tree

การทำงานของ SWI-Prolog มีลักษณะเป็น interactive เครื่องหมาย 1?- หมายถึงการพร้อมรอรับคำสั่ง ผู้ใช้สามารถสั่งงานด้วยการเริ่มใช้คำสั่งแรกของโมดูล Main คือคำสั่ง rt. (คำสั่งในภาษา Prolog จะต้องจบด้วยเครื่องหมาย ".") สิ่งที่ปรากฏจะเป็นข้อความให้เลือกว่าจะสร้างต้นไม้ตัดสินใจแบบปกติ (ใส่ค่า 0) หรือจะสร้างต้นไม้ตัดสินใจด้วยเทคนิคที่ทนทานต่อข้อมูลรบกวน (ใส่ค่า 1) ตัวอย่างในรูปที่ 3.4 เลือกการสร้างต้นไม้ตัดสินใจแบบปกติ ต่อจากนั้นจะปรากฏข้อความให้ผู้ใช้ใส่ชื่อไฟล์ข้อมูล การใส่ชื่อไฟล์นี้ไม่ต้องระบุนามสกุลในส่วน .pl



รูปที่ 3.4 ผลลัพธ์ของโมเดลในลักษณะของต้นไม้ตัดสินใจ

เมื่อโปรแกรมสร้างโมเดลเสร็จจะแสดงผลลัพธ์ในลักษณะของต้นไม้ตัดสินใจ โครงสร้างต้นไม้ที่แสดงจะอยู่ในลักษณะของ text จึงต้องอ่านในแนวนอนจากซ้ายไปขวาและการเยื้องในบรรทัดถัดไปจะหมายถึงโหนดในระดับลูก โหนดที่อยู่ทางขวามือของเครื่องหมาย => จะเป็นโหนดสุดท้ายของกิ่ง หรือ leaf node

```
outlook=sunny
        humidity=high => [(class=no)/3]
        humidity=normal => [(class=yes)/2]
outlook=overcast => [(class=yes)/4]
outlook=rainy
        windy=true => [(class=no)/2]
        windy=false => [(class=yes)/3]
```

จากโมเดลข้างต้น(ตรงกับโมเดลในรูปที่ 3.4) โหนดรากของต้นไม้คือแอททริบิวต์ outlook ซึ่งจะแตกค่าออกเป็นสามกิ่ง ในกรณี outlook มีค่าเป็น sunny โหนดในระดับต่อมาจะเป็นแอททริบิวต์ humidity ซึ่งจะแตกกิ่งต่อไปเป็นสองกิ่งตามค่าที่เป็นไปได้ทั้งหมดของ humidity กรณี humidity มีค่า high จะเชื่อมต่อไปโหนดใบที่ระบุคลาสเป็นค่า no หมายถึงนักกอล์ฟจะไม่ออกไปเล่นกอล์ฟในวันที่ outlook=sunny AND humidity=high ในส่วนที่ระบุคลาส [(class=no)/3] ตัวเลข 3 หมายถึงจากข้อมูลฝึกทั้งหมด 14 เรคคอร์ด มี 3 เรคคอร์ดที่เข้าเงื่อนไข outlook=sunny AND humidity=high ในส่วนกิ่งอื่นของต้นไม้ตันสินใจนี้อ่านได้ในทำนองเดียวกัน

หลังจากแสดงโมเดลในลักษณะโครงสร้างต้นไม้ โปรแกรมจะแสดงข้อมูลสรุปบอกขนาดของต้นไม้ว่าประกอบด้วยโหนดภายในกี่โหนดและโหนดใบกี่โหนด พร้อมทั้งรายงานระดับความทนทานของอัลกอริทึมที่ใช้สร้างโมเดลและเวลาที่ใช้ในการสร้างโมเดล ข้อความ "2?-" เป็นส่วนที่สร้างโดย SWI Prolog เพื่อเป็นเครื่องหมายพร้อมรอรับคำสั่งต่อไปจากผู้ใช้ (ตามตัวอย่างนี้จะนับเป็นคำสั่งที่สอง)

3.2.3 โมดูล Main

โมดูลหลักที่ทำหน้าที่โต้ตอบกับผู้ใช้ และทำหน้าที่เป็นส่วนเชื่อมต่อไปยังโมดูลอื่นๆ ของโปรแกรม Robust-tree คือโมดูล rt และ rtree (ในคำศัพท์ของภาษา Prolog จะเรียกโมดูลหรือฟังก์ชันเหล่านี้ว่า เพรดิเคต) อัลกอริทึมของโมดูล Main แสดงได้ดังรูปที่ 3.5

Steps:

1. Write greeting message and inform user for choices of either running ID3 or Robust-tree
2. Read(Choice)
3. Inform user to enter training-file name
4. Read(File)
5. Open(File)
6. Clear node and counter database /* this database is for generating node information and counting on node sequence number, respectively */
7. Start(Timing)
8. If Choice=0 Then call rtree(0, AttributeList) /* running ID3 */
9. If Choice=1 Then call rtree(1, AttributeList) /* running Robust-tree */
10. Stop(Timing)
11. Report execution time

รูปที่ 3.5 อัลกอริทึม Main ที่ทำหน้าที่เป็นส่วนติดต่อกับผู้ใช้

การ implement โมดูล Main ในรูปแบบของโปรแกรมโปรล็อกแสดงรายละเอียดของคำสั่งในเพรดิเคต rt ได้ดังนี้

```
%% Main module:  rt
%% ==========
  rt :-
    writeln('Robust tree induction for data classification:'), nl,
    writeln(' There are two level of robustness'),
    writeln('  0 =   = simply ID3 style without noise handling function'),
    writeln('  1 = grouping data then select representatives to build tree'),  nl,
    write('    Please specify level of robustness (and end command with a period): '),
    read(L),
    write('   Training-data file name (e.g. data-sample.) ==> '),
     read(D),              % get data file name as typed by user
     consult(D),           % then open and compile that file; data is also a prolog program
     get_time(StartTime),
                           % retractall:  clear all nodes and node-ID counter in the DB
                           %       node and counter are two global values of this program
     retractall(node(_, _, _)),
     retractall(counter(_)),
                           % findall:  make list Attr of all attribute names
                           %           except attribute class
     findall(A, (attribute(A, _), A \= class), Attr),
     rtree(L, Attr),       % then call  robust tree building module
     get_time(FinishTime),
     Time is FinishTime-StartTime,
      nl,write('ROBUST-TREE:: robust level '),write(L),write(',  '),
              write('Model building time = '), write(Time),writeln('  sec.').
```

เพรดิเคต rt จะมีหน้าที่หลักในการแสดงข้อความโต้ตอบด้วยคำสั่ง write และทำหน้าที่รับชื่อไฟล์ข้อมูลและเปิดไฟล์นั้นด้วยคำสั่ง read และ consult ตามลำดับ จากนั้นเริ่มจับเวลาเริ่มต้นสร้างโมเดลด้วยคำสั่ง get_time การเตรียมการสร้างโมเดลจะต้องลบข้อมูลเกี่ยวกับโหนด

ต่างๆของต้นไม้และตัวนับจำนวนโหนด (counter) ที่อาจจะค้างอยู่ในหน่วยความจำจากการรันโปรแกรมก่อนหน้านี้ ด้วยการใช้คำสั่ง retractall ต่อจากนั้นจะเรียกใช้โปรแกรม rtree เพื่อทำหน้าที่สร้าง robust tree โปรแกรม rtree จะมีสองรูปแบบตามการกำหนดระดับความทนทานต่อข้อมูลรบกวนที่ผู้ใช้ระบุ รูปแบบแรกคือ rtree(0, Attr) เมื่อระบุระดับความทนทานเป็นศูนย์ ซึ่งจะมีกระบวนการทำงานเหมือนโปรแกรม ID3 (Quinlan 1986) และรูปแบบที่สองคือ rtree(1, Attr) เมื่อระบุระดับความทนทานเป็นระดับ 1 ซึ่งจะมีการเพิ่มเทคนิคจัดการกับข้อมูลรบกวน ก่อนที่จะเริ่มการสร้างต้นไม้ตัดสินใจ โปรแกรม rtree ที่เพิ่มเทคนิคจัดการกับข้อมูลรบกวนแสดงในลักษณะของอัลกอริทึมได้ดังรูปที่ 3.6

Steps:

1. Read(K) /* K can be inferred from number of classes */
2. Means = Clustering(Instances, K) /* Call module clustering to get characteristics of each cluster mean; all means are stores in a list called Means */
3. Sample = DataSelection(Instances, K, Means) /* Call module DataSelection to select data samples near central point of each cluster; selected data are stored in a list called Sample */
4. MinInstance = K-log((removedData+K)/ totalInstances */ Set a heuristic MinInstances to limit tree creation; stop growing tree when number of instances in a node less than MinInstance */
5. call Induce_tree(root, Sample, AttributeList, MinInstances)
6. call Print_tree_model

รูปที่ 3.6 อัลกอริทึมสร้าง Robust-tree ที่ทนทานต่อข้อมูลรบกวน

การแปลงอัลกอริทึม Robust-tree เป็นคำสั่งโปรล็อกในรูปแบบ rtree(1,Attr) และการแปลงอัลกอริทึม ID3 เป็นคำสั่งโปรล็อกในรูปแบบ rtree(0,Attr) แสดงรายละเอียดได้ดังนี้

```
% ------------------------------------
% start traditional tree-induction with ID3 algorithm

  rtree(0,Attr) :- !,
                          % make a list Ins = [1,2,...,n] of all instance ID
     findall(N, instance(N, _, _), Ins),
                  % create decision tree, start with the root node
                  % set MinInstance in leaf nodes = 1
                  % then show model as decision tree once finish building phase
     induce_tree(root, Ins, Attr, 1),
     print_tree_model.

%-------------------------------------------
% start clustering before induce tree

  rtree(1,Attr) :- !,
```

```
attribute(class, ClassList),
length(ClassList, K),                    % K is for specifying number of clusters
findall(N, instance(N,_,_),Ins),
clustering(Ins, K, Clusters, Means),                    % grouping instances
select_DataSample(Clusters,K,Means,[],Sample),    % then select Sample
removed_Data(Sample, Ins, Removed),
length(Removed, R),
length(Ins, I),
MinInstance is K-log((R+K)/I),        % a heuristic to prune tree
induce_tree(root,Sample,Attr,MinInstance),
print_tree_model,
                        % the rest is simply for giving information to user
write('Min instances in each branch = '), writeln(MinInstance),
nl,write('Initial Data = '),write(I),writeln(' instances'),
write('Removed Data = '), writeln(Removed),
      write('    removed = '),write(R), writeln('   instances'),nl .
```

คำสั่งโปรล็อกในรูปแบบ rtree(0,Attr) จะสร้างลิสต์ชื่อ Ins บรรจุหมายเลขข้อมูลฝึกทั้งหมด เช่นถ้าข้อมูลฝึกมี 14 เรคคอร์ด ลิสต์ Ins = [1,2,3,4,5,6,7,8,9,10,11,12,13,14] จากนั้นส่งลิสต์นี้ให้โมดูล induce_tree ทำหน้าที่สร้างต้นไม้ตัดสินใจ โดยระบุให้เริ่มสร้างจาก root node และกำหนดเกณฑ์ MinInstance เป็น 1 ค่านี้ใช้เป็นข้อกำหนดเพื่อหยุดสร้างต้นไม้ย่อยเมื่อจำนวนข้อมูลในโหนดต่ำกว่าเกณฑ์นี้

คำสั่งในรูปแบบ rtree(1,Attr) จะมีขั้นตอนเพิ่มขึ้นอีกสองขั้นตอน โดยก่อนสร้างต้นไม้ตัดสินใจจะต้องจัดกลุ่มข้อมูลด้วยคำสั่ง clustering ผลลัพธ์ที่ได้คือข้อมูลที่ถูกจัดกลุ่มแล้ว (ข้อมูลจะถูกส่งกลับมายังตัวแปร Clusters) และ ลักษณะที่เป็นค่ากลางของแต่ละกลุ่ม (ตัวแปร Means) ผลลัพธ์ Clusters และ Means นี้จะถูกส่งต่อไปให้โมดูล select_DataSample เพื่อคัดเลือกข้อมูลที่เป็นตัวแทนกลุ่มเก็บไว้ในตัวแปร Sample จากนั้นจึงส่งข้อมูล Sample นี้ไปสร้างต้นไม้ตัดสินใจ พร้อมกับระบุเกณฑ์ MinInstance ด้วยฮิวริสติกต่อไปนี้

$$MinInstance = K - \log [(R+K) / I]$$

เมื่อ K คือ จำนวนกลุ่มของข้อมูล ใช้ระบุจากจำนวนคลาสทั้งหมดที่เป็นไปได้ของข้อมูลฝึก,

R คือ จำนวนข้อมูลที่ถูกคัดทิ้งเนื่องจากเป็นข้อมูลที่ไม่เกาะกลุ่ม,

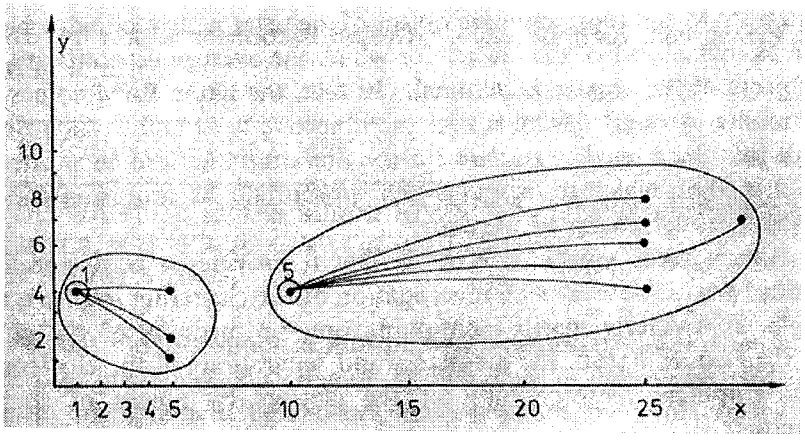
I คือ จำนวนข้อมูลฝึกทั้งหมดที่ปรากฏในไฟล์ข้อมูล

3.2.4 โมดูล Clustering

การจัดกลุ่มข้อมูลมีแนวคิดพื้นฐานจากอัลกอริทึม k-means โดยในขั้นเริ่มต้นใช้การสุ่มเลือกข้อมูลจำนวน k เรคคอร์ดเพื่อทำหน้าที่เป็นจุดกึ่งกลางของกลุ่ม จากนั้นจัดข้อมูลแต่ละตัวเข้ากลุ่มที่ใกล้ที่สุดหรือมีความคล้ายคลึงกันมากที่สุด เมื่อจัดข้อมูลเข้ากลุ่มเสร็จจะคำนวณค่ากลาง

ของกลุ่มใหม่อีกครั้ง ถ้าค่ากลางเปลี่ยนแปลงไปจากค่าเดิมแสดงว่าจะต้องเริ่มกระบวนการจัดข้อมูลแต่ละตัวเข้ากลุ่มซ้ำอีกครั้งเพราะระยะห่างหรือความคล้ายคลึงอาจจะไม่ใช่ค่าเดิม เหตุผลของแนวคิดนี้แสดงด้วยแผนภาพได้ดังรูปที่ 3.7

X	Y
1	4
5	1
5	2
5	4
10	4
25	4
25	6
25	7
25	8
29	7



รูปที่ 3.7 แสดงการจัดกลุ่มข้อมูลสองมิติด้วยอัลกอริทึม k-means

จากรูปที่ 3.7 ข้อมูลสองตัวที่ถูกสุ่มเลือกเป็นจุดกึ่งกลางหรือจุดศูนย์กลางกลุ่มชั่วคราว ได้แก่ ข้อมูลตัวที่หนึ่งที่มีค่า x และ y เป็น (1,4) และข้อมูลตัวที่ห้าที่มีค่า x และ y เป็น (10,4) ในรอบแรกเมื่อจัดข้อมูลแต่ละตัวเข้ากลุ่มที่อยู่ใกล้ที่สุด (ใช้วิธีวัดระยะห่างจากข้อมูลเทียบกับจุดศูนย์กลางของแต่ละกลุ่ม) จะเห็นได้จากภาพว่าในกลุ่มแรก(ด้านซ้ายมือ)มีข้อมูลสี่ตัว ข้อมูลแต่ละตัวแทนด้วยจุด และจุดศูนย์กลางชั่วคราวของกลุ่มแรกคือ (1,4) จะไม่ใช่จุดกึ่งกลางกลุ่มอีกต่อไป จะต้องหาระยะทางเฉลี่ยของจุดทั้งสี่จุดเพื่อเป็นจุดศูนย์กลางใหม่ที่ถูกต้องของกลุ่ม เมื่อจุดศูนย์กลางเปลี่ยน ข้อมูลที่ถูกจัดเข้ากลุ่มแต่แรกก็อาจจะมีการเปลี่ยนแปลงกลุ่ม จากภาพจะสังเกตได้ว่าข้อมูลตัวที่ห้าที่มีค่า (10,4) ควรจะถูกย้ายจากกลุ่มที่สองมาอยู่กลุ่มแรก เนื่องจากอยู่ใกล้จุดศูนย์กลางกลุ่มนี้มากที่สุด การจัดกลุ่มจึงเป็นกระบวนการทำซ้ำจนกว่าข้อมูลแต่ละตัวไม่เปลี่ยนกลุ่มอีกต่อไปจึงจะได้ผลลัพธ์การจัดกลุ่มที่ต้องการ ซึ่งเมื่อเขียนเป็นอัลกอริทึมจะได้ดังรูปที่ 3.8

Steps:

1. Initialize K means /* Create temporary mean points for all K clusters. */
   /* The following is the part for the assign_clusters sub-module*/
2. call find_clusters(K, Instances, Means) /* assign each data to the closest cluster; reference point is the mean of cluster */
3. call find_means(K, Instances, NewMeans) /* compute new mean of each cluster; this computation is based on current members of each cluster */
4. If Means ≠ NewMeans Then repeat step 2
5. Output mean values and instances in each clusters

รูปที่ 3.8 อัลกอริทึม clustering ที่ทำหน้าที่จัดกลุ่มข้อมูล

การทำ clustering จะต้องเรียกใช้โมดูลย่อยสองโมดูลคือ initialized_means ทำหน้าที่กำหนดจุดกึ่งกลางชั่วคราวของกลุ่มและ assign_clusters ทำหน้าที่จัดข้อมูลแต่ละตัวเข้ากลุ่ม โมดูล assign_clusters นี้จะทำงานซ้ำตราบใดที่ค่า entropy ที่ใช้วัดการกระจายตัวของข้อมูลภายในกลุ่มยังคงมีค่าลดลง (รายละเอียดการ implement อัลกอริทึม clustering และโมดูลย่อยต่างๆที่ถูกเรียกใช้โดยโมดูล assign_clusters แสดงขั้นตอนโดยละเอียดทั้งหมดในส่วนของภาคผนวก ข)

3.2.5 *โมดูล Data selection*

ข้อมูลที่ผ่านการจัดกลุ่มด้วยโปรแกรม clustering ผลลัพธ์ที่ได้จะถูกส่งออกมาจากโปรแกรมในชื่อของอาร์กิวเม้นต์ Clusters ซึ่งใช้โครงสร้างข้อมูลลิสต์ และข้อมูลในทุกกลุ่มจะถูกรวมมาในอาร์กิวเม้นต์นี้ เช่นในกรณีข้อมูล data-weather เมื่อจัดกลุ่มแล้วจะได้ Clusters = [13/1, 10/1, 9/1, 7/1, 5/1, 3/1, 1/1, 14/2, 12/2, 11/2, 8/2, 6/2, 4/2, 2/2, 13/2, 10/2, 9/2, 7/2, 5/2, 3/2, 1/2] รูปแบบของสมาชิกในลิสต์คือ instanceID/clusterNumber ข้อมูลในลิสต์นี้จะถูกส่งต่อไปให้โมดูล Data selection เพื่อคัดเลือกเฉพาะข้อมูลที่เป็นตัวแทน (ในโปรแกรมจะเรียกชื่อว่า DataSample) เกณฑ์ในการคัดเลือกใช้วิธีวัดค่า similarity ของข้อมูลแต่ละตัวในกลุ่มเทียบกับข้อค่ากึ่งกลางของกลุ่ม จากนั้นหาค่า variance ของค่า similarity นี้ กำหนดเกณฑ์ขั้นต่ำ (threshold) ด้วยค่า 2*variance หรือสองเท่าของค่าความแปรปรวนกลุ่ม ข้อมูลตัวใดที่มีค่า similarity ต่ำกว่าเกณฑ์ขั้นต่ำนี้จะถูกคัดทิ้ง รายละเอียดการทำงานของโมดูล Data selection นี้แสดงเป็นอัลกอริทึมได้ดังรูปที่ 3.9

Steps:

1. For each data cluster
2. Compute similarity of each member compared to cluster mean
3. Computer average similarity score of a cluster
4. Computer variance on similarity of a cluster
5. Threshold = 2* variance
6. Remove member with similarity score < Threshold
7. Return K clusters with selected data

รูปที่ 3.9 อัลกอริทึม Data selection สำหรับคัดเลือกข้อมูลตัวแทนในกลุ่ม

3.2.6 *โมดูล Tree induction*

โมดูลหลักของซอฟต์แวร์ Robust-tree คือโมดูลการอุปนัยโครงสร้างต้นไม้ตัดสินใจ ในโมดูลนี้โปรแกรมแรกที่เริ่มต้นทำงานคือโปรแกรม induce_tree ในการสร้างต้นไม้ตัดสินใจจะมีกรณีต่างๆที่อาจเกิดขึ้นได้ 5 กรณีคือ

- กรณีที่ 1 ไม่มีข้อมูลในไฟล์ข้อมูลหรือข้อมูลถูกอ่านหมดแล้ว โปรแกรมจะหยุดการสร้างต้นไม้และบันทึกค่าโหนดใบที่ระบุชื่อคลาส แต่จำนวนข้อมูลในคลาสเป็นศูนย์ เช่น [(class=yes)/0]
- กรณีที่ 2 จำนวนข้อมูลที่เหลือในขณะนั้นต่ำกว่าค่า MinInstance ซึ่งเป็นเกณฑ์ขั้นต่ำของการสร้างกิ่งในต้นไม้ย่อย โปรแกรมจะบันทึกค่าโหนดใบที่ระบุชื่อคลาสทั้งหมดที่มีในขณะนั้น เช่น [(class=yes)/1] หรือ [(class=yes)/1, (class=no)/2] ในตัวอย่างแบบที่สองนี้แปลความได้ว่าโมเดลจะทำนายคลาสของข้อมูลตามคลาสของกลุ่มใหญ่ นั่นคือ class=no
- กรณีที่ 3 ข้อมูลที่เหลือในขณะนั้นเป็นคลาสเดียวกันทั้งหมด โปรแกรมจะหยุดการสร้างต้นไม้และบันทึกค่าโหนดใบที่ระบุชื่อคลาสเช่น [(class=yes)/4]
- กรณีที่ 4 จำนวนข้อมูลที่เหลือในขณะนั้นสูงกว่าค่า MinInstance และข้อมูลยังมีหลายคลาสปะปนกัน การสร้างต้นไม้จะถูกทำซ้ำจนกว่าข้อมูลจะเป็นกรณีที่ 1, 2 หรือ 3
- กรณีที่ 5 จำนวนข้อมูลที่เหลือในขณะนั้นสูงกว่าค่า MinInstance และข้อมูลยังมีหลายคลาสปะปนกัน แต่ลักษณะของข้อมูลหรือค่าในแอททริบิวต์จะขัดแย้งกัน (inconsistent data) ทำให้ไม่มีรูปแบบร่วมหรือ pattern ในข้อมูล โปรแกรมจะหยุดสร้างต้นไม้และบันทึกค่าโหนดใบที่ระบุชื่อคลาสทั้งหมดที่มีในขณะนั้น เช่น [(class=yes)/3, (class=no)/2]

ขั้นตอนการทำงานของโมดูล Tree induction แสดงเป็นอัลกอริทึมได้ดังรูปที่ 3.10

```
Steps:
1.  If data set is empty          /* Case 1 */
2.        Then Assert(node(leaf,[Class/0], ParentNode)
3.              Exit      /* insert a leaf node in a database, then exit */
4.  If number of data instances < MinInstances     /* Case 2 */
5.        Then Compute distribution of each class
6.              Assert(node(leaf, ClassDistribution, ParentNode)
7.  If all data instances have the same class label    /* Case 3 */
8.         Then  Assert(node(leaf, ClassDistribution, ParentNode)
9.  If data  > MinInstances and data have mixing class labels  /* Case 4 */
10.        Then BuildSubtree
11. If data attributes conflict with the existing attribute values of a tree
                                                          /* Case 5 */
12.          Then stop growing and create a leaf node with mixing class labels
13. Return a decision tree
```

รูปที่ 3.10 อัลกอริทึม Tree induction ที่ทำหน้าที่สร้างต้นไม้ตัดสินใจ

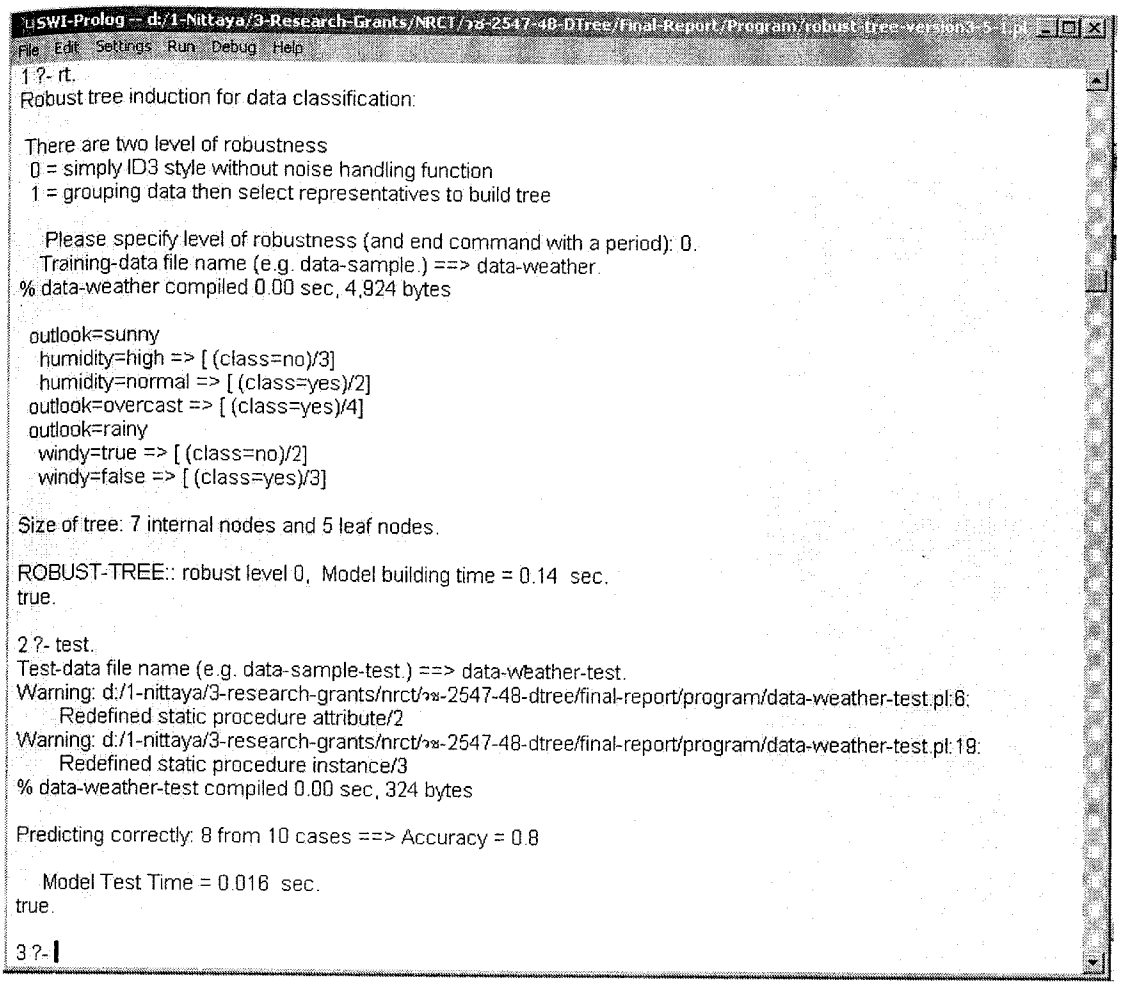
อัลกอริทึมที่แสดงดังรูปที่ 3.10 แปลงเป็นโปรแกรมโปรล็อกได้ในชื่อของโปรแกรม induce_tree (รายละเอียดปรากฏในภาคผนวก ข) ที่เป็นโปรแกรมหลักของโมดูลนี้ โปรแกรม induce_tree เรียกใช้โปรแกรมประกอบอื่นๆ อีก 5 โปรแกรมคือ

- โปรแกรม classDistribution ทำหน้าที่นับความถี่ของข้อมูลในแต่ละคลาส
- โปรแกรม choose_attribute ทำหน้าที่คัดเลือกแอททริบิวต์ที่ให้ค่า gain สูงที่สุด แอททริบิวต์ที่มีค่า gain สูงที่สุดจะถูกเลือกให้ทำหน้าที่เป็น decision attribute ของต้นไม้ตัดสินใจในแต่ละระดับ
- โปรแกรม compute_info ทำหน้าที่คำนวณค่า info ของกลุ่มข้อมูล โดยค่า Info นี้จะถูกใช้ในการคำนวณหาค่า gain ของแอททริบิวต์
- โปรแกรม build_subtree ทำหน้าที่สร้างต้นไม้ย่อย
- โปรแกรม print_tree_model ทำหน้าที่แสดงภาพโครงสร้างต้นไม้

โดยโปรแกรมเหล่านี้ได้แสดงรายละเอียดการ implement ในรูปแบบภาษาโปรล็อกไว้ในส่วนของภาคผนวก ข โดย comment ที่ปรากฏในแต่ละโปรแกรมจะอธิบายหน้าที่หลักของโปรแกรมประกอบเหล่านั้นรวมทั้งอธิบายอาร์กิวเม้นต์ต่างๆ ของโปรแกรม

3.2.7 การทดสอบโมเดลด้วยโมดูล Testing

หลังจากโปรแกรม Robust-tree สร้างผลลัพธ์ในลักษณะของ tree model แล้ว ผู้ใช้สามารถสั่งให้โปรแกรมทดสอบความแม่นตรงของโมเดล ด้วยการพิมพ์คำสั่ง test และระบุชื่อไฟล์ข้อมูลที่จะทำหน้าที่เป็น test data โปรแกรมจะอ่านข้อมูลทดสอบทีละเรคคอร์ด ทำนายคลาสของข้อมูลด้วยโมเดล จากนั้นเปรียบเทียบกับคลาสที่แท้จริงของข้อมูล เมื่อทดสอบครบทุกเรคคอร์ดแล้วจะแสดงรายงานสรุปถึงค่าความแม่นตรงและเวลาที่ใช้ทดสอบให้ผู้ใช้ทราบ รายละเอียดคำสั่งในโปรแกรมส่วน Testing module แสดงในภาคผนวก ข ส่วนการโต้ตอบกับผู้ใช้แสดงตัวอย่างจอภาพดังรูปที่ 3.11



รูปที่ 3.11 จอภาพแสดงการโต้ตอบกับผู้ใช้เพื่อทดสอบโมเดล

บทที่ 4

การทดสอบความแม่นตรงและความทนทานของต้นไม้ตัดสินใจ

ซอฟต์แวร์ Robust-tree ที่พัฒนาขึ้นได้รับการทดสอบประสิทธิภาพในสองประเด็นหลักคือ ความแม่นตรง (accuracy) ในการจำแนกข้อมูล และความทนทาน (tolerant) ต่อข้อมูลรบกวน นอกจากประสิทธิภาพในสองด้านนี้แล้วผู้วิจัยยังได้ทดสอบเวลาที่ใช้ในการสร้างโมเดลและทดสอบโมเดล ผลพลอยได้ที่สำคัญอีกประการของงานวิจัยนี้คือ การตรวจสอบความสามารถในการตรวจจับข้อมูลรบกวนของซอฟต์แวร์ที่พัฒนาขึ้น

4.1 ข้อมูลที่ใช้ในการทดสอบ

การทดสอบความแม่นตรงและความทนทานของต้นไม้ตัดสินใจใช้ข้อมูลมาตรฐานจาก UCI Repository (www.ics.uci.edu/~mlearn/MLRepository.html) จำนวน 4 ชุดข้อมูลประกอบด้วยข้อมูล Monk, Audiology, Breast cancer และ Vote ข้อมูลแต่ละชุดประกอบด้วย training data และ test data ดังนั้นวิธีการทดสอบความแม่นตรงจึงใช้วิธี hold out หรือวิธีที่แยกข้อมูลทดสอบออกจากข้อมูลฝึก เพื่อให้การทดสอบโมเดลเป็นการทดสอบกับข้อมูลใหม่โดยแท้จริง รายละเอียดของข้อมูลที่ใช้ทั้งสี่ชุดข้อมูลสรุปได้ดังตารางที่ 4.1

ตารางที่ 4.1 รายละเอียดของข้อมูลที่ใช้ทดสอบประสิทธิภาพของโมเดล

ชื่อชุดข้อมูล	จำนวนข้อมูลฝึก (instances)	จำนวนข้อมูลทดสอบ (instances)	จำนวน predicting attributes	จำนวนคลาสใน goal attribute
Monk	124	432	6	2
Audiology	150	76	69	24
Breast cancer	191	95	9	2
Vote	300	135	16	2

4.2 วิธีการทดสอบความแม่นตรงและความทนทาน

การทดสอบทั้งในด้านความแม่นตรงและความทนทาน จะใช้วิธีการรันโปรแกรม Robust-tree ทดสอบระหว่างการใช้อัลกอริทึม ID3 (Quinlan 1986) ซึ่งเป็นการสร้างโมเดลโดยไม่ได้มีการใช้เทคนิคตรวจจับและจัดการกับข้อมูลรบกวน และใช้อัลกอริทึมการจัดกลุ่มและคัดเลือกข้อมูลที่เพิ่มเติมไว้ในโปรแกรม Robust-tree เพื่อกรอง noise ออกไปจากข้อมูล

สมมุติฐานเบื้องต้นของการทดลองคือเมื่อข้อมูลมี noise อัลกอริทึม ID3 จะไม่ทราบว่าข้อมูลส่วนใดคือ noise แต่จะพยายามสร้าง tree model เพื่ออธิบายข้อมูลทั้งหมด ทำให้โมเดลมีขนาดใหญ่และมีโอกาสที่จะเกิด overfitting (หมายถึง โมเดลที่จำเพาะเจาะจงกับข้อมูลฝึกมากเกินไป) ถ้าโมเดลมีลักษณะ overfitting ผลเสียที่ตามมาก็คือ เมื่อนำโมเดลนี้ไปทำนายคลาสของข้อมูลชุดอื่น โมเดลมีโอกาสจะทำนายผิดได้สูงมาก เพื่อทดสอบสมมุติฐานเบื้องต้นนี้ ผู้วิจัยได้ทดลองสร้าง noise กับข้อมูลขนาดเล็กคือข้อมูลสภาพอากาศที่ใช้ประกอบการตัดสินใจเล่นกอล์ฟ โดยเปลี่ยนค่าคลาสของข้อมูลในเรคคอร์ดแรกและเรคคอร์ดที่สอง จากค่า no ให้เป็นค่า yes (noise ที่เกิดในแอททริบิวต์เป้าหมายเรียกว่า class noise แต่ถ้าเกิดในแอททริบิวต์อื่นๆ จะเรียกว่า attribute noise) บันทึกข้อมูลนี้ไว้ในไฟล์ชื่อ data-weather-noise.pl จากนั้นทดลองสร้าง tree model จากข้อมูลชุดนี้ด้วยอัลกอริทึม ID3 และทดลองอีกครั้งด้วยอัลกอริทึมที่พัฒนาขึ้นใน Robust-tree ผลของการเปรียบเทียบแสดงได้ดังรูปที่ 4.1

```
outlook=sunny
  humidity=high
    windy=true => [ (class=no)/1]
    windy=false
      temperature=hot => [ (class=yes)/1]
      temperature=mild => [ (class=no)/1]
      temperature=cool => [ (class=yes)/0]
  humidity=normal => [ (class=yes)/2]
outlook=overcast => [ (class=yes)/4]
outlook=rainy
  windy=true => [ (class=no)/2]
  windy=false => [ (class=yes)/3]

Size of tree: 12 internal nodes and 8 leaf nodes.

ROBUST-TREE:: robust level 0, Model building
time = 0.125 sec.
```

(a) tree model ของ ID3

```
Initial mean points = [13/1, 10/2]
Selected data = [13, 10, 9, 7, 5, 3, 1]
Selected data = [14, 12, 11, 8, 6, 4, 2, 13, 10, 9, 7, 5, 3, 1]

 outlook=sunny
   humidity=high => [ (class=no)/2, (class=yes)/1]
   humidity=normal => [ (class=yes)/2]
 outlook=overcast => [ (class=yes)/4]
 outlook=rainy
   windy=true => [ (class=no)/2]
   windy=false => [ (class=yes)/3]

Size of tree: 7 internal nodes and 5 leaf nodes.
Min instances in each branch = 3.94591

Initial Data = 14 instances
Removed Data = []
   removed = 0  instances


ROBUST-TREE:: robust level 1, Model building time =
0.0940001 sec.
```

(b) tree model ของ Robust-tree

รูปที่ 4.1 เปรียบเทียบโครงสร้างโมเดลที่สร้างโดย ID3 และ Robust-tree เมื่อข้อมูลมี noise

จากการเปรียบเทียบโครงสร้างโมเดลในรูปที่ 4.1 จะสังเกตข้อแตกต่างได้ในสองประเด็น คือขนาดของโมเดลที่สร้างโดย Robust-tree จะมีขนาดเล็กกว่า คือมีจำนวนโหนดภายใน 7 โหนดและโหนดใบ 5 โหนด รวมเป็น 12 โหนด ในขณะที่โมเดลที่สร้างจาก ID3 มีจำนวนโหนดภายใน 12 โหนด และโหนดใบ 8 โหนด รวมเป็น 20 โหนด ข้อเปรียบเทียบในด้านจำนวนโหนดนี้จะบอกถึงความสามารถในการใช้พื้นที่หน่วยความจำ การมีจำนวนโหนดมากหมายถึงจะต้องใช้หน่วยความจำมากในการบันทึกค่าของโหนด

ความแตกต่างในประเด็นที่สองคือ เวลาที่ใช้ในการสร้างโมเดล อัลกอริทึม ID3 สร้างต้นไม้โดยไม่มีข้อจำกัดด้านความลึกของต้นไม้ (นั่นคือไม่มีการใช้เทคนิค pruning) ทำให้ใช้เวลาสร้างโมเดล 0.125 วินาที ในขณะที่ Robust-tree มีฮิวริสติกช่วยในการพิจารณา stopping criteria ขณะสร้างต้นไม้ ทำให้ลดเวลาการสร้างโมเดลเหลือเพียง 0.094 วินาที การใช้ฮิวริสติกนี้ทำให้การจำแนกข้อมูลเป็นลักษณะโดยประมาณ เช่น ในส่วนต้นไม้ย่อยที่ระบุเงื่อนไข outlook=sunny AND humidity=high จะมีข้อมูลคลาส no และคลาส yes ปนกัน เมื่อทำนายคลาสจะทำนายตามค่าส่วนใหญ่ คือ คลาส no

เมื่อเปรียบเทียบ tree model ในด้านความแม่นตรง (รูปที่ 4.2) โดยการเปรียบเทียบใช้ข้อมูลทดสอบชุดเดียวกันคือ data-weather ที่มีข้อมูลที่ถูกต้องจำนวน 14 เรคคอร์ด จะเห็นว่าโมเดลที่สร้างด้วยอัลกอริทึม ID3 (นั่นคือเมื่อระบุ robust level = 0) มีความแม่นตรงในการทำนายคลาส 0.928571 ในขณะที่โมเดลที่สร้างด้วยอัลกอริทึม Robust-tree (robust level = 1) มีความแม่นตรงในการทำนายคลาส 1.0 ซึ่งหมายถึงทำนายได้ถูกต้อง 100% จากรายงานผลการทดสอบที่แสดงในรูปที่ 4.2 จะสังเกตได้อีกประการหนึ่งว่า เวลาที่ใช้ในการทดสอบ robust tree model (รูปที่ 4.2b) จะใช้เวลาน้อยกว่า ID3 tree model (รูปที่ 4.2a) ทั้งนี้เนื่องจากโครงสร้างต้นไม้ที่สั้นกว่า

```
ROBUST-TREE:: robust level 0, Model building
time = 0.125 sec.
true.

7 ?- test.
Test-data file name (e.g. data-sample-test.) ==>
data-weather.
% data-weather compiled 0.00 sec, -512 bytes

Predicting correctly: 13 from 14 cases ==>
Accuracy = 0.928571

   Model Test Time = 0.016 sec.
true.

8 ?-
```

(a) ผลการทดสอบ tree model ของ ID3

```
ROBUST-TREE:: robust level 1, Model building
time = 0.0940001 sec.
true .

5 ?- test.
Test-data file name (e.g. data-sample-test.) ==>
data-weather.
% data-weather compiled 0.00 sec, 0 bytes

Predicting correctly: 14 from 14 cases ==>
Accuracy = 1

   Model Test Time = 0.0 sec.
true.

6 ?-
```

(b) ผลการทดสอบ tree model ของ Robust-tree

รูปที่ 4.2 เปรียบเทียบความแม่นตรงของโมเดล ID3 และ Robust-tree ที่สร้างจากข้อมูลที่มี noise

การทดลองกับข้อมูล data-weather ที่มีจำนวนข้อมูลเพียง 14 เรคคอร์ด เป็นเพียงการทดสอบสมมุติฐานเบื้องต้นเกี่ยวกับประสิทธิภาพของ Robust-tree เพื่อให้ได้ผลการทดสอบที่น่าเชื่อถือมากขึ้น ผู้วิจัยได้ทดสอบผลในทำนองเดียวกันกับชุดข้อมูลมาตรฐานจำนวน 4 ชุดข้อมูล (ตามรายละเอียดในตารางที่ 4.1) ในการทดสอบจะนำข้อมูลแต่ละชุดมาสร้างเป็นข้อมูลที่มี noise ในปริมาณต่างๆกัน ตั้งแต่ noise 0%, 1%, 5%, 10%, 15%, 20%, 25% ไปจนถึงระดับ 30% ในการทดลองนี้ได้สร้าง noise จะให้มี noise เพียงหนึ่งแห่งในเรคคอร์ดที่สุ่มขึ้นมา โดย noise สามารถปรากฏที่แอททริบิวต์ใดก็ได้ (นั่นคือเป็นได้ทั้ง class noise และ attribute noise) การคำนวณปริมาณของ noise จะใช้การเทียบสัดส่วนจำนวนเรคคอร์ดกับเปอร์เซ็นต์ของ noise ที่ต้องการ เช่น ถ้าข้อมูลทั้งชุดมีจำนวน 124 เรคคอร์ด การทำให้เป็นข้อมูลที่มี noise 15% จะต้องมีข้อมูลผิดพลาด = 0.15 x 124 = 18.6 นั่นคือต้องมีข้อมูลผิดพลาดหรือ noise ปรากฏใน 19 เรคคอร์ด และการคัดเลือก 19 เรคคอร์ดจะใช้วิธีการสุ่มทั้งหมายเลขเรคคอร์ดและสุ่มเลือกหมายเลขแอททริบิวต์เพื่อกำหนดตำแหน่งสร้าง noise

ข้อมูลมาตรฐานแต่ละชุดจะถูกจำลองให้เป็นข้อมูลที่มี noise ในระดับต่างๆ แปดระดับ ตั้งแต่ 0% ไปจนถึง 30% จากนั้นทดลองสร้างโมเดลกับข้อมูลที่มี noise ต่างๆกันนี้ ด้วยวิธี ID3 และวิธี Robust-tree การทดสอบความแม่นตรงของโมเดลจะใช้ข้อมูลทดสอบชุดเดียวกัน ดังนั้นจึงเป็นวิธีการทดสอบความแม่นตรงแบบ hold out

ในการทดลองนอกจากตรวจสอบขนาดของโมเดล ความแม่นตรงของโมเดล และเวลาที่ใช้แล้ว ยังได้ทดสอบความสามารถในการตรวจจับ noise เพิ่มเติมขึ้นอีกขั้นหนึ่งด้วย จากรูปที่ 4.1 จะสังเกตได้ว่าเมื่อโปรแกรม Robust-tree สร้างโมเดลเสร็จ จะรายงานผลด้วยว่ามีข้อมูลเรคคอร์ดหมายเลขใดบ้างที่ถูกคัดทิ้ง จากข้อมูลนี้ทำให้สามารถตรวจสอบได้ว่าในจำนวนข้อมูลที่ถูกคัดทิ้ง มีเรคคอร์ดที่ถูกสุ่มเพื่อสร้าง noise ถูกคัดทิ้งไปด้วยหรือไม่ ถ้ามีก็จะนับว่าโปรแกรมสามารถตรวจพบ noise และแยก noise ออกไปได้สำเร็จ ข้อมูลของการตรวจสอบ noise และนับจำนวน noise ที่ถูกตรวจพบจึงปรากฏในผลการทดลองด้วย

4.3 ผลการทดสอบ

การทดลองเพื่อตรวจสอบความแม่นตรง ความทนทานของโมเดล และความสามารถในการตรวจจัด noise ของโปรแกรม Robust-tree เปรียบเทียบกับ ID3 โดยทดสอบกับข้อมูลมาตรฐานจำนวนสี่ชุดข้อมูล ปรากฏผลดังแสดงในตารางที่ 4.2-4.5

ตารางที่ 4.2 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Monk

Noise level	ID3			Robust-tree				
	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Instances removed	Noise removed : all noise [d]
0%	0.359+0.093=0.488	95+62=157	331/432=0.766	0.312+0.093=0.405	35+24=59	335/432=0.775	50	0:0
1%	0.344+0.141=0.485	95+62=157	343/432=0.794	0.344+0.094=0.438	35+24=59	347/432=0.803	50	0:1
5%	0.343+0.110=0.453	85+56=141	346/432=0.801	0.297+0.063=0.36	35+24=59	347/432=0.803	50	0:6
10%	0.360+0.094=0.454	81+52=133	385/432=0.891	0.375+0.094=0.469	35+24=59	347/432=0.803	50	6:12
15%	0.313+0.110=0.423	82+53=135	391/432=0.905	0.375+0.078=0.453	38+26=64	343/432=0.794	52	8:19
20%	0.766+0.250=1.016	257+162=419	216/432=0.500	0.797+0.25=1.047	179+111=290	243/432=0.563	30	12:25
25%	2.609+0.954=3.563	658+432=1090	226/432=0.523	2.593+0.968=3.561	578+380=958	237/432=0.549	11	4:31
30%	2.594+0.859=3.453	658+432=1090	225/432=0.521	2.656+0.906=3.562	563+370=933	234/432=0.542	11	4:37

[a] เวลาที่ใช้ในการสร้างโมเดล + เวลาที่ใช้ในการทดสอบโมเดล

[b] จำนวน internal nodes + จำนวน leaf nodes

[c] สัดส่วนของจำนวนข้อมูลที่โมเดลทำนายถูกต่อจำนวนข้อมูลทดสอบทั้งหมด

[d] สัดส่วนของจำนวน noise ที่ถูกคัดทิ้งต่อจำนวน noise ทั้งหมด

ตารางที่ 4.3 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Audiology

Noise level	ID3			Robust-tree				
	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Instances removed	Noise removed : all noise [d]
0%	0.414+0.069=0.483	105+67=172	63/76=0.829	0.398+0.041=0.439	89+52=141	64/76=0.842	30	0:0
1%	0.409+0.092=0.501	105+67=172	64/76=0.842	0.387+0.044=0.431	89+52=141	63/76=0.829	30	0:2
5%	0.383+0.077=0.460	99+61=160	62/76=0.816	0.379+0.038=0.417	83+47=130	65/76=0.855	31	3:8
10%	0.397+0.095=0.492	97+64=161	66/76=0.868	0.361+0.046=0.407	71+42=113	69/76=0.908	27	5:15
15%	0.471+0.133=0.604	120+65=185	63/76=0.829	0.375+0.033=0.408	71+42=113	66/76=0.868	29	7:23
20%	1.577+0.181=1.758	302+103=405	51/76=0.671	0.884+0.059=0.943	103+64=167	60/76=0.789	32	11:30
25%	1.965+0.911=2.876	413+207=620	43/76=0.566	1.726+0.656=2.382	209+68=277	55/76=0.724	20	13:38
30%	2.018+0.965=2.983	413+207=620	43/76=0.566	1.998+0.701=2.699	319+68=387	51/76=0.671	20	11:45

[a] เวลาที่ใช้ในการสร้างโมเดล + เวลาที่ใช้ในการทดสอบโมเดล

[b] จำนวน internal nodes + จำนวน leaf nodes

[c] สัดส่วนของจำนวนข้อมูลที่โมเดลทำนายถูกต่อจำนวนข้อมูลทดสอบทั้งหมด

[d] สัดส่วนของจำนวน noise ที่ถูกคัดทิ้งต่อจำนวน noise ทั้งหมด

ตารางที่ 4.4 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Breast cancer

Noise level	ID3			Robust-tree				
	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Instances removed	Noise removed : all noise [d]
0%	0.210+0.011=0.221	245+186=431	60/95=0.634	0.204+0.008=0.212	220+168=388	61/95=0.642	27	0:0
1%	0.197+0.015=0.409	245+186=431	60/95=0.634	0.213+0.015=0.228	220+168=388	61/95=0.642	27	0:2
5%	0.274+0.048=0.322	300+197=497	61/95=0.642	0.186+0.011=0.197	201+171=372	63/95=0.663	27	2:10
10%	0.211+0.069=0.280	378+201=579	60/95=0.634	0.197+0.020=0.217	201+171=372	61/95=0.642	18	3:10
15%	0.298+0.053=0.351	378+201=579	60/95=0.634	0.214+0.029=0.243	311+175=486	64/95=0.674	18	7:19
20%	0.323+0.079=0.402	411+259=670	53/95=0.558	0.231+0.037=0.268	342+184=526	55/95=0.579	25	11:29
25%	0.465+0.93=1.395	411+259=670	53/95=0.558	0.240+0.056=0.296	342+184=526	54/95=0.568	21	7:48
30%	0.512+0.104=0.616	508+296=804	49/95=0.516	0.443+0.077=0.520	398+201=599	53/95=0.558	30	14:57

[a] เวลาที่ใช้ในการสร้างโมเดล + เวลาที่ใช้ในการทดสอบโมเดล

[b] จำนวน internal nodes + จำนวน leaf nodes

[c] สัดส่วนของจำนวนข้อมูลที่โมเดลทำนายถูกต่อจำนวนข้อมูลทดสอบทั้งหมด

[d] สัดส่วนของจำนวน noise ที่ถูกคัดทิ้งต่อจำนวน noise ทั้งหมด

ตารางที่ 4.5 ผลการเปรียบเทียบประสิทธิภาพของ ID3 และ Robust-tree กับชุดข้อมูล Vote

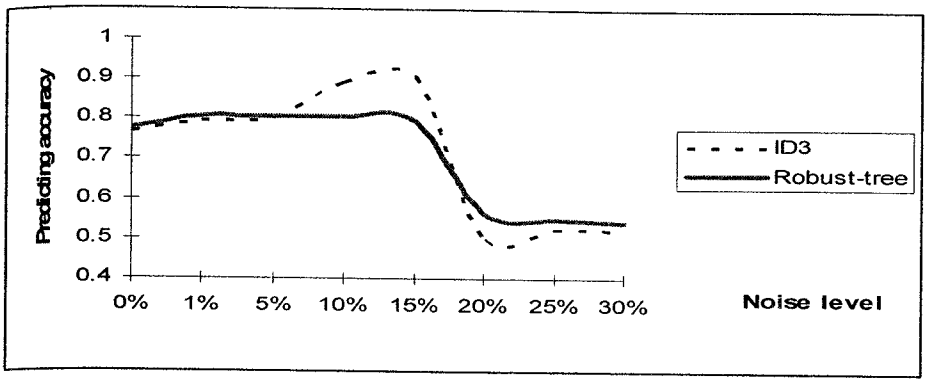
Noise level	ID3			Robust-tree				
	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Time (sec.) [a]	Size of model [b]	Predicting accuracy [c]	Instances removed	Noise removed : all noise [d]
0%	0.146+0.026=0.172	89+56=145	125/135=0.926	0.075+0.011=0.086	77+48=125	122/135=0.904	39	0:0
1%	0.159+0.017=0.176	89+56=145	125/135=0.926	0.079+0.024=0.103	77+48=125	126/135=0.933	39	1:3
5%	0.204+0.036=0.235	112+67=179	127/135=0.941	0.083+0.019=0.102	77+48=125	126/135=0.933	39	4:15
10%	0.289+0.031=0.320	134+73=207	123/135=0.911	0.092+0.028=0.120	92+61=153	125/135=0.926	39	7:30
15%	0.276+0.019=0.295	134+73=207	122/135=0.904	0.089+0.026=0.115	92+61=153	127/135=0.941	39	12:45
20%	0.377+0.024=0.401	150+81=231	113/135=0.837	0.194+0.038=0.232	116+70=186	117/135=0.867	27	13:60
25%	0.359+0.028=0.387	150+81=231	111/135=0.837	0.201+0.044=0.245	118+70=186	119/135=0.881	29	16:75
30%	0.371+0.019=0.390	190+95=285	101/135=0.748	0.217+0.050=0.267	123+76=199	114/135=0.844	35	16:90

[a] เวลาที่ใช้ในการสร้างโมเดล + เวลาที่ใช้ในการทดสอบโมเดล

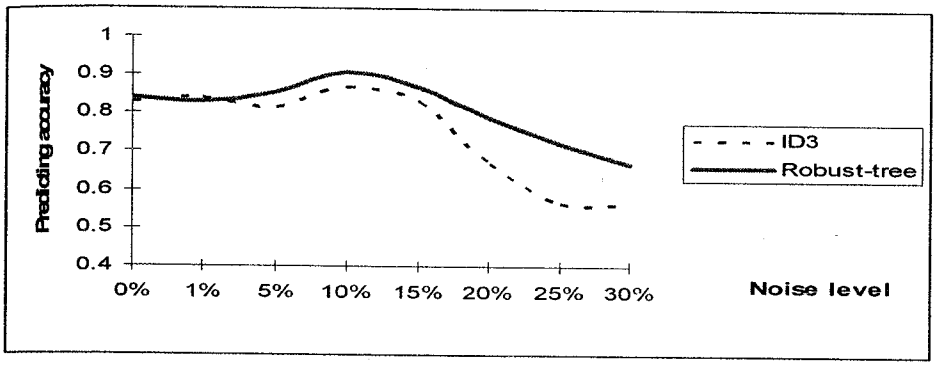
[b] จำนวน internal nodes + จำนวน leaf nodes

[c] สัดส่วนของจำนวนข้อมูลที่โมเดลทำนายถูกต่อจำนวนข้อมูลทดสอบทั้งหมด

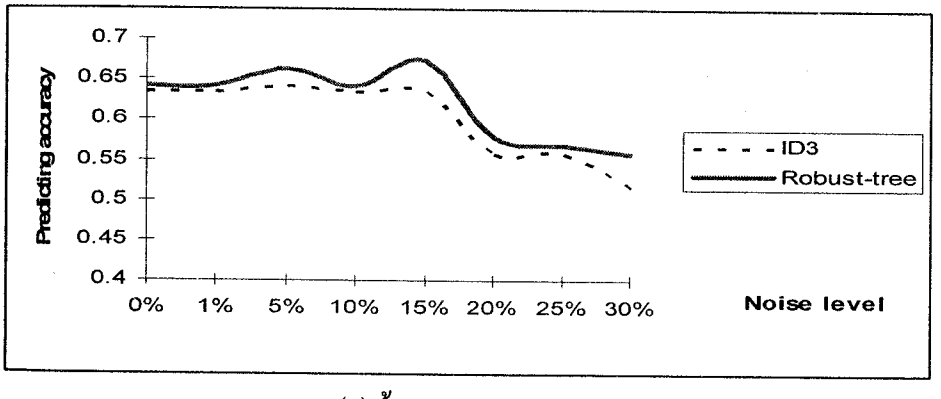
[d] สัดส่วนของจำนวน noise ที่ถูกคัดทิ้งต่อจำนวน noise ทั้งหมด



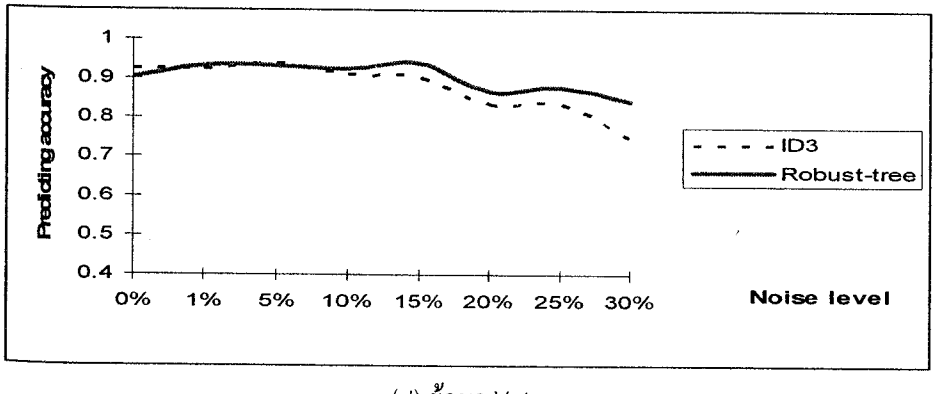
(a) ข้อมูล Monk



(b) ข้อมูล Audiology

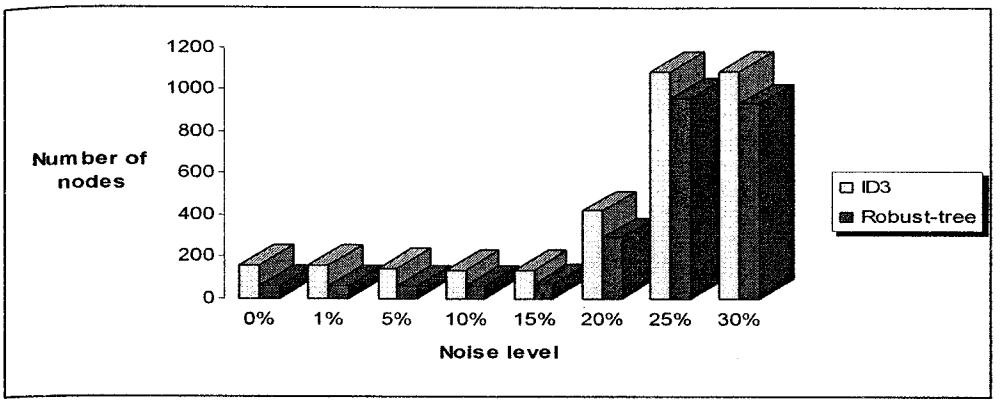


(c) ข้อมูล Breast cancer

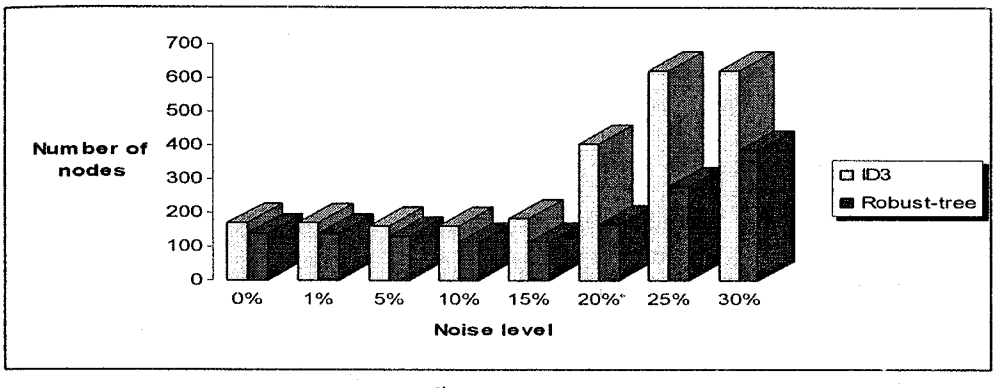


(d) ข้อมูล Vote

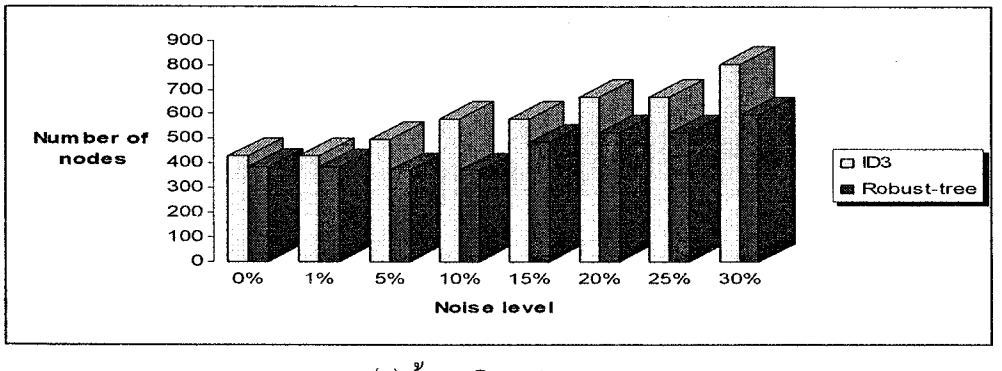
รูปที่ 4.3 กราฟเปรียบเทียบค่าความแม่นตรงของ ID3 model และ Robust-tree model



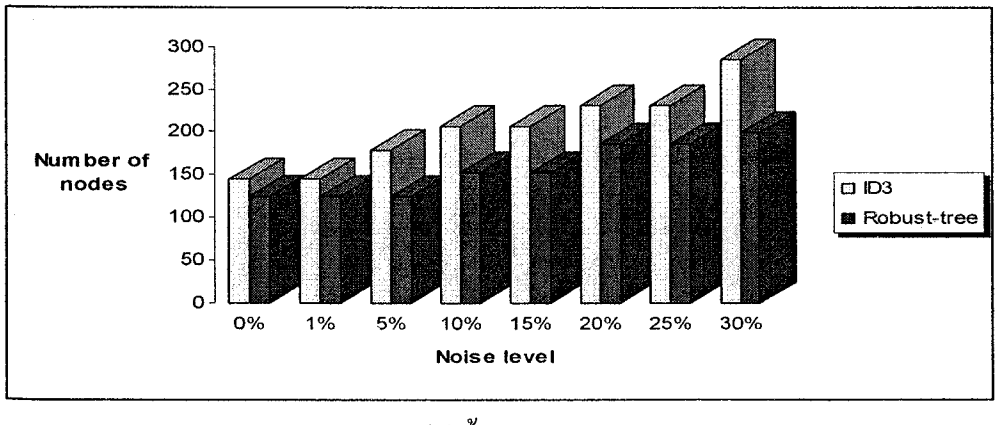
(a) ข้อมูล Monk



(b) ข้อมูล Audiology



(c) ข้อมูล Breast cancer



(d) ข้อมูล Vote

รูปที่ 4.4 กราฟเปรียบเทียบขนาดของ ID3 model และ Robust-tree model

รูปที่ 4.3 แสดงภาพกราฟค่าความแม่นตรงของโมเดลที่สร้างจากอัลกอริทึม ID3 เพื่อเปรียบเทียบกับค่าความแม่นตรงของโมเดลที่สร้างจากอัลกอริทึม Robust-tree เมื่อข้อมูลมี noise เพิ่มขึ้นจาก 0% ไปจนถึง 30% จากข้อมูลมาตรฐานทั้งสี่ชุดจะเห็นได้ว่าค่าความแม่นตรงของโมเดลจะลดลงเมื่อ noise เพิ่มปริมาณมากขึ้น การลดลงของความแม่นตรงนี้จะเห็นได้ชัดเจนเมื่อ noise มีปริมาณมากกว่า 15% เมื่อเปรียบเทียบระหว่างความทนทานต่อ noise ของ ID3 model และ Robust-tree model จะเห็นว่า Robust-tree model มีความทนทานมากกว่า โดยโมเดลจะทนทานต่อ noise ในระดับ 20-30% ได้โดยค่าความแม่นตรงค่อนข้างคงที่ แต่โมเดลของ ID3 จะลดความแม่นตรงลงตามลำดับ

การเปรียบเทียบขนาดของโมเดล (แสดงในรูปที่ 4.4) โดยนับจากจำนวนโหนดของโครงสร้างต้นไม้ในโมเดลที่เป็นผลลัพธ์ จะสัมพันธ์โดยตรงกับประสิทธิภาพการใช้เนื้อที่หน่วยความจำ โมเดลที่มีจำนวนโหนดมากจะใช้หน่วยความจำมาก และขนาดของโมเดลจะมีผลต่อเวลาที่ใช้ในการทดสอบความแม่นตรงของโมเดล โมเดลที่มีขนาดเล็กกว่าจะใช้เวลาในการทดสอบสั้นกว่า จากกราฟแท่งที่เปรียบเทียบ ID3 model และ Robust model จะเห็นว่าขนาดของ Robust tree model เล็กกว่าในทุกชุดข้อมูล ทำให้สรุปได้ว่า Robust tree จะมีประสิทธิภาพสูงกว่า ID3 ในด้านการใช้พื้นที่หน่วยความจำ

กล่าวโดยสรุปแล้วงานวิจัยนี้พัฒนาต่อยอดแนวคิดของอัลกอริทึม ID3 (Quinlan, 1986) ซึ่งเป็นอัลกอริทึมที่นิยมใช้ในงานทำเหมืองข้อมูลและงาน machine learning อัลกอริทึมนี้ใช้ข้อมูลเพื่อสร้างโมเดลของข้อมูลเหล่านั้นในลักษณะของต้นไม้ตัดสินใจ แต่ข้อด้อยของ ID3 คือสร้างโมเดลที่มีความแม่นตรงต่ำและมีลักษณะ overfitting เมื่อข้อมูลที่ใช้ฝึกมีข้อมูลรบกวน

งานวิจัยนี้จึงมีแนวคิดที่จะปรับปรุงกระบวนการสร้างโมเดลให้มีความทนทานมากขึ้นต่อข้อมูลรบกวน แนวคิดพื้นฐานที่ใช้ในการปรับปรุงคือเพิ่มขั้นตอนการทำ clustering หรือการจัดกลุ่มข้อมูล แนวคิดนี้ใช้สมมุติฐานที่ว่าข้อมูลรบกวนเป็นข้อมูลที่มีค่าผิดพลาดในบางแอททริบิวต์ ทำให้เมื่อนำข้อมูลทั้งรายการหรือทั้งเรคคอร์ดที่ประกอบขึ้นจากหลายแอททริบิวต์ มาเปรียบเทียบกับข้อมูลส่วนใหญ่ของกลุ่ม (ใช้ค่า mean หรือค่าที่เป็นตัวแทนกลุ่ม เป็นค่าอ้างอิงในการเปรียบเทียบ) จะทำให้ข้อมูลรบกวนอยู่ในระยะที่ห่างจากค่า mean (ในกรณีที่ข้อมูลเป็นตัวเลข) หรือมีความแตกต่างจากค่า mean (ในกรณีที่ข้อมูลเป็นข้อความ) เมื่อวัดจากระยะห่างหรือวัดจากความต่าง จะช่วยให้เราสามารถกำจัดข้อมูลรบกวนออกจากชุดข้อมูลฝึกได้ ประโยชน์ที่ได้คือโมเดลที่สร้างขึ้นจะไม่มีลักษณะ overfitting ซึ่งสามารถทดสอบได้ด้วยการนำโมเดลไปทดสอบกับชุดข้อมูลทดสอบ จากสมมุติฐานดังกล่าวจึงเป็นจุดเริ่มต้นของการพัฒนาซอฟต์แวร์ Robust-tree

ซอฟต์แวร์ Robust-tree ที่พัฒนาขึ้นมีข้อแตกต่างจาก ID3 ตรงที่นำข้อมูลฝึกมาผ่านกระบวนการจัดกลุ่มเพื่อหาค่ากึ่งกลาง หรือค่าตัวแทนของกลุ่ม จากนั้นนำค่าตัวแทนนี้เป็นเกณฑ์

พิจารณาคัดเลือกข้อมูลที่จะใช้ในขั้นตอนการสร้างโมเดลในลักษณะต้นไม้ตัดสินใจ การคัดเลือกใช้ฮิวริสติก Threshold = 2* Variance-of-cluster-similarity นั่นคือจะต้องคำนวณค่า variance ของความคล้ายหรือความใกล้เคียงกับค่ากึ่งกลางของข้อมูลทั้งกลุ่ม จากนั้นคัดเลือกข้อมูลในกลุ่มเก็บไว้เฉพาะข้อมูลที่มีระดับความคล้ายกับค่ากึ่งกลาง สูงกว่าค่า Threshold

ข้อมูลที่ผ่านการคัดเลือกจะถูกส่งไปเข้ากระบวนการสร้างต้นไม้ตัดสินใจ โดยในขั้นตอนนี้จะใช้ฮิวริสติกของการกำหนดค่า MinInstance ซึ่งเป็นค่าของจำนวนข้อมูลในแต่ละโหนดขณะมีการสร้างโหนดเพิ่มในต้นไม้ตัดสินใจ โดยฮิวริสติกกำหนดได้ดังนี้

$$MinInstance = K - \log [(R+K) / I]$$

เมื่อ K คือ จำนวนกลุ่มของข้อมูล ใช้ระบุจากจำนวนคลาสทั้งหมดที่เป็นไปได้ของข้อมูลฝึก,

R คือ จำนวนข้อมูลที่ถูกคัดทิ้งเนื่องจากเป็นข้อมูลที่ไม่เกาะกลุ่ม,

I คือ จำนวนข้อมูลฝึกทั้งหมดที่ปรากฏในไฟล์ข้อมูล

จากการเพิ่มส่วน clustering, ทำ data selection ด้วยฮิวริสติก Threshold และใช้ค่า MinInstance เป็นฮิวริสติกในการสร้างโหนดของต้นไม้ตัดสินใจ ทำให้ซอฟต์แวร์ Robust-tree ที่พัฒนาขึ้นมีความทนทานมากขึ้นต่อข้อมูลรบกวน ดังจะเห็นได้จากผลการทดสอบในตารางที่ 4.2-4.5 และแสดงเป็นภาพสรุปเปรียบเทียบกับ ID3 ได้ดังรูปที่ 4.3 และ 4.4

บทที่ 5
บทสรุป

การทำเหมืองข้อมูลเป็นกระบวนการวิเคราะห์ข้อมูลอัตโนมัติ โดยเน้นการทำงานกับข้อมูลขนาดใหญ่ เช่น ข้อมูลภาพถ่ายดาวเทียม ข้อมูลสำมะโนประชากร การวิเคราะห์มิได้เน้นผลความเที่ยงตรงและถูกต้องแต่ประการเดียวเหมือนการวิเคราะห์ในเชิงสถิติ แต่เน้นการค้นหาโมเดลที่สามารถอธิบายข้อมูล และการค้นหาอย่างฉลาดเพื่อให้ได้แพทเทิร์นหรือรูปแบบที่ไม่ได้คาดหมายล่วงหน้าที่แฝงอยู่ในข้อมูลขนาดใหญ่เหล่านั้น ซึ่งการค้นหารูปแบบความสัมพันธ์ในลักษณะนี้จะต้องใช้เวลาในการค้นหานานมาก จึงต้องใช้โปรแกรมคอมพิวเตอร์ทำหน้าที่ค้นหาโมเดลที่น่าสนใจจากข้อมูลขนาดใหญ่เหล่านั้น แต่เนื่องจากการทำงานของโปรแกรมเป็นระบบอัตโนมัติที่ไม่สามารถทราบลักษณะของข้อมูลที่กำลังวิเคราะห์อยู่ได้ว่าเป็นข้อมูลที่ถูกต้อง หรือเป็นข้อมูลรบกวนที่มีบางค่าในข้อมูลผิดเพี้ยนไป ทำให้ในการวิเคราะห์ข้อมูลจริงที่มักจะมีข้อมูลรบกวนปะปนอยู่ไม่ได้โมเดลที่ดีเท่าที่ควร งานวิจัยนี้จึงมีวัตถุประสงค์ที่จะพัฒนาเทคนิคที่ช่วยให้การค้นหาโมเดลจากข้อมูลที่มีข้อมูลที่ดีปะปนอยู่กับข้อมูลรบกวนได้อย่างมีประสิทธิภาพ

5.1 สรุปผลการวิจัย

โครงการวิจัยนี้เป้าหมายหลักสองประการคือ (1) การพัฒนาแนวทางหรืออัลกอริทึมที่จะช่วยให้สามารถกำจัดข้อมูลรบกวนออกไปจากข้อมูลที่ดีได้ และใช้ข้อมูลที่คัดเลือกแล้วสร้างโมเดลในลักษณะของต้นไม้ตัดสินใจ และ (2) พัฒนาอัลกอริทึมให้เป็นซอฟต์แวร์ที่สามารถใช้งานได้จริงและมีประสิทธิภาพสูง

ในส่วนของการพัฒนาอัลกอริทึม สามารถสรุปเป็นขั้นตอนการทำงานได้ดังนี้

Input: Data D with class label
Output: A tree model M
Steps:

1. Read D and extract class label to check distinctive values K
2. Cluster D to group data into K groups
3. In each group
   - 3.1 Get mean attribute values
   - 3.2. Compute similarity of each member compared to its mean
   - 3.3 Compute average similarity and variance
   - 3.4 Set threshold T = 2*Variance
   - 3.5 Select only data with similarity > T
4. Set stopping criteria S for tree building as

$$S = K - \log [(\text{number of removed data} + K) / |D|]$$

5. Send selected data and criteria S into tree-induction module
6. Return tree model

ขั้นตอนที่ 5 และ 6 ของอัลกอริทึมเป็นขั้นตอนปกติของโปรแกรม ID3 (Quinlan, 1986) และโปรแกรม decision-tree induction โดยทั่วไป ขั้นตอนที่ได้รับการพัฒนาเพิ่มเติมในงานวิจัยนี้คือขั้นตอนที่ 2, 3 และ 4 โดยในขั้นตอนที่ 2 เป็นการจัดกลุ่มข้อมูลออกเป็น K กลุ่ม เมื่อ K คือเลขจำนวนเต็มที่โปรแกรมตีความได้โดยอัตโนมัติ (ในขั้นตอนที่ 1 ของอัลกอริทึม) จากจำนวนคลาสของชุดข้อมูลฝึก เช่น ถ้าข้อมูลฝึกมีคลาสที่แตกต่างกันสองคลาส ค่า K จะมีค่าเป็น 2 ซึ่งหมายถึงในขั้นตอนที่ 2 ของอัลกอริทึมจะมีการจัดกลุ่มข้อมูล (clustering) ออกเป็นสองกลุ่ม

การจัดกลุ่มข้อมูลมีวัตถุประสงค์ที่จะหาลักษณะตัวแทน (representative) ของข้อมูลในแต่ละคลาส ลักษณะตัวแทนนี้เป็นค่าส่วนใหญ่ที่ข้อมูลในคลาสนั้นมีคล้ายกัน ดังนั้นเมื่อมีข้อมูลรายการใดในคลาสที่มีค่าแตกต่างอย่างมากจากลักษณะตัวแทน ทำให้สันนิษฐานได้ว่าข้อมูลรายการนั้นเป็นข้อมูลที่มีข้อผิดพลาด หรือเรียกว่าข้อมูลรบกวน (noisy data) อาจมีบางกรณีที่ไม่ใช่ข้อมูลผิดพลาด แต่เป็นข้อมูลที่มีค่าแตกต่างจากกลุ่มอย่างชัดเจน เรียกว่า ข้อมูลขอบ หรือ outlier ข้อมูลทั้งที่เป็น noise และ outlier จะไม่มีประโยชน์ต่อการสร้างโมเดล เนื่องจากโมเดลเป็นลักษณะโดยรวมของข้อมูลทั้งกลุ่มที่แสดงลักษณะเหล่านั้นออกมาคล้ายกัน ข้อมูลรบกวนและข้อมูลขอบเป็นข้อมูลที่แตกต่างจากกลุ่มจึงไม่แสดงโมเดลที่เป็นลักษณะร่วม ยิ่งกว่านั้นการมีข้อมูลรบกวนและข้อมูลขอบในปริมาณที่มาก อาจทำให้โมเดลที่สังเคราะห์ขึ้นมีความละเอียดมากเกินไป (เรียกว่า overfitting) เนื่องจากพยายามรวมลักษณะของข้อมูลรบกวนและข้อมูลขอบเข้ามาในโมเดล ซึ่งจะส่งผลให้โมเดลมีความแม่นตรงต่ำ เมื่อนำไปอธิบายหรือทำนายคลาสของข้อมูลชุดอื่น

จากการวิเคราะห์ลักษณะของข้อมูลรบกวนและข้อมูลขอบข้างต้น ทำให้ผู้วิจัยออกแบบขั้นตอนของการทำ clustering ก่อนที่จะมีการสร้างโมเดล ผลลัพธ์จากการทำ clustering จะได้กลุ่มของข้อมูลจำนวน K กลุ่มพร้อมด้วยคำอธิบายลักษณะของข้อมูลที่ถูกเลือกเป็นตัวแทนกลุ่มของทั้ง K กลุ่ม คำอธิบายลักษณะตัวแทนกลุ่มนี้เป็นข้อมูลสำคัญที่จะถูกนำไปใช้ในขั้นตอนที่ 3 ของอัลกอริทึม ซึ่งเป็นขั้นตอนของการคัดเลือกเฉพาะข้อมูลที่เหมาะสมบางตัวจากแต่ละ K กลุ่ม เกณฑ์ของความเหมะสมใช้ค่าฮิวริสติก T = 2 * Variance โดยค่า T จะเป็นค่าเฉพาะกลุ่มและเป็นค่าขั้นต่ำที่ใช้คัดเลือกข้อมูลในกลุ่ม (ข้อมูลที่มีแนวโน้มว่าจะเป็นข้อมูลรบกวนและข้อมูลขอบ จะไม่ถูกคัดเลือก)

ในขั้นตอนของการคัดเลือกจะใช้การวัดความคล้าย (similarity) หรือค่า Sim ของข้อมูลแต่ละตัวในกลุ่มเทียบกับค่าที่เป็นตัวแทนกลุ่ม นั่นคือข้อมูลแต่ละตัวจะมีค่าความคล้ายนี้กำกับอยู่ จากนั้นคำนวณค่าความคล้ายโดยเฉลี่ยของทั้งกลุ่ม (หรือ mean(Sim)) เพื่อใช้ค่าเฉลี่ยนี้ไปคำนวณค่าความแปรปรวน (variance) ของทั้งกลุ่ม (หรือ variance(Sim)) ปริมาณสองเท่าของความแปรปรวนนี้จะถูกใช้เป็นฮิวริสติก T (หรือ threshold) เพื่อการคัดเลือกข้อมูล โดยข้อมูลแต่ละ

ตัวในกลุ่มจะถูกประเมินค่า Sim และถ้าค่า Sim มีค่ามากกว่าเกณฑ์ขั้นต่ำ T ข้อมูลตัวนั้นจะถูกคัดเลือกไว้ ข้อมูลที่ไม่ถูกเลือกจะแยกเป็นอีกกลุ่มเรียกว่ากลุ่มที่ถูกคัดทิ้ง

จำนวนข้อมูลทั้งหมดในกลุ่มที่ถูกคัดทิ้ง จะนำไปใช้ในการคำนวณค่าฮิวริสติก S ที่ทำหน้าที่เป็น stopping criteria หรือเกณฑ์ในการหยุดสร้างต้นไม้ตัดสินใจ การคำนวณค่า S เกิดขึ้นในขั้นตอนที่ 4 ของอัลกอริทึม โดยสร้างเป็นสูตรคำนวณดังนี้

$$S = K - \log [(\text{number of removed data} + K) / |D|]$$

นั่นคือ stopping criteria S จะเป็นผลต่างของจำนวนกลุ่ม กับค่าลอการิทึมของจำนวนข้อมูลที่ถูกคัดทิ้งบวกด้วยจำนวนกลุ่ม และหารด้วยจำนวนข้อมูลฝึกทั้งหมด การที่ต้องใช้ค่า K ที่เป็นจำนวนกลุ่มบวกกับจำนวนข้อมูลที่ถูกคัดทิ้งก็เพื่อป้องกันกรณีที่ไม่มีข้อมูลถูกคัดทิ้ง (เนื่องจากไม่มี noisy data และ outlier ในข้อมูล) ซึ่งจะทำให้เป็นการหาค่าลอการิทึมของศูนย์ (และจะเกิด semantic error)

ค่าฮิวริสติก S นี้จะถูกส่งต่อไปยังขั้นตอนที่ 5 ที่เป็นขั้นสร้างต้นไม้ตัดสินใจ โดยค่า S จะทำหน้าที่เป็นค่าขั้นต่ำประกอบการพิจารณาว่าเมื่อโหนดของต้นไม้ ณ ขณะใดมีจำนวนข้อมูล (นับรวมข้อมูลทุกคลาส) ในโหนดนั้นต่ำกว่า S ให้หยุดการสร้างโหนดลูกในระดับต่อจากโหนดนั้น และให้โหนด ณ ขณะนั้นทำหน้าที่เป็นโหนดใบ (leaf node) พร้อมทั้งบันทึกคลาสของข้อมูลทุกคลาสลงในโหนดนั้น เช่น node(leaf, [(class=yes/2), (class=no/3)], 4) หมายถึงโหนดใบนี้มีจำนวนข้อมูล 5 รายการ (สมมุติให้ S=6) ประกอบด้วยข้อมูลในคลาส yes สองรายการ และข้อมูลในคลาส no สามรายการ (นั่นคือถ้าจะใช้โหนดนี้ไปทำนายคลาสของข้อมูล ควรจะทำนายว่าเป็นคลาส no เนื่องจากเป็นคลาสส่วนใหญ่) จำนวนเลข 4 ที่ปรากฏเป็นอาร์กิวเมนต์สุดท้ายหมายถึงโหนดแม่ของโหนดนี้ นั่นคือโหนดใบนี้แตกกิ่งมาจากโหนดหมายเลข 4

การพัฒนาอัลกอริทึมให้เป็นโปรแกรมที่สามารถใช้งานได้จริง ใช้แนวทางของการโปรแกรมเชิงประกาศ (declarative programming) ด้วยภาษา Prolog ลักษณะการโปรแกรมเชิงประกาศจะแตกต่างจากการโปรแกรมเชิงกระบวนคำสั่ง (procedural or imperative programming) ตรงที่รูปแบบคำสั่งจะสั้นกว่าและใช้วิธีการประกาศแพทเทิร์นของอินพุทและเอาท์พุท โดยไม่ต้องระบุรายละเอียดขั้นตอน ข้อแตกต่างนี้จะเห็นได้ชัดเจนจากตัวอย่างต่อไปนี้ที่แสดงการเปรียบเทียบวิธีการเขียนโปรแกรมเรียงลำดับข้อมูลแบบ quick sort ด้วยวิธีการโปรแกรมเชิงประกาศ (ด้วยภาษาโปรล็อก) เทียบกับวิธีการโปรแกรมเชิงกระบวนคำสั่ง (ด้วยภาษาซี)

Prolog

```
qs([],[]).
qs([ X | Xs]) :- part(X, Xs, Littles, Bigs),
                 qs( Littles, Ls),
                 qs( Bigs, Bs),
                 append(Ls, [X| Bs], Ys).
part(_, [], [], []).
part(X, [Y|Xs], [Y|Ls], Bs) :- X>Y, part(X, Xs, Ls, Bs).
part(X, [Y|Xs], Ls, [Y|Bs]) :- X=<Y, part(X, Xs, Ls, Bs).
```

C

```
int partition(int y[], int f, int l);
void quicksort(int x[], int first, int last) {
        int pivIndex = 0;
        if(first < last) {
                        pivIndex = partition(x,first, last);
                        quicksort(x,first,(pivIndex-1));
                        quicksort(x,(pivIndex+1),last);
        }
 }
int partition(int y[], int f, int l) {
        int up, down, temp, cc, piv = y[f];
        up = f;
        down = l;
        do {
                while (y[up] <= piv && up < l)  { up++; }
                while (y[down] > piv  )  { down--; }
                if (up < down ) {   temp = y[up];
                                    y[up] = y[down];
                                    y[down] = temp;    }
        } while (down > up);
        temp = piv;
        y[f] = y[down];
        y[down] = piv;
        return down;
      }
```

การโปรแกรมเชิงประกาศจะใช้เทคนิคการเทียบรูปแบบ หรือ pattern matching เพื่อระบุรูปแบบอินพุทและเอาท์พุท และใช้วิธีการเรียกตัวเองซ้ำในลักษณะของ recursive ทำให้โปรแกรมเขียนได้สะดวก ดังตัวอย่างการค้นหาเลขไฟโบนาชชีด้วยภาษาโปรล็อกต่อไปนี้

```
% Fibonacci function in Prolog

fib(0, 0).                  % pattern 1: input is 0, output is 0.
fib(1, 1).                  % pattern 2: input is 1, output is 1.
fib(N, F) :-  N > 1,        % pattern 3: input is N and N>1
        N1 is N-1,                  % then declare N1 = N-1
        N2 is N-2,                  %              N2 = N-2
        fib(N1, F1), fib(N2, F2),   % recursive calls fib() to get F1 and F2
        F is F1 + F2.               %      compute result F = F1+F2
```

จากตัวอย่างข้างต้นที่แสดงประสิทธิภาพของภาษาโปรล็อก ที่สอดคล้องกับแนวทางการโปรแกรมเชิงประกาศ ทำให้ผู้วิจัยเลือกพัฒนาโปรแกรมด้วยแนวทางนี้ และพบว่าซอฟต์แวร์ Robust-tree ที่พัฒนาขึ้นมีขนาดค่อนข้างสั้น ทำให้ง่ายต่อการตรวจสอบความถูกต้องของโปรแกรม

5.2 ข้อจำกัดของซอฟต์แวร์และข้อเสนอแนะ

ซอฟต์แวร์ Robust-tree ที่ทำหน้าที่สร้างโมเดลของข้อมูลในลักษณะของต้นไม้ตัดสินใจ ที่พัฒนาขึ้นนี้มีข้อเด่นที่สามารถทนทานต่อข้อมูลรบกวน สร้างโมเดลที่มีความแม่นตรงสูง สามารถทำงานกับข้อมูลที่มีค่าแอททริบิวต์เป็นข้อความ (categorical) และตัวเลข (numeric) แต่ซอฟต์แวร์ยังมีข้อจำกัดที่ไม่สามารถทำงานกับข้อมูลชนิดตัวเลขที่เป็นค่าต่อเนื่อง (continuous) ซึ่งกรณีของ continuous data จะต้องมีโปรแกรมอื่นช่วยในลักษณะของการทำ pre-process เพื่อการจัดช่วงของค่าให้เป็นลักษณะ discrete (เรียกว่า การทำ discretization) ก่อนที่จะนำข้อมูลนั้นมาประมวลผลด้วยซอฟต์แวร์ Robust-tree

นอกจากนี้ในกรณีที่ข้อมูลมีขนาดใหญ่มาก ใช้เนื้อที่เกินความจุของหน่วยความจำหลักส่วนที่เป็น data segment ซอฟต์แวร์จะไม่สามารถทำงานได้เนื่องจากเป็นโปรแกรมที่พัฒนาขึ้นเพื่อทำงานกับข้อมูลที่จะต้องอยู่ในหน่วยความจำหลักเท่านั้น แนวทางการพัฒนาต่อไปในอนาคตคือ ปรับปรุงซอฟต์แวร์ให้สามารถทำงานกับข้อมูลในฮาร์ดดิสก์ (หรือ secondary storage ประเภทอื่นๆ) ได้โดยตรง ซึ่งอาจจะทำให้ซอฟต์แวร์ต้องใช้เวลาในการประมวลผลมากกว่าเวอร์ชันที่ใช้อยู่ในปัจจุบัน

อัลกอริทึมที่พัฒนาขึ้นนี้ใช้เทคนิค clustering เป็นเทคนิคหลักในการจัดการกับข้อมูลรบกวน ผู้วิจัยคาดหมายว่าแนวทางอื่นน่าจะใช้ตรวจจับข้อมูลรบกวนได้ดีเช่นเดียวกัน เช่น วิธีวิเคราะห์ความสัมพันธ์ภายในกลุ่มข้อมูล หรือ association analysis เพื่อค้นหาความสัมพันธ์ที่เกิดขึ้นบ่อย โดยมีสมมุติฐานเบื้องต้นว่าข้อมูลรบกวนน่าจะมีลักษณะที่เบี่ยงเบนไปจากลักษณะปกติ ดังนั้นถ้าตรวจสอบจากความสัมพันธ์ที่เกิดขึ้นบ่อยหรือ frequent patterns จึงน่าจะพบข้อมูลที่เป็น noise ได้เช่นเดียวกัน

นอกจากนี้ซอฟต์แวร์ Robust-tree ที่พัฒนาขึ้นยังไม่สามารถทำงานกับข้อมูลที่เกิดขึ้นอย่างต่อเนื่องเช่น ข้อมูลสตรีม การปรับปรุงซอฟต์แวร์ให้รองรับข้อมูลสตรีมได้จึงเป็นอีกแนวทางหนึ่งที่จะพัฒนางานวิจัยนี้ต่อไป

บรรณานุกรม

C. Blake, E. Keogh, and C.J. Merz. UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Department of Information and Computer Science, University of California, Irvine, CA, 1998.

Marko Bohanec and Ivan Bratko. Trading accuracy for simplicity in decision trees. *Machine Learning*, 15:223-250, 1994.

Leo Breiman, Jerome Freidman, Richard Olshen, and Charles Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 988-994, Chambery, France, 28 August-3 September 1993.

P.R. Cohen and D. Jensen. Overfitting explained. *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, Florida.

Floriana Esposito, Donato Malerba, and Giovanni Semeraro. A further study of pruning methods in decision tree induction. *AI&Statistics-95: Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 211-218, Ft. Lauderdale, Florida, 4-7 January 1995.

Jerome H. Freidman. A recursive partitioning decision rule for nonparametric classifiers. *IEEE Transactions on Computers,* C-26: 404-408, April 1977.

S.B. Gelford and C.S. Ravishankar. A tree-structured piecewise-linear adaptive filter. *IEEE Transactions on Information Theory*, 39(6): 1907-1922, November 1993.

Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*, second edition. Morgan Kaufmann, 2006.

Hyunsoo Kim and G.J. Koehler. An investigation on the conditions of pruning an induced decision tree. *European Journal of Operational Research,* 77(1):82, August 1994.

John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2): 227-243, 1989.

John Ross Quinlan. Induction of decision trees. *Machine Learning*, 1:81-106, 1986.

John Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221-234, 1987.

John Ross Quinlan. *C4.5: Programs for machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

John Ross Quinlan and Ronald Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80(3): 227-248, March 1989.

Cullen Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10: 153-178, 1993.

Jan L. Talmon and P. McNair. The effect of noise and biases on the performance of machine learning algorithms. *International Journal of Bio-Medical Computing*, 31(1): 45-57, July 1992.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson, 2006.

C.S. Wallace and J.D. Pattrick. Coding decision trees. *Machine Learning*, 11(1): 7-22, April 1993.

David H. Wolpert. On overfitting avoidance as bias. *Technical Report SFI TR 92-03-5001*, The Santa Fe Institute, 1992.

ภาคผนวก ก

บทความวิจัยนำเสนอในการประชุมวิชาการ

Proceedings of IADIS Multi Conference on Computer Science and Information Systems,
Amsterdam, Netherlands, 22-27 July 2008.

A DECLARATIVE PROGRAMMING PARADIGM AND THE DEVELOPMENT OF KNOWLEDGE MINING AGENTS

Nittaya Kerdprasop and Kittisak Kerdprasop
Data Engineering and Knowledge Discovery (DEKD) Research Unit,
School of Computer Engineering, Suranaree University of Technology,
Nakhon Ratchasima, Thailand

ABSTRACT

Agent is a conceptual entity designed to solve a complex problem. It differs from other software design concepts with its special capabilities of acting autonomously, adapting to changing circumstances, and communicating with other agents through high-level interactions. The significance of the agent-based approach in data mining, knowledge discovery, and Web intelligence has been realized by many researchers over the past decade. Several agent-based data mining tools have been developed. Most of them were implemented with imperative languages such as C and Java. We propose the agent model that has been implemented with a more powerful programming paradigm using declarative languages such as Haskell and Prolog. The advantages of these languages are their advancement in program structures, pattern matching and reasoning features, including higher order computation and meta-level programming. These language features are essential in developing intelligent agents. Even though the major drawback of most declarative languages is their computation speed, we have shown via experimental results that the percentage of speed decrease is insignificant comparing to imperative language implementation.

KEYWORDS

Knowledge mining agents, machine intelligence.

1. INTRODUCTION

Agents are key players in most current intelligent systems. According to Russell and Norvig [1995], agent is an entity (either a computer or a human) that perceives and acts in a particular environment. It is also defined [Wooldridge, 1997; Wooldridge and Jennings, 1995] as a computer system designed to work in some environment and has the capability to act autonomously in order to meet its designed goals. In complex and dynamic environments, multiagents are often utilized as a collaborative group of performers. A multiagent system [Weiss, 1999] is a group of entities working together to perform tasks that are beyond the individual capabilities of each entity. Agents may co-exist on a single processor, or they may be physically separated to perform activities on their own and build a community through communication. Intelligent agents [Wooldridge, 2002] employ additional capabilities of goal-directed task accomplishment, response due to changes in their environments, ability to interact with other agents, and learning to improve performances as they perform their assigned tasks.

To achieve intelligence, agents utilize several artificial intelligence techniques such as machine learning, inductive and deductive reasoning. On the contrary, intelligent agent technology can play an important role in the design and development of knowledge discovery, or data mining, systems. Knowledge discovery is the process of identifying valid, novel, potential useful and understandable patterns in data that may be distributed and heterogeneous in terms of content and structures [Fayyad et al., 1995; Han and Kamber, 2006]. This complex discovery process involves several phases including data selection, data preprocessing, data transformation, data analysis (or mining), interpretation and evaluation. These phases are iterative and adaptive in their nature, therefore it is a good setting for the application of intelligent agent technology. The ability of an agent to communicate, cooperate, and coordinate with other agents in multiagent system benefits the design of knowledge discovery tools to locate and mine potential knowledge in a distributed environment.

The application of agent technology as a major method to the implementation of data mining techniques has been studied by many researchers. Kargupta et al. [1997] applied agent technology to design a parallel and distributed data mining system named PADMA (Parallel Data Mining Agents). The system interfaces with users via a Web browser. Bose and Sugumaran [1999] designed an agent-based data mining system called the Intelligent Data Miner (IDM). They implemented a prototype of IDM using Java language and Java Agent Template Lite (JATLite available from http://java.stanford.edu) which is a set of Java templates and agent infrastructure. Some researchers proposed to employ heterogeneous techniques to perform data mining tasks. Recon [Kerber et al., 1995] is an example of a hybrid system containing inductive, clustering, case-based reasoning and statistical package for data mining. Zhang and Zhang [2004] also implemented data mining based hybrid intelligent systems. They demonstrated agent perspectives through the re-implementation of Weka system [Witten and Frank, 2005] using the agent communication language KQML (Knowledge Query and Manipulation Language) [Finin et al., 1997]. Gao et al. [2005] proposed a model called CoLe (Cooperative Learning) to handle the situation that agents employ different methods to access different types of information in heterogeneous data sets. Ong et al. [2005] also developed a multiagent system based on the concept of data stream processing to perform a data mining task in distributed dynamic environments.

An agent-based approach has been widely accepted as an appropriate paradigm to implement an intelligent system because of its flexibility, modularity and ability to take advantage of distributed resources. The integration of heterogeneous data source is one major characteristic of practical data mining systems that have to search for interesting patterns from huge amount of data, possibly locating at remote sites. An agent technology is thus the promising technique in the knowledge discovery setting that real-world data is evolving, distributed and non-homogeneous. Pursuing the same direction as other researchers, we also propose an agent-based model to implement knowledge discovery. We, however, consider a different paradigm on the agent-based data mining implementation. Instead of implementing with imperative paradigm using common languages such as C, Java, Visual Basic, we employ the declarative paradigm and implement the system with Haskell and Prolog languages. The power of declarative programming has paid of as shown in our experimental results. The rest of this paper is organized as follows. The next section briefly discusses the concepts of declarative versus imperative programming. We then present our agent model in section 3. The detail and some excerpts of our implementation are explained in section 4. Section 5 illustrates the experimental results. Section 6 concludes the paper.

2. DECLARATIVE VERSUS IMPERATIVE PROGRAMMING

In declarative languages such as Haskell and Prolog, programs are sets of definitions and recursion is the main control structure of the program computation. In imperative languages such as C and Java, programs are sequences of instructions and loops are the main control structure. A functional programming language like Haskell is a declarative language in which programs are sets of *function* definitions. The evaluation of a program is simply the evaluation of functions. A logic programming language like Prolog is a declarative language in which programs are sets of *predicate* definitions. Predicates are truce or false when applied to an object or set of objects, while functions return a result. A predicate typically has one more argument (to serve as a returned value) than the equivalent function. Either function or predicate definitions, each definition has a dual meaning: (1) it describes what is the case, and (2) it describes the way to compute something.

Declarative languages are mathematically sound. It is easy to prove that a declarative program meets its specification which is a very important requirement in software industry. Declarative style makes a program better engineered, that is, easier to debug, easier to maintain and modify, and easier for other programmers to understand. The examples of coding quick sort in C, Haskell and Prolog (figure 1) verify the previous statement.

One major task in data mining is searching for frequent patterns. A pattern is a set of items co-occurrence across a database. Given a candidate pattern, the task of pattern matching is to search for its frequency looking for the patterns that are frequent enough. The outcome of this search is frequent patterns that suggest strong co-occurrence relationships between items in the dataset. The search for patterns of interest can be efficiently programmed using the Haskell language. Haskell has evolved as a strongly typed, lazy, pure functional language since 1987 [Hudak et al., 1996].

Haskell

```
sort [ ]     = [ ]
sort (x:xs) =  sort [y | y<-xs, y<x ]
               ++ [x]
               ++ sort [y | y<-xs, y>=x]
```

Prolog

```
qs([ ],[ ]).
qs([X|Xs]) :- part(X, Xs, Littles, Bigs),
              qs( Littles, Ls),
              qs( Bigs, Bs),
              append(Ls, [X|Bs], Ys).
part(_, [ ], [ ], [ ]).
part(X, [Y|Xs], [Y|Ls], Bs) :-  X>Y,
              part(X, Xs, Ls, Bs).
part(X, [Y|Xs], Ls, [Y|Bs]) :-  X=<Y,
              part(X, Xs, Ls, Bs).
```

C

```
int partition(int y[], int f, int l);
void quicksort(int x[], int first, int last) {
   int pivIndex = 0;
   if(first < last) {
            pivIndex = partition(x,first, last);
      quicksort(x,first,(pivIndex-1));
      quicksort(x,(pivIndex+1),last);
      } }
int partition(int y[], int f, int l) {
   int up,down,temp;
   int cc;    int piv = y[f];
   up = f;
   down = l;
   do { while (y[up] <= piv && up < l) { up++;}
        while (y[down] > piv  ) { down--; }
        if (up < down) {  temp = y[up];
                          y[up] = y[down];
                          y[down] = temp;   }
     } while (down > up);
   temp = piv;
   y[f] = y[down];
   y[down] = piv;
   return down;}
```

Figure 1. Quick sort program in Haskell, Prolog, and C languages.

Pattern matching is one of the most powerful features of Haskell. Defining functions by specifying argument patterns is a common practice in programming with Haskell. As an illustration, consider the following example:

```
fib :: Int -> Int        -- declaring a function that takes one Int and returns an Int
fib 0 = 0                    -- pattern 1: argument is 0
fib 1 = 1                -- pattern 2: argument is 1
fib n = fib (n-2) + fib (n-1)     -- pattern 3: argument is Int other than 0 and 1
```

The function fib returns the n^{th} number in the Fibonacci sequence. The left hand sides of function definitions contain patterns such as 0, 1, n. When applying a function these patterns are matched against actual parameters. If the match succeeds, the right hand side is evaluated to produce a result. If it fails, the next definition is tried. If all matches fail, an error is returned.

In Prolog, the feature of pattern matching can be defined through the use of arguments. For example, the following program demonstrates the fib function (in Prolog it is called predicate instead of function) to find the n^{th} number in the Fibonacci sequence. Last argument is normally a place holder for an output.

```
% Fibonacci function in Prolog
fib(0, 0).              % pattern 1: input number is 0, then output is 0
fib(1, 1).              % pattern 2: input number is 1, then output is 1
fib(N, F) :-  N > 1,          % pattern 3: input number >1, then
        N1 is N-1, N2 is N-2,          % create new variables: N1 and N2
            fib(N1, F1), fib(N2, F2),        % recursively call fib
            F is F1 + F2.                    % compute final result F
```

3. THE AGENT-BASED MODEL

We propose an agent-based knowledge discovery model (as shown in figure 2) to compose of three layers: data source layer, agent layer, and external layer. A community of agents is in the agent layer situated to help users to access and get only promising knowledge for their discovery tasks. Locating and accessing, filtering, and mining are three major activities of these agents.

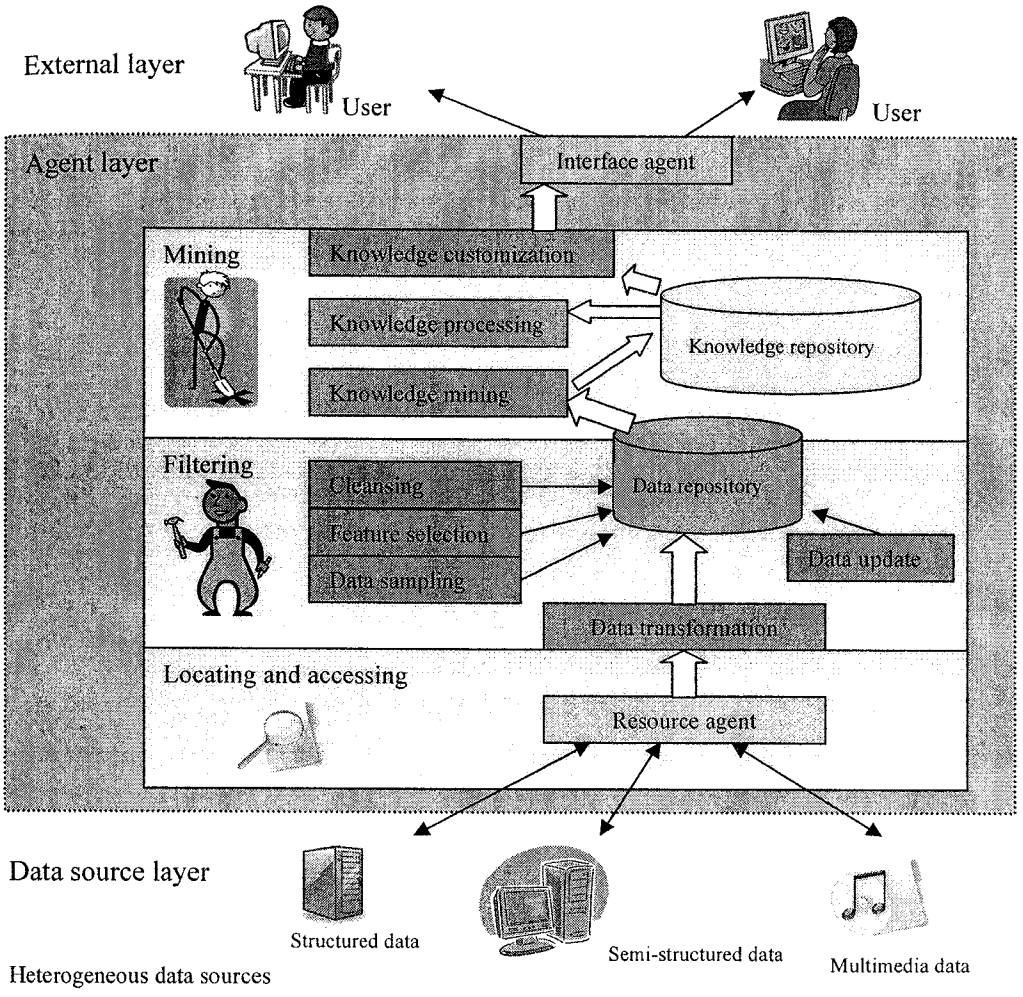


Figure 2. Agent-based model in a knowledge discovery system.

Locating and accessing. At the lowest level of the proposed framework, multiple heterogeneous data sources are located in an enterprise environment. These data sources may be distributed across a network such as intranet or internet. Resource agent is thus responsible for making the underlying data available to the data transformation agent in the upper filtering sub-layer. The resource agent also monitors changes in data contents to report any corresponding modification to the data-update agent. To implement the functions of resource agent, the following modules are required.

- *Data source specification.* The resource agent must be able to announce its location and the specification of its contents to other agents in the community.
- *Query processor.* The agent has to handle the update and the query upon the data contents. The query processor must also have the ability to reason whether its data contents match the needs announced by the knowledge mining agent.
- *Event-detection module.* This module is responsible for detecting the update on the data contents.
- *Data access module.* The resource agent assigns different access modules for different kinds of data sources.

Filtering. The agents in this class are the most autonomous and sophisticated ones due to the self-adjusting and specific functioning of each agent. The agents in this class are composed of:

- *Data update agent.* The agent communicates with resource agent to probe any changes in the environment and reflect those changes to data repository.
- *Data transformation agent.* Its main responsibility is to turn the input data to the right format.

- *Cleansing agent.* This agent is responsible for getting rid of any noise and handling missing values in the data contents.
- *Feature selection agent.* This agent efficiently evaluates and selects the most promising features out of the available data.
- *Data sampling agent.* This agent is invoked to obtain representatives appropriate for a specific mining task.

Mining. The agents in this class are mainly responsible for performing the data mining techniques. Data obtained from the filtering sub-layer will be turned into valuable and actionable knowledge by these agents:

- *Knowledge mining agent.* It is actually a group of agents, each agent performs a specific mining technique.
- *Knowledge processing agent.* Mined knowledge could be overwhelming or low accurate. It is thus the responsibility of this agent to post-process knowledge discovered by the mining agents.
- *Knowledge customization agent.* Some knowledge might be accurate but uninterested to the user. This agent is responsible for getting only knowledge pertaining to each user interest and delivers customized knowledge through the interface agent.

4. IMPLEMENTATION

We implemented association mining to discover frequent patterns with Apriori algorithm [Agrawal et al., 1993; Agrawal and Srikant, 1994]. Some parts of the program are shown in figure 3. In Haskell, each item is represented by the item identifier which is an integer. Thus, a set of patterns (patternset) is denoted as a set of Int declared in the first line of the Haskell code. The function sumi is defined to count the number of occurrence of each element in patternSet. Functions listC and listC' perform the task of enumerating candidate frequent patternSet. Only patternSet that satisfy the *minS* threshold are reported from the functions listL and listL' as frequent patternSet. The complete implementation of frequent pattern discovery using Haskell functional language takes only 37 lines of code.

Prolog implementation to discover frequent patterns contains around 58 lines of code. In Prolog, data type definition is not necessary because Prolog is weakly typed. Thus, pattern matching in Prolog is more general than that of Haskell. We use the set union to construct candidate patterns of length two or more as in Haskell implementation.

```
patternSet :: [Set Int]
patternSet =[Set.singleton x | x<-[1..9]]
sumi::Set Int->[Set Int]->Int
sumi s [] =0
sumi s (y:ys)|(Set.isSubsetOf s y)= 1+(sumi s ys)
             |otherwise = (sumi s ys)
listC ::Int->[(Set Int,Int)]
listC 1=[let n=(sumi s dataB) in (s,n) |
                s<- patternSet]
listC n=[let n=(sumi s dataB) in (s,n) |
                        s<- Set.toList(listC' n)]
listC' :: Int->Set(Set Int)
listC' 2=Set.fromList[(Set.union x y) |x<-(listL' 1),
                      y<-(listL' 1),x/=y]
listC' n=Set.fromList[(Set.union x y) |
                x<-(listL' (n-1)),
                y<-(listL' (n-1)),  x/=y,
               (Set.size(Set.union x y))==n]
listL ::Int->[(Set Int,Int)]
listL n=[(x,y)|(x,y)<-listC n, y>=minS]
listL'::Int->[Set Int]
listL' n =[x|(x,_)<-listL n]
```

(a) Haskell implementation

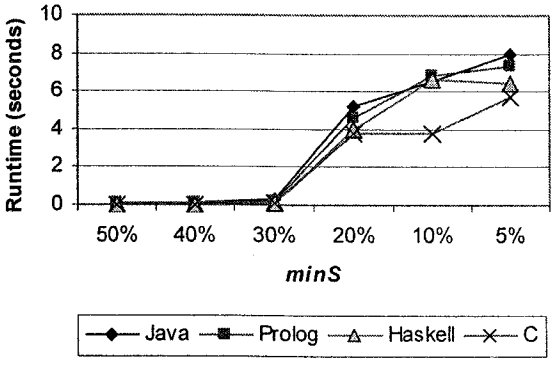
```
r1:- n(X), cL1(X).
r2(X):- cC2(X).
clear:-retractall(l1(_)),  retractall(c1(_)),
       retractall(c2(_)) , retractall(l2(_)).
% Create L1
cL1([]).
cL1([H|T]) :-  findall(X,f([H],X),L), length(L, Len),
               Len >= 2 , !,
               cL1(T), assert(l1(([H], Len)))
               ;
               cL1(T).
% Create C2, L2
cC2(X) :- l1((X,_)), l1((X2,_)), X\==X2,
          write(X-X2), union(X, X2, Res),
          assert(c2((Res))), retract(l1((X,_))),  nl.
crC2(L) :- findall(X,c2(X),L).
cL2([]).
cL2([H|T]) :- findall(X,f(H,X),L), length(L,Len),
              Len >= 2 ,!, cL2(T),
              assert(l2((H,Len)))    ;   cL2(T).
f(H, X) :-  item(X), subset(H, X).
```

(b) Prolog implementation

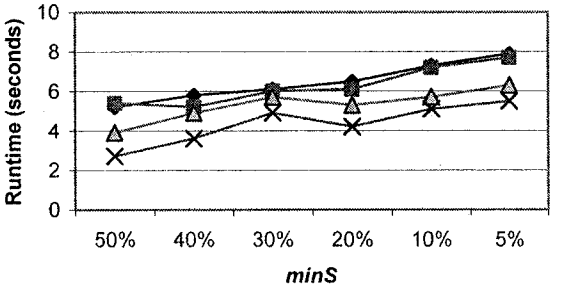
Figure 3. Frequent pattern discovery implemented with declarative languages.

5. EXPERIMENTATION

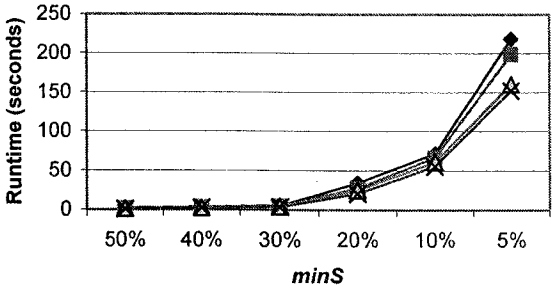
We comparatively study the performance of our implementations of frequent pattern discovery using Haskell and Prolog versus C and Java (source codes of C and Java implementations are taken from [Borgelt, 2003]). All experimentations have been performed on a 796 MHz AMD Athlon notebook with 512 MB RAM and 40 GB HD. We tested the speed and memory usage of the programs on different datasets obtained from the UCI Machine Learning Database Repository (http://www.ics.uci.edu/~mlearn/MLRepository.html). Some results on four datasets, vote data (13.2 KB, 300 transactions, 17 items), chess data (237 KB, 2130 transactions, 37 items), DNA data (252 KB, 2000 transactions, 61 items), and mushroom data (916 KB, 5416 transactions, 23 items) are shown in this section. The frequent pattern discovery implementations have been tested on each dataset with various *minS* (minimum support) values.



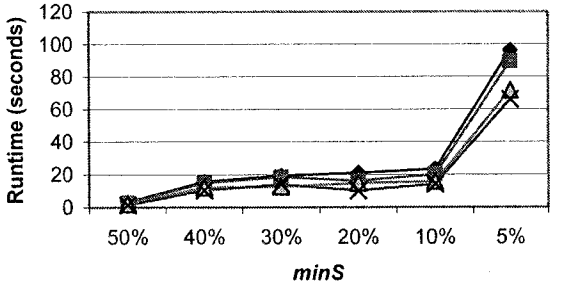
(a) vote data



(b) chess data



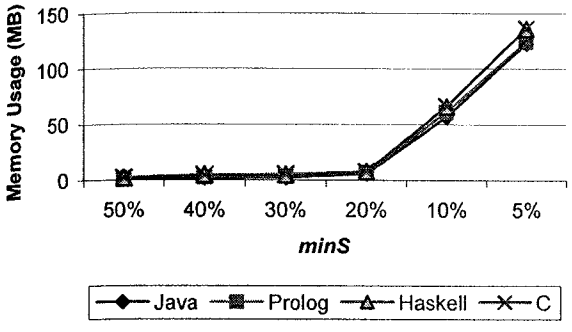
(c) DNA data



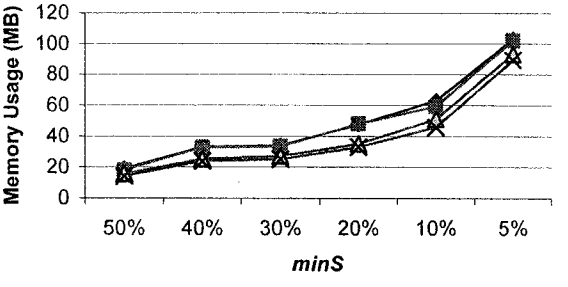
(d) mushroom data

Figure 4. The comparison on computation speed of declarative versus imperative programming.

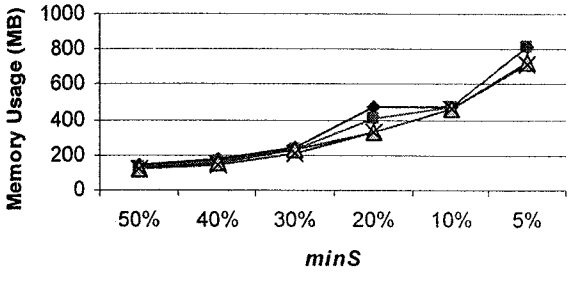
It can be noticed from the experimental results that on a speed comparison (figure 4), C implementation is the fastest, Haskell comes at a second fastest following by Prolog and Java. On the memory usage comparison (figure 5), the ordering is the same as those on the speed comparison. However, it can be noticed from the results that the degree of difference is insignificant and almost negligible. When taking into consideration the length of the source codes, Haskell: 37 lines, Prolog: 58 lines, C: 352 lines, Java: 663 lines, the declarative style of coding absolutely consumes less effort and development time than the coding with imperative style.



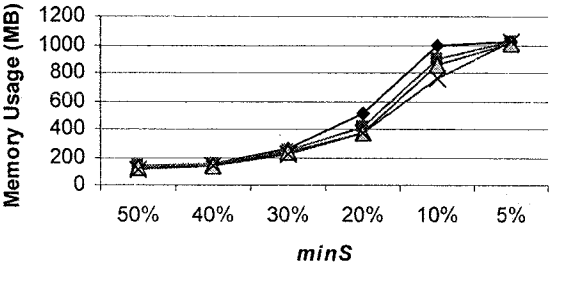
(a) vote data



(b) chess data



(c) DNA data



(d) mushroom data

Figure 5. Memory usage comparison of declarative versus imperative programming.

6. CONCLUSION

The contribution of this paper is the design and implementation of a knowledge discovery system to provide an integrated, flexible, and efficient platform supported by a community of agents. This platform provides mechanisms of data browsing and extracting, data arrangement, data quality evaluation, data mining, knowledge processing and knowledge customization for the whole process of knowledge discovery. The agent model is designed with the three-layer architecture. Data source layer is at the back-end responsible for locating and accessing data from the remote sites. External layer is the user interface part. The core of our design is the agent layer which is in the middle between the external and the data source layers. Agent layer is divided into three sub-layers: locating and accessing, filtering, and mining. These agents work autonomously and cooperatively to deliver knowledge assets that meets specific interest of each user.

The proposed agent model has been implemented with declarative programming using Haskell and Prolog languages. We employ this paradigm with the intuitive idea that the problem of knowledge discovery should be efficiently and concisely implemented with high-level declarative languages. This idea has been tested on a specific problem of frequent pattern discovery which is a major problem in the areas of data mining and business intelligence. The problem concerns finding frequent patterns hidden in a large database. Frequent patterns are patterns such as set of items that appear in data frequently.

Coding in declarative style takes less effort because pattern matching is a fundamental feature supported by functional and logic languages. The implementations of Apriori algorithm using Haskell and Prolog confirm our hypothesis about conciseness of the program. The performance studies also support our intuition on efficiency because our implementations are not significantly less efficient than C or Java implementations in terms of speed and memory usage.

This preliminary study supports our belief regarding declarative programming paradigm towards a complex problem of knowledge discovery. We focus our future research on the design of data organization to optimize

the speed and storage requirement. We also consider the extension of implementation in the course of concurrency to improve its performance.

Agents are designed to be active and intelligent. They are able to react appropriately to unpredictable situations, evaluate and apply their own problem solving strategies. However, the current design has to be extensively tested on various application domains. Several areas of extensions are currently being investigated. The functionalities of filtering agents can be extended to support new techniques of cleansing and adaptive sampling. Mining agents are also in the course of further improvement.

ACKNOWLEDGEMENT

This work was supported by the Thailand Research Fund under grant RMU-5080026 and the National Research Council of Thailand. The authors are with the Data Engineering and Knowledge Discovery (DEKD) research unit which is fully supported by research fund of Suranaree University of Technology.

REFERENCES

Agrawal, R. et al., 1993. Mining association rules between sets of items in large databases. *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 207-216.

Agrawal, R. and Srikant, R., 1994. Fast algorithm for mining association rules. *Proceedings of International Conference on Very Large Data Bases*, pp. 487-499.

Borgelt, C., 2003. *Frequent Item Sets Miner for FIMI 2003.* http://www.borgelt.net/software.html.

Bose, R. and Sugumaran, V., 1999. Application of intelligent agent technology for managerial data analysis and mining. *In The Data Base for Advances in Information Systems*, Vol.30, No.1, pp. 77-94.

Fayyad, U. et al., 1995. From data mining to knowledge discovery: An overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth (eds.), *Advances in Knowledge Discovery and Data Mining*. AAAI Press, pp. 1-34.

Finin, T. et al., 1997. KQML as an agent communication language. In J. Bradshaw (ed.), *Software Agents*, AAAI Press/ The MIT Press, pp. 291-316.

Gao, J. et al., 2005. A cooperative multi-agent model and its application to medical data on diabetes. *Proceedings of International Workshop on Autonomous Intelligent Systems: Agents and Data Mining*, pp. 93-107.

Han, J. and Kamber, M., 2006. *Data Mining: Concepts and Techniques, 2nd edition.* Morgan Kaufmann.

Hudak, P. et al., 1996. A gentle introduction to Haskell. *Technical Report Yale U/DCS/RR-901*, Yale University.

Kargupta, H. et al., 1997. Scalable, distributed data mining using an agent based architecture. *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pp. 211-214.

Kerber, R. et al., 1995. A hybrid system for data mining. In S. Goonatilake and S. Khebbal (eds.), *Intelligent Hybrid System*, John Wiley & Sons, pp. 121-142.

Ong, K. et al., 2005. Agents and stream data mining: A new perspective. *In IEEE Intelligent Systems*, Vol.20, No.3, pp. 60-67.

Russell, S. and Norvig, P., 1995. *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Weiss, G. (ed.), 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* The MIT Press.

Witten, I. and Frank, E., 2005. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd edition.* Morgan Kaufmann.

Wooldridge, M., 1997. Agent-based software engineering. *In IEE Proceedings on Software Engineering*, Vol.144, No.1, pp. 26-37.

Wooldridge, M., 2002. *An Introduction to Multiagent Systems.* John Wiley & Sons.

Wooldridge, M. and Jennings, N., 1995. Intelligent agents: Theory and practice. *In The Knowledge Engineering Review*, Vol.10, No.2, pp. 115-152.

Zhang, Z. and Zhang, C., 2004. Constructing hybrid intelligent systems for data mining from agent perspectives. In N. Zhong and J. Liu (eds.), *Intelligent Technologies for Information Analysis*, Springer, pp. 333-359.

Proceedings of 31st Congress on Science and Technology of Thailand, Suranaree University of Technology, Nakhon Ratchasima, Thailand, 18-20 October 2005.

การศึกษาเพื่อเปรียบเทียบประสิทธิภาพของเทคนิคการตัดกิ่งต้นไม้ตัดสินใจ
A COMPARATIVE STUDY OF METHODS FOR PRUNING DECISION TREES

นฤพนต์ ว่องประชานุกูล, นิตยา เกิดประสพ และ กิตติศักดิ์ เกิดประสพ

Narupon Wongprachanukul, Nittaya Kerdprasop and Kittisak Kerdprasop

Data Engineering and Knowledge Discovery (DEKD) Research Unit, School of Computer Engineering, Suranaree University of Technology, Nakhon Ratchasima, Thailand,
E-mail address: narupon@nsrc.or.th, nittaya@sut.ac.th, kerdpras@sut.ac.th

บทคัดย่อ: งานวิจัยนี้เป็นการศึกษาเพื่อเปรียบเทียบประสิทธิภาพของเทคนิคการตัดกิ่งต้นไม้ตัดสินใจที่มีชื่อเสียงสองวิธีคือ Reduced-error pruning และ Error-based pruning โดยมีจุดมุ่งหมายเพื่อวิเคราะห์ค่าความเที่ยงตรงในการจำแนกคลาสข้อมูล เวลาที่ใช้ในการสร้างโมเดล และขนาดของต้นไม้ตัดสินใจ เราทำการทดลองกับสิบชุดข้อมูลด้วยเทคนิคการตัดกิ่งเหล่านี้ เพื่อลดขนาดของต้นไม้และแก้ไขปัญหา "overfitting" ต้นไม้ที่ได้รับการตัดกิ่งแล้วจะใช้เวลาในการสร้างลดลงเนื่องจากขนาดที่เล็กลง และยังคงสามารถจำแนกข้อมูลใหม่ได้อย่างถูกต้อง

Abstract: We make a comparative study of two well-known pruning methods, reduced-error pruning and error-based pruning. The predictive accuracy, the time taken to build the model, and size of the pruned trees are evaluated for each pruning method. We conduct the experiments on ten data sets. Pruning methods aim at simplifying decision trees to avoid overfitting problem. The pruned trees result in faster classification and do not decrease their predictive accuracy.

Introduction: Decision tree is one of the tools used for data mining. The main application area is classification task. The model is built from a set of records, called training set. Each record consists of a number of attribute-value pairs. One of these attributes represents class of the record. We also have a test set for evaluating the performance of a decision tree.

When a decision tree is built, many of the branches may be overly expanded due to noise or outliers in the training set. The built model is too complex, since it tries to classify all records in the training set including noise and outliers. This problem is called "overfitting". We use tree pruning method to remove the least reliable branches, generally resulting in faster classification and improvement in the ability of the tree to correctly classify unknown data.

We study the performance of the post-pruning approach. A tree node is pruned by removing its branches from a fully grown tree (T_{max}) [1]. In the following subsections, we summarize the concepts of two pruning methods whose performances are evaluated in this paper.

Reduced-error pruning (REP): This method is probably the simplest pruning technique. It uses the pruning set to evaluate the goodness of a subtree of the complete tree. It starts with T_{max} and runs the test data through it. For each internal node, the number of classification errors is counted if the subtree is kept compared with if the subtree is pruned.

The decision on whether or not to prune the subtree is based on which alternative yields a minimum error.

A pruning operation involves replacing a subtree by a leaf. REP will perform this operation if it does not increase the total number of classification errors. Traversing the tree in a bottom-up strategy ensures that the result is the smallest pruned tree that has minimum error on the pruning data.

Error-based pruning (EBP): This method is implemented by the well-known decision tree inducer C4.5 [3]. Unlike REP, EBP uses the training set for building and simplifying trees. It visits nodes of T_{max} according to a bottom-up traversal strategy and uses the certainty factor (CF) parameter to control the pruning. CF is used to estimate the upper limit of the probability that an error occurs over the population at a leaf.

The subtree replacement is performed if the error estimate for the expected leaf is not greater than the sum of the error estimates for the current leaf nodes of the subtree. EBP also performs a pruning operation called "subtree raising" that replaces a subtree with its most populated branch if this does not increase the estimated error.

Methodology: We conducted experiments and used the decision tree on ten data sets from UCI Machine Learning Repository [2] with the above pruning methods and use the decision tree inducer C4.5. Model accuracy was tested with ten-fold cross-validation technique. The main characteristics of the data sets are presented in Table 1.

Table 1. The main characteristics of the data sets used for experiments

Data set	No. of Instances	No. of Attributes	No. of Nominal attributes	No. of Numeric attributes	Missing values	No. of Classes
Anneal	898	38	32	6	yes	5
Audiology	226	69	69	0	yes	24
Glass	214	9	0	9	no	7
Glass-2	163	9	0	9	no	2
Hepatitis	155	19	13	6	yes	2
Ionosphere	351	34	0	34	no	2
Iris	150	4	0	4	no	3
Labor	57	16	8	8	yes	2
Soybean	683	35	35	0	yes	19
Vote	435	16	16	0	yes	2

Results, Discussion and Conclusion: We compare the predictive accuracy, the time taken to build the model and size of the pruned trees with the unpruned trees. These results are reported in Table 2.

Table 2. Accuracy, time taken to build the model and size of the pruned trees compare with the unpruned trees

	Time (s)			Tree sizes			Accuracy (%)		
Data set	REP	EBP	unpruned	REP	EBP	unpruned	REP	EBP	unpruned
Anneal	0.55	1.32	2.14	113	78	155	92.87	90.98	93.10
Audiology	0.11	0.22	0.22	47	54	62	71.24	77.43	76.55
Glass	0.11	0.22	0.17	17	59	59	71.50	66.82	65.89
Glass-2	0	0.05	0.05	15	17	17	74.23	80.37	80.37
Hepatitis	0.05	0.11	0.06	13	21	31	81.94	78.71	78.06
Ionosphere	1.65	2.14	1.70	13	35	35	90.60	88.03	88.32
Iris	0	0.06	0	9	9	9	94.67	96.0	96.0
Labor	0	0.05	0	7	5	22	82.46	73.68	78.95
Soybean	0.27	0.55	0.38	120	93	175	87.55	91.51	91.36
Vote	0.06	0.06	0.06	9	11	37	95.63	96.32	96.32

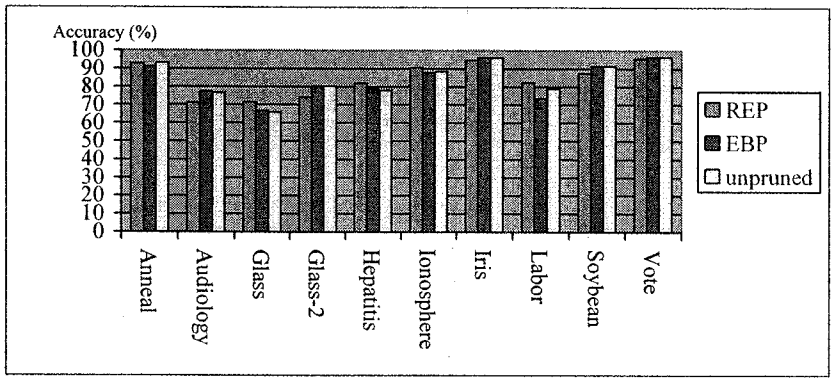


Figure 1. The predictive accuracy of the pruned trees compares with the unpruned trees

Both REP and EBP reduce the size of a fully grown tree by removing some unnecessary branches and do not significantly decrease the predictive accuracy of most final trees. REP produces the pruned tree in the shortest period of time, since it must not estimate classification errors. These experiments are still preliminary and need more systematic and extensive studies including additional comparative studies to other pruning methods.

References: [1]. Esposito, F., Malerba, D., and Semeraro, G. *A Comparative Analysis of Methods for Pruning Decision Trees.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 19, 5, 476-491. 1997.

[2]. Merz, C.J., and Murphy, P.M. *UCI Repository of machine learning databases.* http://www.ics.uci.edu/~mlearn/MLRepository.html. 1996.

[3]. Quinlan, J.R. *C4.5: Programs for Machine Learning.* 1993.

Keywords: decision trees, pruning methods, reduced-error pruning, error-based pruning.

ชื่องานวิจัย : **การศึกษาเปรียบเทียบวิธีการลดความซับซ้อนของโมเดลข้อมูล**

คณะผู้วิจัย : ผู้ช่วยศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ, ผู้ช่วยศาสตราจารย์ ดร.นิตยา เกิดประสพ และ นายนฤพนต์ ว่องประชานุกูล

ผู้นำเสนอผลงานวิจัย : นายนฤพนต์ ว่องประชานุกูล

สังกัด : หน่วยปฏิบัติการวิจัยวิศวกรรมข้อมูลและการค้นหาความรู้, สาขาวิชาวิศวกรรมคอมพิวเตอร์, สำนักวิชาวิศวกรรมศาสตร์, มหาวิทยาลัยเทคโนโลยีสุรนารี

ที่อยู่สำหรับติดต่อ : 111 ถนนมหาวิทยาลัย, ต.สุรนารี, อ.เมือง, จ.นครราชสีมา 30000

โทรศัพท์: 044-224432 อีเมล์: nittaya@ccs.sut.ac.th

กลุ่มวิชา : กลุ่มงานวิจัยด้านวิศวกรรมศาสตร์

<u>วัตถุประสงค์ :</u>

งานวิจัยนี้มีจุดมุ่งหมายที่จะศึกษาเปรียบเทียบเทคนิคต่างๆ ที่ใช้ในการลดความซับซ้อนของโมเดลข้อมูล โดยมุ่งเน้นที่โมเดลประเภทต้นไม้ตัดสินใจที่นิยมใช้มากในงานทำเหมืองข้อมูลประเภทการจำแนกข้อมูลอัตโนมัติ และการแสดงลักษณะร่วมของข้อมูล โครงสร้างต้นไม้ตัดสินใจที่มีขนาดใหญ่เกินไปจะซับซ้อนเข้าใจยากและนำไปสู่ปัญหาสำคัญคือ เป็นโมเดลที่จำเพาะมากเกินไป (overfitting) การหาวิธีลดความซับซ้อนโดยคงความเที่ยงตรงของโมเดล จะเป็นประโยชน์ต่อการพัฒนาอัลกอริทึมสังเคราะห์โมเดลเพื่อการจำแนกที่มีประสิทธิภาพและเที่ยงตรงสูง

<u>วิธีการ :</u>

เทคนิคหลักของการลดความซับซ้อนที่จะใช้ในการศึกษาวิจัยนี้ จะประกอบด้วยเทคนิคที่ใช้วิธีวิเคราะห์ความสัมพันธ์ในเชิงสถิติเพื่อตัดบางส่วนของโครงสร้างต้นไม้ที่ไม่ก่อประโยชน์ทิ้งไป เทคนิคที่จะนำมาใช้ศึกษาเปรียบเทียบคือ การกำหนดเงื่อนไขแบบฮิวริสติกเพื่อกำหนดระดับที่เหมาะสมในการตัดโครงสร้างต้นไม้ ผลจาก

การศึกษาวิเคราะห์เปรียบเทียบจะช่วยให้สามารถพัฒนาเทคนิคใหม่ที่ช่วยให้การสร้างและกำหนดความซับซ้อนของโมเดลในลักษณะต้นไม้ตัดสินใจมีประสิทธิภาพมากขึ้น การลดความซับซ้อนจะส่งผลให้ลดเวลาในการสังเคราะห์โครงสร้างต้นไม้ นอกจากนี้ยังช่วยลดเนื้อที่หน่วยความจำที่ต้องใช้ในการเก็บแต่ละกิ่งของโครงสร้างต้นไม้ ประโยชน์โดยตรงของการใช้หน่วยความจำลดลงคือช่วยให้โปรแกรมสังเคราะห์โมเดลทำงานกับข้อมูลขนาดใหญ่ได้

ผลที่ได้ :

ข้อมูลที่ใช้ในการทดลองนี้เป็นข้อมูลมาตรฐานที่นิยมใช้ในการศึกษาเปรียบเทียบประสิทธิภาพการทำเหมืองข้อมูล รายละเอียดของข้อมูลแสดงได้ดังตารางที่ 1 ผลการทดสอบเปรียบเทียบประสิทธิภาพการลดความซับซ้อนของโมเดลและคุณภาพของโมเดลที่ได้ระหว่างวิธีวิเคราะห์ความสัมพันธ์ในเชิงสถิติและวิธีเชิงฮิวริสติกที่ใช้พื้นฐานจากทฤษฎีสารสนเทศ แสดงได้ดังตารางที่ 2 และ 3 ตามลำดับ

ตารางที่ 1 รายละเอียดข้อมูลที่ใช้ในการทดลอง

ชื่อชุดข้อมูล	จำนวนข้อมูลฝึก	จำนวนข้อมูลทดสอบ	จำนวนแอททริบิวต์	จำนวนแอททริบิวต์ชนิดสัญลักษณ์	จำนวนแอททริบิวต์ชนิดตัวเลข
Adult	32,561	16,281	15	8	6
Credit-German	666	334	21	13	7
Hepatitis	103	52	20	13	6
Mushroom	5,416	2,708	23	22	-
Vote	300	135	17	16	-

ตารางที่ 2 ผลการทดสอบเปรียบเทียบประสิทธิภาพการลดความซับซ้อนของโมเดลระหว่างวิธีวิเคราะห์ความสัมพันธ์ในเชิงสถิติและวิธีเชิงฮิวริสติกที่ใช้พื้นฐานจากทฤษฎีสารสนเทศ แสดงด้วยเปอร์เซ็นต์การลดจำนวนโหนดในโครงสร้างต้นไม้ที่เป็นโมเดลผลลัพธ์

ชื่อชุดข้อมูล / วิธีลดความซับซ้อนของโมเดล	Adult	Credit-German	Hepatitis	Mushroom	Vote
วิธีเชิงฮิวริสติกที่ใช้พื้นฐานจากทฤษฎีสารสนเทศ	22.12%	34.27%	26.95%	35.72%	41.12%
วิธีวิเคราะห์ความสัมพันธ์ในเชิงสถิติ	12.96%	25.18%	36.54%	38.78%	47.83%

ตารางที่ 3 ผลการทดสอบเปรียบเทียบประสิทธิภาพของโมเดลที่ได้จากการลดความซับซ้อนด้วยวิธีวิเคราะห์ความสัมพันธ์ในเชิงสถิติและวิธีเชิงฮิวริสติกที่ใช้พื้นฐานจากทฤษฎีสารสนเทศ

ชื่อชุดข้อมูล / วิธีลดความซับซ้อนของโมเดล	Adult	Credit-German	Hepatitis	Mushroom	Vote
วิธีเชิงฮิวริสติกที่ใช้พื้นฐานจากทฤษฎีสารสนเทศ	82.13%	74.25%	76.92%	95.79%	91.85%
วิธีวิเคราะห์ความสัมพันธ์ในเชิงสถิติ	82.99%	75.15%	86.54%	98.71%	97.04%

<u>สรุปผลการทดลอง</u> :

การลดความซับซ้อนของโมเดลจากทั้งสองวิธีการจากผลการทดลองจะเห็นได้ว่าวิธีการทางสถิติให้ผลลัพธ์เป็นโมเดลที่มีความเที่ยงตรงสูงกว่า แต่ขนาดของโมเดลจะใหญ่กว่าโมเดลที่ได้จากวิธีการเชิง ฮิวริสติก แนวทางการพัฒนาในอนาคตคือการสร้างวิธีการลดความซับซ้อนที่มีคุณภาพสูงเทียบเท่ากับวิธีการเชิงสถิติแต่ให้ขนาดของโมเดลที่เล็กลง

Proceedings of the 1st KMITL International Conference on Integration of Science and Technology for Sustainable Development, August 25-26, 2004, Lad Krabang, Bangkok, Thailand, pages 275-278

THE IMPACT OF NOISE AT DIFFERENT DATA ATTRIBUTES

Nittaya Kerdprasop, Kittisak Kerdprasop, Laksamee Khomnotai and Thammasak Thianniwet

Data Engineering and Knowledge Discovery (DEKD) Research Unit
School of Computer Engineering
Suranaree University of Technology
Nakhon Ratchasima 30000, Thailand
E-mail: nittaya, kerdpras@ccs.sut.ac.th

ABSTRACT

Real-world data often suffer from corruptions or noise. The most serious negative impact of noise is that it can reduce machine learning performance in terms of learning accuracy. Most learning algorithms have integrated various approaches to handle noisy data. However, rare research has been conducted to systematically explore the impact of noise, especially when noise occurs at different attributes. We investigate the effect of class noise, noise in principal attributes, and noise in irrelevant attributes to the learning accuracy. Our conclusions can be served as a preliminary step toward the designing of handling mechanisms for a specific kind of noise.

Keywords
Noise, Class noise, Attribute noise, Noise impact

1. INTRODUCTION

Noise is a random error in data. Noisy data contain incorrect attribute values caused by many possible reasons, for instance, faulty data collection instruments, human errors at data entry, errors in data transmission. If noise occurs in the training data, it can lower the performance of the learning algorithm. The most serious effect of noise is that it can confuse the learning algorithms to produce complex and distorted results. The long and complex results are due to the attempt to fit every training data into the concept descriptions. This situation is named the overfitting problem.

Most learning algorithms are designed with the awareness of noisy data. Thus, there exist some mechanisms in dealing with noise, for example, the ID3 algorithm[3] uses the prepruning technique to avoid growing a decision tree too deep down to cover the noisy training data. Some algorithms adopt the technique of postprocessing to reduce the complexity of the learning results. Postprocessing technique includes the cost-complexity pruning, reduced error pruning, and pessimistic pruning described in Quinlan[4,5].

Even though most existing learning algorithms include various noise-handling techniques, the existence of noise can still affect the learning results negatively. The focus of this paper is to observe the impact of noise to the learning algorithms. We categorize noise into three groups: class noise, noise in principal attributes, and noise in irrelevant attributes. We investigate the relationship between various groups of noise and learning accuracy. Our conclusions can be used to enhance the handling techniques specific to the noise of different types.

2. EXPERIMENTAL METHODOLOGY

We study the impact of noise on three data sets: Monk1 (124 instances), Ionosphere (234 instances), and Chess (2,130 instances). These data sets are UCI[2] standard data for testing the performance of machine learning algorithms. We generate noise varying from 0% to 45% on different groups of attributes. Class noise[1] is an occurrence of random error in target attribute (i.e., the instances are mislabeled). Attribute noise is a random error in predicting attributes. Every attribute except a class attribute has an equal probability of noise occurrence. Instead of simply studying attribute noise, we extend our investigation to the impact of noise that occurs in the principal attributes, i.e., attributes highly correlate to class prediction, and the impact of noise if it occurs at less-relevant attributes.

We test the impact of noise on two learning algorithms: naive Bayes, and neural

network. These algorithms are known as a noise-tolerant system. We compare their noise-tolerant performance when noise is introduced to the class attribute, the principal predicting attributes, and attributes irrelevant to the prediction. For consistency, we supply the same data set to each test.

3. RESULTS AND DISCUSSIONS

The tolerance against class noise of naive Bayes and neural network algorithms tested on three data sets is shown in Figure 1. The effects of noise in principal attributes and irrelevant attributes are shown in Figures 2 and 3, respectively.

The experimental results reveal that:

(1) Class noise has more impact on the neural network algorithm than on the naive Bayes. This negative effect can be noticed clearly on a large data set (Chess data).

(2) When noise occurs in highly predictive (or principal) attributes, it has much more effect on the neural network algorithm than the naive Bayes.

(3) For the small data set (Monk1), noise in irrelevant attributes has no effect on both algorithms. But on larger data set, this kind of noise can slightly degrade the performance of the neural network algorithm.

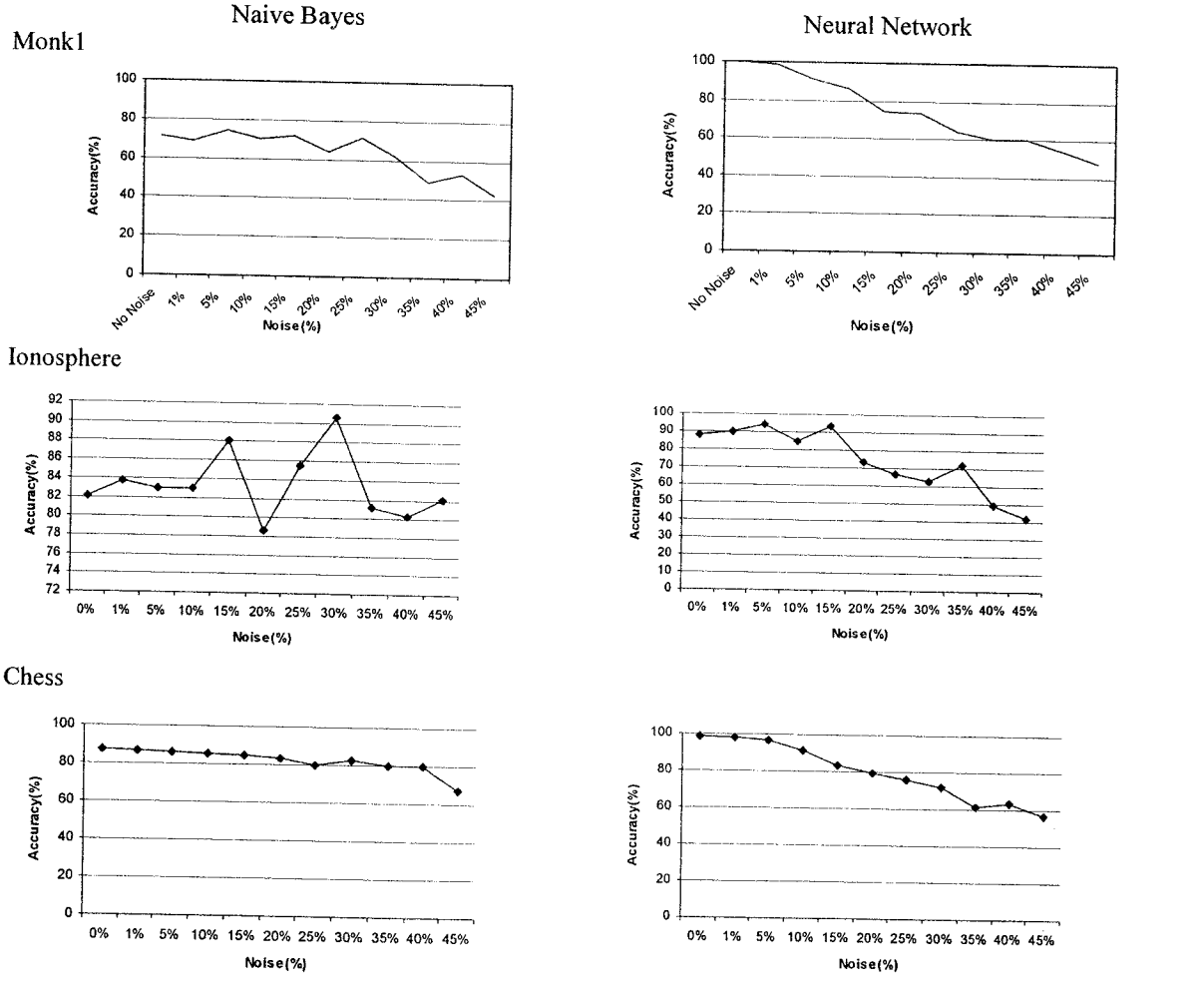


Figure 1. The effect of class noise

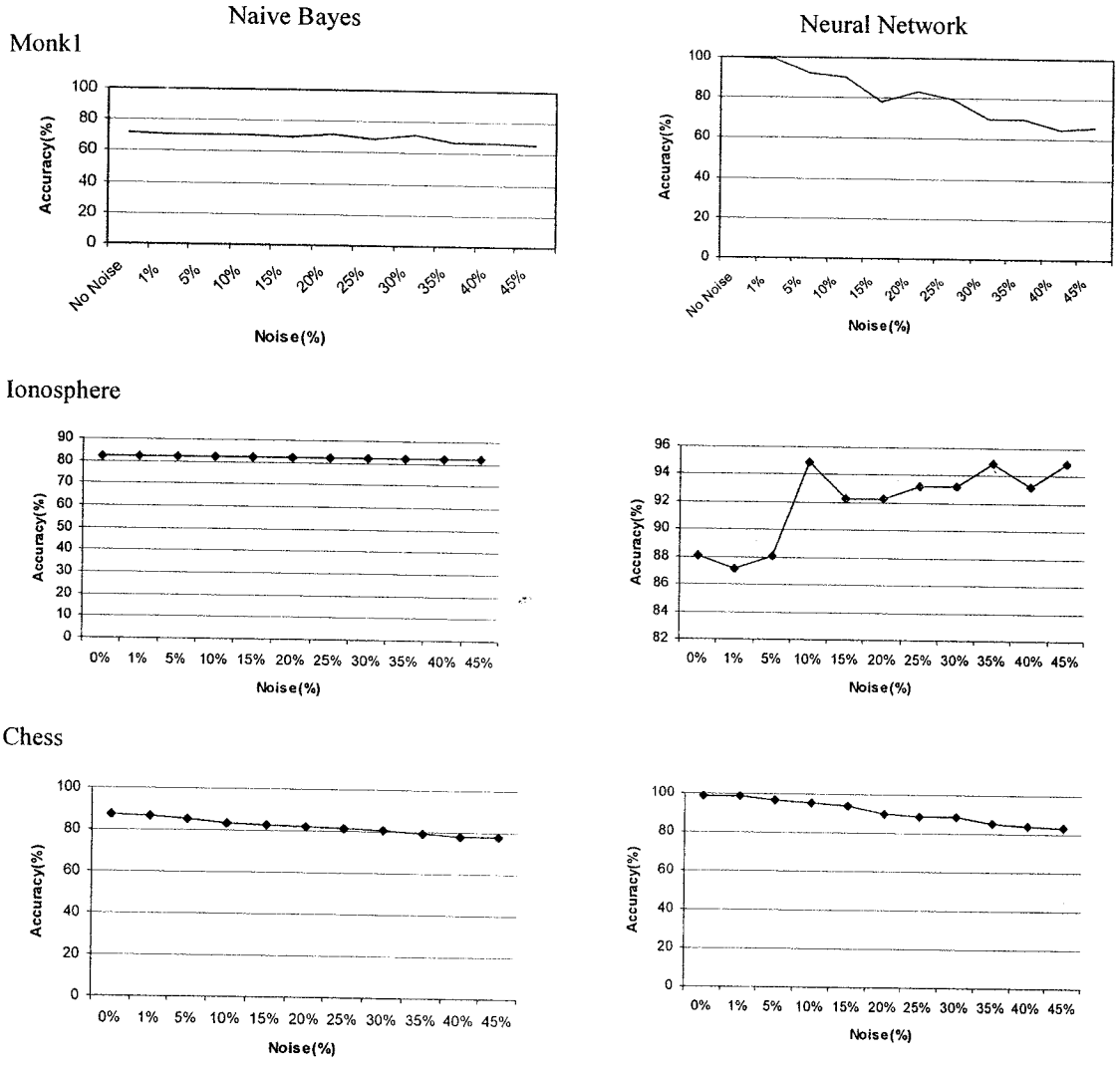


Figure 2. The effect of principal attribute noise

4. CONCLUSION

Noise in a data set can happen in different forms: (1) misclassification or wrong labeled instances, (2) erroneous or distorted attribute values, (3) contradictory or duplicate instances having different labels, (4) missing attribute values. All kinds of noise can more or less affect the learning performance. We specific our investigation to the fist two kinds of noise, which are termed class noise and attribute noise, respectively. Class noise has been studied extensively by many researchers, whereas attribute noise is less thoroughly studied. We extend the study on attribute noise by categorize it further to principal attribute noise and irrelevant attribute noise. We find that principal attribute noise has more negative impact on the learning performance than the class noise. Noise in irrelevant attributes can somehow affect the neural network algorithm. Our future research is to extend our study and to design a handling mechanism specific to each kind of noise.

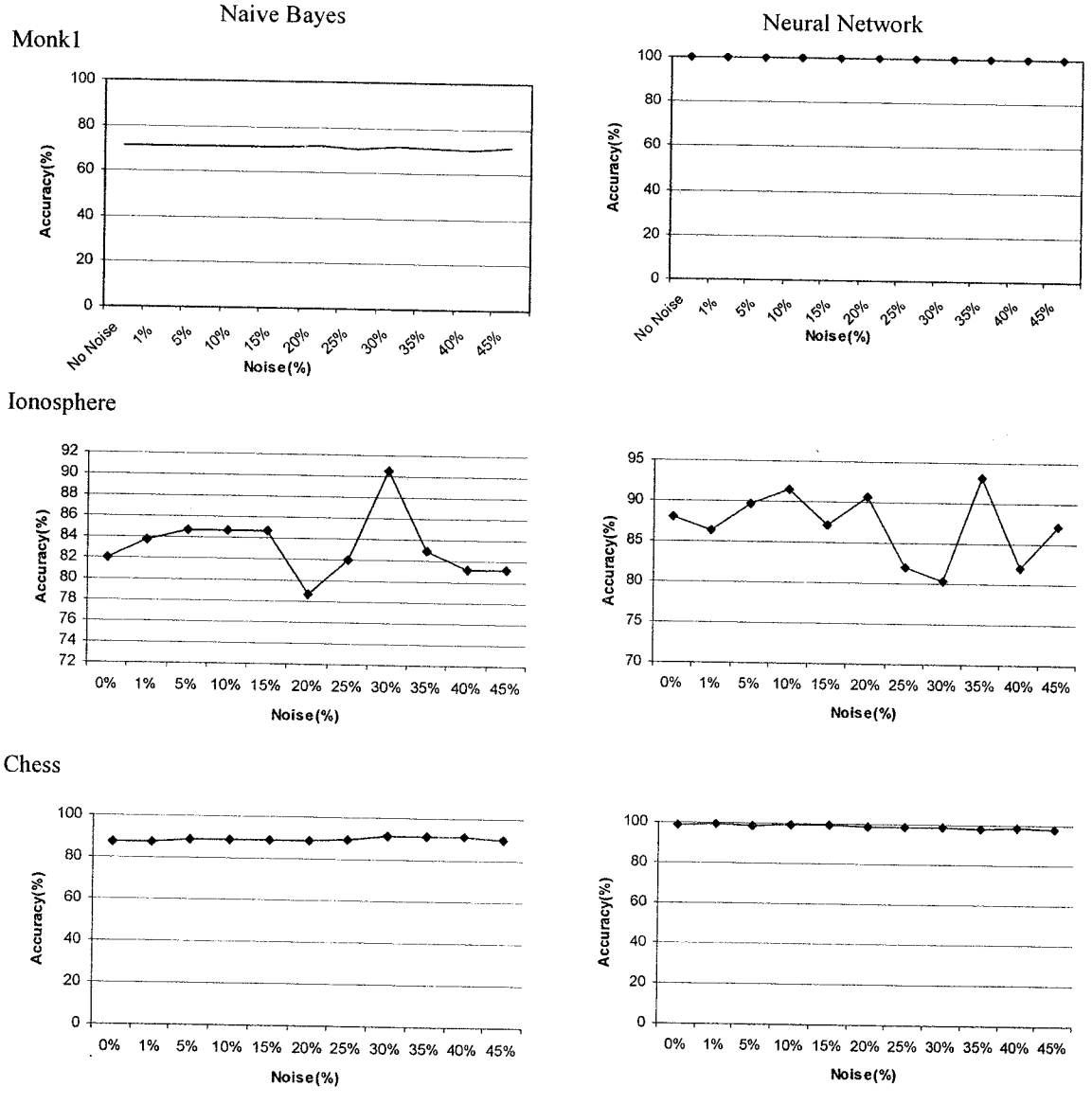


Figure 3. The effect of irrelevant attribute noise

5. REFERENCES

1. Angluin, D., and Laird, P. (1988). *Learning from noisy examples*. Machine Learning, 2, 343-370.

2. Merz, C.J., and Murphy, P.M. (1997). *UCI Repository of machine learning database.* http://www.ics.uci.edu/~mlearn/MLRepository.html

3. Quinlan, J.R. (1986). *Induction of decision tree*. Machine Learning, 1, 81-106.

4. Quinlan, J.R. (1989). *Simplifying decision tree*. In B. Gaines and J. Boose (Editors), Knowledge Acquisition for Knowledge Based Systems, Vol. 1, Academic Press.

5. Quinlan, J.R. (1992). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Proceedings of 1st International Forum on Information and Computer Science (IFICT 2(
Shizuoka University, Japan, 9-10 January 2003, pp. 114-118

Data Partitioning for Incremental Data Mining

Nittaya Kerdprasop and Kittisak Kerdprasop

School of Computer Engineering, Suranaree University of Technology
111 University Avenue, Muang District, Nakorn Ratchasima 30000, THAILAND
nittaya , kerdpras @ccs.sut.ac.th

ABSTRACT

Data repositories of interest in data mining applications can be very large. Many of the existing learning algorithms do not scale up to extremely large data set. One approach to deal with this problem is to apply the concept of incremental learning. However, incremental data mining is not the same as incremental machine learning. The former handles one subset of data at a time, whereas the latter handles a single data instance at a time. The size of data subset determines both the performance and speed of the mining process. We thus focus the study on the partitioning of a data into a proper subset and propose an algorithm to return a data subset for both classification and association mining tasks. We also perform a set of experiments to observe the behavior of classification and association data mining on various data partitioning. The experimental results confirm our criteria on data partitioning.

Keywords: data mining, incremental, data partitioning

1 INTRODUCTION

Data mining is the process of extracting useful information such as previously unknown patterns or association hidden in a large data set [4]. Recent advances in digital information storage and data acquisition technologies have made it possible to acquire and store large volumes of data. Therefore, data repositories of interest in data mining applications are normally very large. Many of the existing mining algorithms do not scale up to extremely large data sets. One approach to deal with this problem is to partition the huge data set into several subsets of manageable size, then learn (probably in parallel) from each subset and finally combine the learning results [3].

Another approach is to employ the incremental machine learning paradigm. Incremental machine learning is the technique to avoid retaining all training data in main memory. Instead, the learning algorithm learns from one data instance at a time and tune the result accordingly [2]. However, incremental data mining is not the same as incremental machine learning. Incremental data mining handles subsets of data one set at a time, not just a single data instance as in the incremental machine learning [16]. Thus, partitioning the data set to a proper subset (or sample) size is certainly beneficial the incremental mini to reach its high accuracy in an acceptable period of time

We propose an algorithm to partition the original lar data set into a manageable and yet learning-effective da subset. We also perform experiments on learni performance of different data partitions on the tv common data mining tasks: classification and associatic On data classification, we investigate learning curves classifiers on various data partitions. For the derivation association rules, we compare the set of rules obtain from each data partitions against the rules derived from t whole complete data set.

This paper is organized as follows. Section 2 gives overview of incremental data mining. Section 3 describ the data-partitioning algorithm. Section 4 explains t experimental setup. Section 5 presents the results and discussion. Section 6 concludes our work.

2 INCREMENTAL DATA MINING

Machine learning techniques can be broad categorized as either batch or incremental [5]. Bat learning examines a whole collection of data set a induces a learning result. In incremental learning, da subsets D_1 , D_2 , ... , D_n are assumed to become availab to the learner at discrete time intervals, and the learner also assumed being unable to store collectively all the da fragments. Thus, it can only maintain and update learning result as new data fragment becomes available.

Sutton and Whitehead [11] have distinguished tv kinds of incremental learning: weakly and strict incremental method. A learning method is weak incremental if it requires additional memory a computation in order to process one additional da instance. Examples of weakly incremental learners are ID [12,13,14] (an incremental version of ID3 [8]) a nearest neighbor algorithms. A learning method is strict incremental if its memory and computation (per da instance) requirements do not increase with the number instances. The learners in this category are STAGGER[1 and most connectionist learning methods.

Recent research on learning ensembles of classifiers [is relevant to incremental data mining. Learning ensembl of classifiers, such as bagging [1] and boosting, generat multiple versions of classifiers by running the learni algorithm many times on a set of re-sampled data. T classification results are combined using a majority vo Each version of the classifier is generated from a sample

the original data set, and each data instance can be used in many samples. To adopt the ensemble method to the setting of incremental mining, -- for instance, Learn++ algorithm [7] -- each data instance in the original data set is partitioned into only one subset and used only once in the learning process. As mentioned in the first section that incremental data mining learns a subset – not just a single data instance, we are however still of limited knowledge about what proper size data partitioning should be. Therefore, we design an algorithm to generate a proper data subsets and test the algorithm on different data sizes to observe the efficiency of incremental data mining.

3 DATA PARTITIONING ALGORITHM

This section describes the algorithm to partition the large data set into a manageable and proper size. The algorithm returns the data partition for both the classification task and association task.

Algorithm Data_Partition

Input: (1) A relational database R.
(2) Predictive level l, the default predictive level can be raise to a 'high' level.
(3) Maximum number of instances, m, that data mining tool can handle.

Output: D_C = a data subset for classification, and
D_A = a data subset for association

Steps:

1. $\mu_{default} = 0.1$, $\mu_{high} = 0.3$
   /* Set parameter for the classification data subset */
2. $\gamma_{default} = 0.2$, $\gamma_{high} = 0.5$
   /* Set parameter for the association data subset */
3. *Sampling_size_classification* = min{ m, ($\mu_{default}$ * number of instances in R) }
   /* Sampling size for classification task at the default predictive level */
4. *Sampling_size_association* = min{ m, ($\gamma_{default}$ * number of instances in R) }
   /* Sampling size for association task at the default predictive level */
5. If l = 'high' then
   5.1 *Sampling_size_classification* = min{ m, (μ_{high} * number of instances in R) }
   5.2 *Sampling_size_association* = min{ m, (γ_{high} * number of instances in R) }
6. return
   D_C = { r_i | r_i = *Sampling(R)* , i = 1,2, ..., *Sampling_size_classification* }
   D_A = { r_i | r_i = *Sampling(R)* , i = 1,2, ..., *Sampling_size_association* }
   /* *Sampling(R)* is the random sampling without replacement from the database R */

4 EXPERIMENTS

We design the experiments to study the effect varying the data partitioning on the efficiency of da mining. The two kinds of data mining task being explor are classification and association. On classification, t algorithms J48 and naïve Bays are selected as a benchma to test the quality of each data partition. J48 is a r implement [15] of C4.5 [9], which is the most well-knov decision tree-based classification algorithm. The advanta of tree-based classifier is its simple and comprehensib representation format. Naïve Bayes is a statistical classifi that can learn the concept rapidly. This high learning ra property makes naïve Bayes a candidate algorithm f incremental data mining. Therefore, we decide to test t classification accuracy on these two classificati algorithms.

The accuracy is estimated on the basis of number test instances correctly classified by the induced classifi The estimation method that we use is the holdout meth in which 66% of the data instances is used for the traini purpose and the remaining 34% is used as the test set.

For the task of association rule derivation, we empl the APRIORI algorithm [15]. The criteria to test the rul deriving efficiency on each data partition is the number association rules that match the rules derived from t whole data set. The two data sets -- mushroom a connect-4-game – used in our experiments are taken fro the UCI repository [6].

5 RESULTS AND DISCUSSION

Table 1 and 2 show the learning results classification and association rule derivation, respectivel The learning curves of J48 and naïve Bayes on each da set are illustrated in Figure 1. Figure 2 graphical compares the quality of association rule derivation various data partitions.

The classification results reveal the fast-learni characteristic of naïve Bayes algorithm. It requires only 10 % of the data set to reach its highest learning accurac This fast-learning property is the major ingredient incremental data mining. When taking association in consideration together with the classification, we may inf from the experimental results that the appropria partitioning (or windowing) should be at the 10% of t whole data set. These results agree with our heuristic setting the threshold parameter in the range 0.1-0.5 for t expected proper size of sample. In the distributed setting which the exact size of the data set could not be guessed advance, the fixed amount of 800-1,000 instances shou give the satisfiable learning result.

Table 1: The classification accuracy on different data partitioning

Data partitioning	Accuracy tested on mushroom data		Accuracy tested on connect-4-game data	
	naïve Bayes (% correct classification)	J48 (% correct classification)	naïve Bayes (% correct classification)	J48 (% correct classification)
1 %	53.5714 %	57.1429 %	60.8696 %	56.9565 %
5 %	61.1511 %	61.5705 %	68.1462 %	68.6684 %
10 %	60.6498 %	64.2599 %	71.0057 %	74.1837 %
20 %	61.1212 %	70.5244 %	71.3975 %	75.8163 %
30 %	61.3993 %	68.2750 %	70.9041 %	76.9700 %
40 %	62.6244 %	69.1403 %	72.7471 %	78.8202 %
50 %	61.7221 %	64.7612 %	71.5107 %	79.1990 %
60 %	63.0277 %	66.5862 %	72.2392 %	79.6691 %
70 %	63.8056 %	67.9938 %	72.1500 %	81.6966 %
80 %	63.0317 %	68.0090 %	72.2627 %	82.1724 %
90 %	62.3492 %	66.0901 %	72.1956 %	83.6405 %
100 %	63.9884 %	60.0796 %	72.1332 %	79.0814 %

Table 2: The quality of association-rule derivation on different partitioning

Data partitioning	mushroom dataset [1]		connect-4-game dataset [2]	
	number of instances	number of rules matched [3]	number of instances	number of rules matched [3]
1 %	81	27	675	53
5 %	406	27	3,377	90
10 %	812	62	6,755	88
20 %	1,624	62	13,511	92
30 %	2,437	62	20,267	95
40 %	3,249	65	27,022	97
50 %	4,062	66	33,778	98
60 %	4,874	100	40,534	98
70 %	5,686	79	47,289	99
80 %	6,499	100	54,045	99
90 %	7,311	100	60,801	99

[1] The complete mushroom data set contains 8,124 instances.

[2] The complete connect-4-game contains 67,557 instances.

[3] Number of rules that matched with the association rules derived from the complete data set.

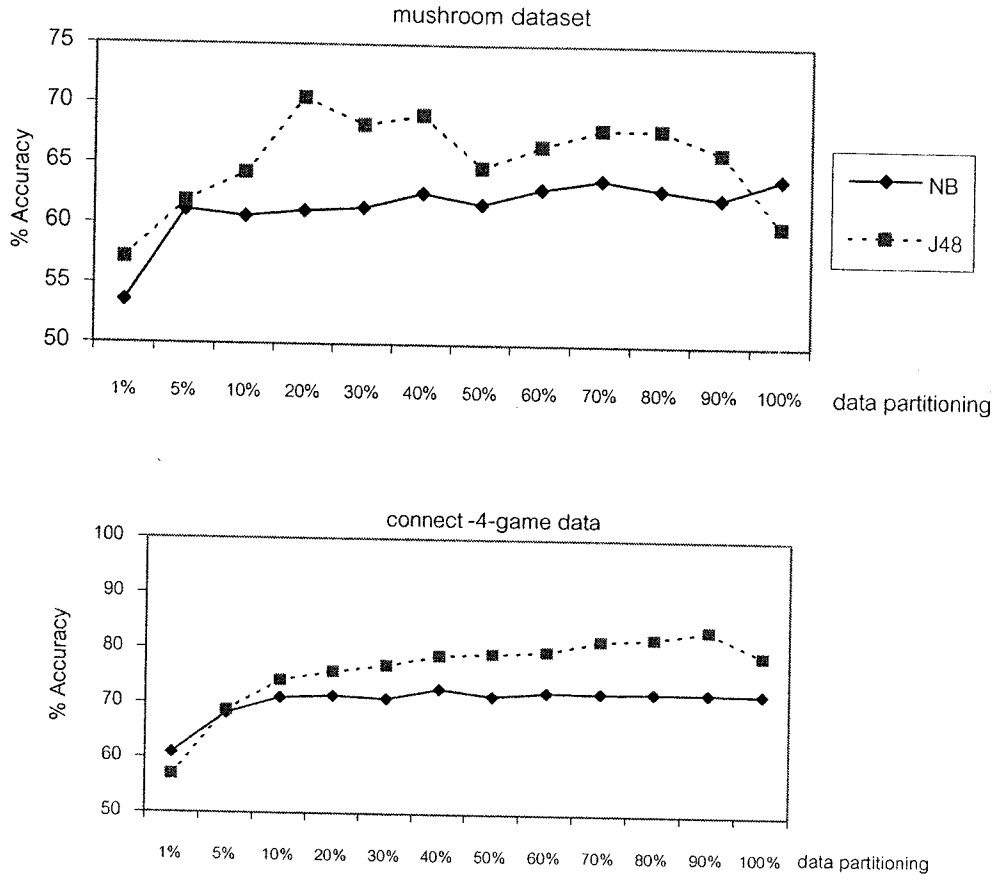


Figure 1: Learning curves on classifying the incremental mushroom data set and connect-4-game data set

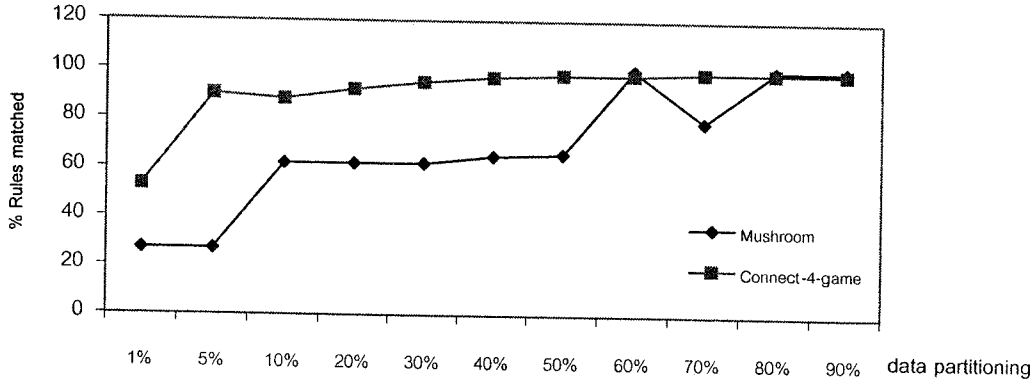


Figure 2: The comparison on quality of association rules derived from each data partitioning

CONCLUSION

Recent advances in data storage and acquisition techniques have made it possible to produce increasingly large data repositories. Many of the existing mining algorithms do not scale up to extremely large data sets. One approach to this problem is to partition the data set into several subsets of manageable size, then learn in parallel or incrementally from each subset. The partitioning of data set into appropriate size is the main focus of our study. We propose an algorithm to use the heuristic to do the sampling for the proper size of data subsets. We perform experiments on the two data mining tasks -- classification and association -- using mushroom and connect-4-game data sets. We can conclude from the results that partitioning the data set at the threshold level 0.1-0.5 yields an acceptable classification accuracy, and moderate to high quality association rules.

REFERENCES

[1] L. Breiman, "Arcing classifiers", Annals of Statistics, 26, 1998.

[2] S.H. Clearwater, T.P. Cheng, H. Hirsh, and B.G. Buchanan, "Incremental batch learning", Proceedings of the Sixth International Workshop on Machine Learning, Morgan Kaufmann, 1989.

[3] T.G. Dietterich, "Machine learning research: Four current directions", AI Magazine, 18(4), 1997, 97-136.

[4] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data mining*, AAAI/MIT Press, Cambridge, MA, 1996.

[5] M. Harries, C. Sammut, and K. Horn, "Extracting hidden context", Machine Learning, 36(2), 1998, 101-126.

[6] C.J. Merz and P.M. Murphy, Uci Repository of machine learning databases, 1996. [http://www.ics.uci.edu/ ~mlearn/MLRepository.html]

[7] R. Polikar, L. Udpa, S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for multilayer percentron networks", Proceedings of the IEEE Conference on Acoustic, Speech, and Signal Processing (ICASSP), 2000.

[8] J.R. Quinlan, "Induction of decision trees", Machine Learning, 1, 1986, 81-106.

[9] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

[10] J.C. Schlimmer and R.H. Granger, "Incremental learning from noisy data", Machine Learning, 1, 1986, 317-354.

[11] R.S. Sutton and S.D. Whitehead, "Online learning with random representations", Proceedings of the Tenth International Conference on Machine Learning, Morgan Kaufmann, 1993, 314-321.

[12] P. Utgoff, "ID5: An incremental ID3", Proceedings of the Fifth International Conference on Machine Learning, Morgan Kaufmann, 1988, 107-120.

[13] P. Utgoff, "Incremental induction of decision trees", Machine Learning, 4, 1989, 161-186.

[14] P. Utgoff, "An improved algorithm for incremental induction of decision trees", Proceedings of the Eleventh International Conference on Machine Learning, Morgan Kaufmann, 1994, 318-325.

[15] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with java Implementations*, Morgan Kaufmann, San Francisco, 2000. [software accessible via the URL http://www.cs.waikato.ac.nz/ml/weka]

[16] X. Wu and W. Lo, "Multi-layer incremental induction", Proceedings of the Fifth Pacific Rim International Conference on Artificial Intelligence, Springer-Verlag, 1998.

Proceedings of 29th Congress on Science and Technology of Thailand, Khon Kaen University, Thailand, 20-22 October 2003

การศึกษาเปรียบเทียบเทคนิคการจัดการข้อมูลสูญหายในการทำเหมืองข้อมูลประเภทงานจำแนก

A Comparative Study of Techniques to Handle Missing Values in the Classification Task of Data Mining

นิตยา เกิดประสพ, กิตติศักดิ์ เกิดประสพ, ยอด สายแวว, ปรีชา พุ่มรุ่งเรือง

Nittaya Kerdprasop, Kittisak Kerdprasop, Yawd Saiveaw, Preecha Pumrungreong
School of Computer Engineering, Suranaree University of Technology, 111 University Ave., Muang District, Nakorn Ratchasima 30000, Thailand; e-mail address: nittaya@ccs.sut.ac.th

บทคัดย่อ: งานวิจัยนี้เป็นการศึกษาเปรียบเทียบเทคนิคต่างๆ ที่ใช้จัดการกับกรณีข้อมูลบางส่วนสูญหายในกระบวนการวิเคราะห์ข้อมูลอัตโนมัติด้วยเทคนิคการทำเหมืองข้อมูล โดยเน้นเฉพาะงานจำแนกประเภทข้อมูล ข้อมูลที่ใช้ในการทดลองนำมาจากแหล่งข้อมูลมาตรฐานของมหาวิทยาลัยแคลิฟอร์เนียที่เออร์ไวน์ โดยเลือกมาทั้งข้อมูลประเภทจำนวนเลขและประเภทข้อความ ข้อมูลมาตรฐานถูกจำลองให้มีบางส่วนสูญหาย จากนั้นใช้เทคนิคที่แตกต่างกันสี่เทคนิคเพื่อเติมส่วนที่สูญหาย การทดสอบประสิทธิภาพของเทคนิคการเติมข้อมูลสูญหายใช้อัลกอริทึมการจำแนกประเภทข้อมูลด้วยวิธีเบย์ส์อย่างง่าย วิธีสร้างต้นไม้ตัดสินใจเชิงอุปนัย และวิธีการคำนวณระยะห่างของข้อมูล ผลการทดลองชี้ว่าถ้าข้อมูลเป็นประเภทจำนวนเลข การตัดทิ้งเรคคอร์ดที่มีข้อมูลสูญหายจะให้ประสิทธิภาพการจำแนกประเภทข้อมูลที่ดีกว่า ในขณะที่กรณีข้อมูลประเภทข้อความการเติมข้อมูลสูญหายด้วยสัญลักษณ์ "?" จะให้ประสิทธิภาพการจำแนกประเภทข้อมูลที่ดีกว่า

Abstract: We study and review the techniques for dealing with missing attribute values in data mining. Then, we conduct the experiments to observe the performance of classification algorithms on each strategy of missing-value substitution. The algorithms we used are naïve Bays, tree-based and instance-based classifiers. Four approaches of handling missing values are introduced to the numeric and nominal data sets taken from the UCI repository. The experimental results reveal the superior suggestive choice of ignoring numerical data instances with missing values, whereas replacing the unknown values with the symbol "?" produces a better classification results for the nominal data set.

Introduction: Data mining, also known as knowledge discovery in databases (KDD), is the process of extracting (or mining) useful knowledge from large volume of data. The different kinds of mined knowledge lead to the different tasks of data mining, for instance, concept description, association, classification, prediction, clustering, trend analysis. Among the diversity on mining tasks, classification is the most extensively studied one. Classification is the process of inducing a set of models from the training data. These models can describe and distinguish important data classes. The main purpose of inducing models is to use them predicting the class of the future data whose class label is unknown.

Obviously, data play an important role in the process of data mining. The quality of the collected data can directly improve the efficiency of the subsequent mining process.

However, data collected in the real-world tend to be incomplete due to some values are missing. This might occur because the value is not relevant to a particular case, was not recorded when the data was collected, or is unspecified by users because of privacy concerns [1]. The incomplete data in which a large percentage of the entries are missing causes a problem since most data mining algorithms assume that the data is completely specified.

The problem of missing values has been investigated since the last two decades [3,6]. The simple solution is to discard the data instances with some missing values [8]. A more sophisticated solution is to try to determine these values [4]. However, techniques to guess the missing values must be efficient, otherwise the replacement may introduce noise.

In this paper, we empirically study the effect on the mining performance of different techniques for dealing with missing values. Several techniques to handle missing values have been discussed in the literature [2,4,6,7]. Some popular methods are as follows:

(1) Ignore and discard the tuples with missing values: This is a simple solution, but not very effective. However, it is recommended if the tuple contains several attributes with missing values.
(2) Replace all missing attribute values with a global constant: Some examples of a global constant are "unknown", "missing", "- ∞".
(3) Replace the missing attribute values with its attribute mean: For example, suppose the average age of employees is 37. Use the value 37 to fill in the missing value for the attribute "age".
(4) Replace the missing attribute values with the attribute mean from the same class: Suppose the mining task is to classify the "top-employee" from the typical employee and some values for the attribute age are missing. The missing value may be filled with the average age among the top-employees.
(5) Replace the missing attribute values with the most probable value: The probable value may be guessed by using regression technique, a Bayesian inference, or decision-tree induction. However, the technique is appropriate for sparse missing values. Difficulties arise if the tuple contains more than one missing attribute values.

Methodology: The experiments have been designed to test the performance of model induction from the data sets using different approaches to fill in the missing values. The induction algorithms are also selected from the different paradigms; that is, the Bayesian, the decision-tree induction, and the instance-based (or nearest neighbor) algorithms. We run our experiments on the Weka system [9] and observe the accuracy of classification on each data set that contains a particular replacement of missing values. The two data sets – Glass identification and Zoo database – are taken from the UCI repository [5]. Both data sets contain no missing value. The Glass-identification data set represents the numeric data, whereas the Zoo data set represents the nominal data. Characteristics of these two data sets are summarized as follows:

data	number of instances	numeric attributes	nominal attributes	# class
Glass identification	214	9	0	7
Zoo database	101	1	15	7

To simulate the data set with missing values, some attribute values are removed from the original two data sets. The glass identification data set contains about 21% of missing values distributed equally among the nine attributes. The zoo data set contains about 32% of missing values, distributed equally as well.

The missing values are replaced with the four approaches:

(1) Replace with a symbol "?" (the symbol normally appears in the UCI data for the unknown values).
(2) Replace with a constant value "99.9" (represents the upper bound value) for the glass identification data set, and with a constant symbol "missing" for the zoo data set.
(3) Replace with the mean value of the attribute that has missing value for the glass identification data set, and with a majority value of that attribute for the zoo data set.
(4) Simply ignore the missing values by removing all instances containing missing values.

We compare the performance of naïve Bayes classification method empirically with the decision-tree induction (J48 [14] – the re-implementation of C4.5 [9]) and the nearest neighbor method. The classification accuracy is estimated using stratified ten-fold cross-validation technique.

Results and Discussion: Table 1 and 2 summarize how the three classification methods perform on data with missing values, which are handle using different approaches. Correct classification is reported as the accuracy of each classification method.

Table 1. Classification accuracy on numerical data with missing values

Data set	Accuracy		
	naïve Bayes	decision-tree induction	nearest neighbor
Glass[1]	47.19632 %	65.4206 %	70.0935 %
Glass_M1[2]	50 %	72.8972 %	61.215 %
Glass_M2[3]	11.215 %	68.6916 %	61.6822 %
Glass_M3[4]	47.1963 %	71.9629 %	65.8879 %
Glass_M4[5]	50.2959 %	72.1893 %	68.6391 %

[1] original data set – no missing value

[2] generate and replace missing values with "?"

[3] generate and replace missing values with a global constant numeric value

[4] generate and replace missing values with attribute's mean value

[5] remove instances that contain missing values

Table 2. Classification accuracy on nominal data with missing values

Data set	Accuracy		
	naïve Bayes	decision-tree induction	nearest neighbor
Zoo[1]	93.0693 %	92.0792 %	98.0198 %
Zoo_M1[2]	94.0594 %	91.0891 %	99.0099 %
Zoo_M2[3]	91.0891 %	85.1485 %	91.0891 %
Zoo_M3[4]	92.0792 %	87.1287 %	91.0891 %
Zoo_M4[5]	89.5522 %	88.0597 %	94.0299 %

[1] original data set – no missing value

[2] generate and replace missing values with "?"

[3] generate and replace missing values with a global constant

[4] generate and replace missing values with attribute's majority value

[5] remove instances that contain missing values

When introducing the missing values and then replacing them with a global maximal value, the classification performance of naïve Bayesian classifier degrades dramatically (comparing to other missing-value handling approaches – as shown in Table 1). The replaced constant value may interfere the computation of mean value on the course of deriving probability. Among the four approaches on dealing with missing values, the strategy of removing all instances with missing values gives the best prediction accuracy.

Note that replacing the missing values with the symbol "?" (representing unknown values) works equally well on the Naïve Bayesian and decision tree induction classifiers. The introduction of "?" to the unknown values is a common practice appeared in the UCI data sets. Therefore, many classification algorithms implement a module to handle the case of reading the unknown value "?", whereas the other kinds of replacement (such as a global constant) are treated as another attribute value. This can mislead the classification process.

For the classification on nominal data (Table 2), replacing the missing values with the unknown-value symbol "?" produces the best accuracy on all three classifiers. Removing instances with missing values can be the second choice for the nearest neighbor and decision-tree induction classifiers.

Conclusion: We study the different approaches of dealing with missing values. When applying data mining to the real-world, learning from the incomplete data is an inevitable situation. Trying to complete missing values is one obvious solution. However, techniques to guess the missing values must not bias the classification method or introduce noise. We thus design the experiments to test the effect of different data replacement strategies on the accuracy of classifying numeric and nominal data sets.

The experimental results either suggest replacing the missing values with the unknown-value symbol "?", or removing the instances with missing values. For the naïve Bayesian

classifier, if the instances are so important that ignoring them may affect the result, replacing the missing values with a mean (for numeric data) or majority (for nominal data) value is another tempting strategy.

The understanding of classifier's behavior on different missing-value replacement strategies is useful for the decision of how to prepare data that best support the classifier.

References:

[1] A. Agrawal and R. Srikant, "Privacy preserving data mining", ACM SIGMOD, 2000.

[2] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, San Francisco, 2001.

[3] R.J.A. Little and D.B. Rubin, *Statistical Analysis with Missing Data*, John Wiley and Sons, 1987.

[4] W.Z. Liu, A.P. White, S.G. Thompson, and M.A. Bramer, "Techniques for dealing with missing values in classification", Second International Symposium on Intelligent Data Analysis, 1997.

[5] C.J. Merz and P.M. Murphy, UCI Repository of machine learning databases, 1996. [http://www.ics.uci.edu/ ~mlearn/MLRepository.html]

[6] J.R. Quinlan, "Unknown attribute values in induction", Proceedings of the Sixth International Workshop on Machine Learning, 1989, 164-168.

[7] A. Ragel and B. Cremilleux, "MVC: A preprocessing method to deal with missing values", Knowledge-Based Systems Journal, 1999, 285-291.

[8] A.P. White, "Probabilistic induction by dynamic path generation in virtual trees", M.A. Bramer (editor), *Research and Development in Expert Systems III*, Cambridge University Press, 1987, 35-46.

[9] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, San Francisco, 2000. [software accessible via the URL http://www.cs. waikato.ac.nz/ml/weka]

ภาคผนวก ข

รหัสต้นฉบับภาษาโปรล็อกของซอฟต์แวร์สร้างต้นไม้ตัดสินใจเชิงอุปนัยที่ทนต่อข้อมูลรบกวน

```
%% Program   Robust-Tree Induction
%%    by Nittaya Kerdprasop
%%    date 8 February 2009
%%    version 1.1
%%
%%  A decision tree induction program that can handle noisy data.
%%  The effect of noise is to be decreased by clustering, then data around means
%%      are selected for further classification by decision-tree induction.
%%
%%  Data format:
%%
%% attribute(size, [small, large]).
%% attribute(color, [red, blue]).
%% attribute(shape, [circle, triangle]).
%% attribute(class, [positive, negative]).
%
%% instance(1, class=positive, [size=small, color=red, shape=circle]).
%% instance(2, class=positive, [size=large, color=red, shape=circle]).
%% instance(3, class=negative, [size=small, color=red, shape=triangle]).
%% instance(4, class=negative, [size=large, color=blue, shape=circle]).
%
%% Node format:         node(NodeID, NodeLabel, ParentNode)
%%    where
%%            NodeID = 1, 2, 3, ..., root, leaf
%%            NodeLabel = "attribute=value / number_of_instances"
%%            ParentNode = 1, 2, 3, ..., root
%% e.g.
%%       node(1, shape=triangle, root).
%%       node(2, shape=circle, root).
%%       node(3, color=blue, 2).
%%       node(4, color=red, 2).
%%       node(leaf, [(class=negative/1)], 1).
%%       node(leaf, [(class=negative/1)], 3).
%%       node(leaf, [(class=positive/2)], 4).
%
%% Start program with the query
%%                ?- rt.              % for robust-tree induction
%%
%%    then specify robustness level:
%%     0 = no addition of robust technique; traditional ID3
%%     1 = extract data around centroids, or central points, as representatives for
%%               tree building
%%                (number of clusters = number of classes,
%%                  clustering technique is K-medoids)
%%
%% Input data with the following format:
%%        data-sample.
%%
%% To test model accuracy, call:        ?-test.
%%
%% Then input test data, e.g.,
%%                                data-sample-test.
% ================================
%% Program source code start here:
%%
%%  Note that  each module will be explained with the following format:
%%         an input argument is prefixed with a plus sign (+),
%%         the output argument is prefixed with a minus sign (-).
```

```
%
%% Main module:  rt
%% ==========
  rt :-
     writeln('Robust tree induction for data classification:'), nl,
     writeln(' There are two level of robustness'),
     writeln('  0 = simply ID3 style without noise handling function'),
     writeln('  1 = grouping data then select representatives to build tree'),   nl,
     write('     Please specify level of robustness (and end command with a period): '),
     read(L),
     write('    Training-data file name (e.g. data-sample.) ==> '),
      read(D),                    % get data file
      consult(D),                 % data is also a prolog program
      get_time(StartTime),
                % clear all nodes and node-ID counter in the DB
                % node and counter are two global values of this program
      retractall(node(_, _, _)),
      retractall(counter(_)),
                              % make list Attr of all attribute names except attribute class
     findall(A, (attribute(A, _), A \= class), Attr),
     rtree(L, Attr),
     get_time(FinishTime),
     Time is FinishTime-StartTime,
      nl, write('ROBUST-TREE:: robust level '), write(L), write(',  '),
      write('Model building time = '), write(Time), writeln('  sec.').

% ------------------------------------
%  start traditional tree-induction with ID3 algorithm

  rtree(0, Attr) :-  !,
                                        % make a list Ins = [1,2,...,n] of all instance ID
     findall(N, instance(N, _, _), Ins),
                                        % create decision tree, start with the root node
                                        % set MinInstance in leaf nodes = 1
                                        % then show model as decision tree once finish building phase
     induce_tree(root, Ins, Attr, 1),
     print_tree_model.

%-------------------------------------------
% start clustering before induce tree

  rtree(1, Attr) :-  !,
     attribute(class, ClassList),
     length(ClassList, K),
     findall(N, instance(N,_,_),Ins),
     clustering(Ins, K, Clusters, Means),
     select_DataSample(Clusters, K, Means, [], Sample),
     removed_Data(Sample, Ins, Removed),
     length(Removed, R),
     length(Ins, I),
     MinInstance is K-log((R+K)/I),        % MinInstance is a heuristic to prune tree
     induce_tree(root, Sample, Attr, MinInstance),
     print_tree_model,
    write('Min instances in each branch = '), writeln(MinInstance),
    nl, write('Initial Data = '), write(I), writeln(' instances'),
     write('Removed Data = '), writeln(Removed),
     write('    removed = '), write(R), writeln('   instances'), nl .
%% ---------------------------------------------------------
```

```
%% Module induce_tree(+ParentNode, +InstanceIDlist, +AttributeList, +MinInstance)
%% ===============
%%This module induces each node of decision tree.
%%There are four possible cases of tree induction based on current data characteristics.
%%
%
%% Case 1: empty data set, do nothing
%% ====
      induce_tree(ParentNode,[],_,_) :-
                instance(_,Class,_),
                assertz(node(leaf, [Class/0], ParentNode)), !.

%%  Case 2: Number of instances =< the specified MinInstance.
%%  ====    Thus, create a leaf node labeled with class distribution.

  induce_tree(ParentNode, InstanceIDlist, _, MinInstance) :-
        length(InstanceIDlist, NumInstances),
        NumInstances =< MinInstance,
                                % a constraint to satisfy case1
                                % count distinctive classes of current InstanceIDlist
                                % e.g. Dist = [class=negative/1, class=positive/2]
        classDistribution(InstanceIDlist, Dist),
                                        % insert a leaf node into the DB, don't try other cases
        assertz(node(leaf, Dist, ParentNode)), !.

%% Case 3: Number of instances > the specified MinInstance,
%% ====    but all instances are in the same class.
%%         Therefore, create a leaf node labelled with a class distribution.

 induce_tree(ParentNode, InstanceIDlist, _, _) :-
        classDistribution(InstanceIDlist, Dist),
        length(Dist, 1),                        % a constraint to assert the case of single class
        assertz(node(leaf, Dist, ParentNode)), !.

%%  Case 4:  Number of instances > the specified MinInstance.
%%  ====     Data contain a mixture of several classes, then grow tree.

 induce_tree(ParentNode, InstanceIDlist, AttrList, MinInstance) :-
     choose_attribute(InstanceIDlist, AttrList, A, Values, RestAttr),
                                        % choose the best attribute A from the AttrList
                                        % then build a subtree with A as a root node
      build_subtree(Values, A, InstanceIDlist, ParentNode, RestAttr, MinInstance), !.

%%   Case 5:  Cannot inducing tree due to inconsistent data
%%   ====     thus, stop growing tree and create a leaf node with
%%            heterogeneous classes, e.g., [(class=positive/2), (class=negative/1)]

 induce_tree(ParentNode, InstanceIDlist, _, _) :-
     node(ParentNode, TestAttribute, _),                % locate the error point
     write(' Inconsistent data: '),
     write(InstanceIDlist),
     write('    Cannot split at node: '),
     writeln(TestAttribute),
     classDistribution(InstanceIDlist, Dist),
     assertz(node(leaf, Dist, ParentNode)), !.          % insert a leaf node into the DB
%
%% ------------------------------------------------------------------------------
```

```
%% Module classDistribution(+InstanceIDlist, -ClassDistribution)
%% ==================
%% e.g.  InstanceIDlist = [1,2,3,4]
%%       ClassDistribution = [class=positive/2, class=negative=2]

  classDistribution(InstanceIDlist, ClassDistribution) :-
        setof(Class, I^AttrList^(member(I, InstanceIDlist),
                                  instance(I, Class, AttrList)), C),
                                        % make a set C of distinctive classes from InstanceIDlist
                                        %   e.g. C = [class=positive, class=negative]
        countClassMember(C, InstanceIDlist, ClassDistribution).
              % count number of instances in each class and return
              % a class distribution, e.g., [class=positive/2, class=negative/2]

   countClassMember([], _, []) :- !.
   countClassMember([C | L], I, [C/N | T]) :-
          findall(X, (member(X, I), instance(X, C, _)), W),
                                                % make a list W of instanceID in each class
                                                % e.g. W = [1,2] for class positive
     length(W, N),                          % output N = number of instances in class C
          countClassMember(L, I, T).        % count remaining classes
%% ------------------------------------------------------------

%% Module choose_attribute(+InstanceIDlist, +AttrList, -A, -Values, -RestAttr)
%% ==================
%%   e.g., InstanceIDlist = [1,2,3,4],  AttrList = [size, color, shape]
%%   A = shape,  Values = [triangle, circle], RestAttr = [size, color]

  choose_attribute(InstanceIDlist, AttrList, A, Values, RestAttr) :-
        length(InstanceIDlist, InsLen),
        compute_info(InstanceIDlist, InsLen, I), !,
                                % I  is expected number of information needed
                                %    to encode class of the given InstanceIDlist
        findall( A/Gain,    % find gain value of each attribute
                            %  with the following pattern of computation
                 ( member(A, AttrList),
                   attribute(A, Values),
                   split_instances(Values, InstanceIDlist, A, InsSubset),
                   subset_info(InsSubset, InsLen, R),
                   Gain is I - R ),
                                % extract attribute name A from AttrList one at a time,
                                % and get all possible values of attribute A
                                % then split instances based on the value of A
                                % compute info of data subset
                                % then compute gain value of A
                 AttributeGainList),
                                % output is a list of attribute/gain
                                % e.g. [size/0, color/0.311278, shape/0.311278]

        maximum(AttributeGainList, A/_),     %  find attribute A with the maximum gain
        attribute(A, Values),           % extract valuelist of this attribute
                                        %  and return the list of remaining attributes
        remainAttr(A, AttrList, RestAttr), !.
```

```
%% supporting module to compute info I of given instances
%   e.g., info([positive/2, negative/1]
%         = -2/3 log 2/3 - 1/3 log 1/3 = 0.918

compute_info(InstanceIDlist, InsLen, I) :-
        attribute(class, CList),              % get a list of class values
        sum_info(CList, InstanceIDlist, InsLen, I).

sum_info(_,_,0,0) :- !.          % base case: zero instance has info = 0
sum_info([], _, _, 0) :- !.      % base case: an empty class has info = 0

sum_info([C | Cs], InsIDlist, InsLen, Info) :-     % inductive case
                    findall(Ins, ( member(Ins, InsIDlist),
                     instance(Ins, class=C, _) ), ClassInstance),
                                % create a list to contain instances of each class
                    length(ClassInstance, N),
                                % then count the instance number
                    sum_info(Cs, InsIDlist, InsLen, I),
                                % do the same with other classes
                     InsLen > 0,
                     P is N / InsLen,
                          % if (N/InsLen) = 0, set Info = 0 to avoid calculate log(0)
                    ( P=0, Info = 0;
                     Info is I - (P) * (log( P ) / log(2) ) ).

    % supporting module split_instances(+Values,+InstanceIDlist, +A, - InsSubset)
    %          e.g.,Values=[large, small],  InstanceIDlist=[1,2,3,4], A=size
    %          the module will return InsSubset = [ [2,4], [1,3]]
split_instances([], _, _, []) :- !.
split_instances([V | Vs], InstanceIDlist, A, [InsIDlist | Rest]) :-
findall( InsID, ( member(InsID, InstanceIDlist),
                instance(InsID, _, L),
                member( A=V, L) ), InsIDlist),
                % split instances into subset InsIDlist based on the attribute value V
                % then, do the same for other attribute values Vs
split_instances(Vs, InstanceIDlist, A, Rest).

    %% supporting module subset_info
subset_info([], _, 0) :- !.
subset_info([InsGroup | OtherGroups], Len, Res) :-
                     length(InsGroup, LenInsGroup),
                     compute_info(InsGroup, LenInsGroup, I), !,
                     subset_info(OtherGroups, Len, R),
                     Len > 0,
                     Res is R + I * LenInsGroup / Len.

 %% supporting module maximum to search for attribute with maximum gain
  %%
maximum([A], A) :- !.                              % base case: list of one attribute
maximum([A/GainA | Rest], Attribute/Gain) :-  % recursively shorten the list
                    maximum(Rest, Att/G),
                    (GainA > G, Attribute/Gain = A/GainA ;
                    Attribute/Gain = Att/G ), !.

    %% supporting module remainAttr
    remainAttr(A, [A | T], T) :- !.
    remainAttr(A, [X | T], [X | Rest] ) :- remainAttr(A, T, Rest).
%% ---------------------------------------------------------------
```

```
%% Module build_subtree(+AttrValues, +A, +InstanceIDlist, +ParentNode, +RestAttr,+MinInstance)
%% ===============
%%   This module recursively create subtree start from the chosen attribute A.
%%   Branches of A are stored in a list AttrValues.
%% e.g., A= shape, AttrValues = [triangle, circle], InstanceIDlist = [1,2,3,4],
%%                      ParentNode = root
%%   the module build subtree extended from te root node with two branceses:
%%       shape = triangle   and  shape = circle
%%  The build_subtree process continues until the stopping criteria MinInstance has been reached.
%%

  build_subtree([], _, _, _, _, _) :- !.    % base case: there is no more attribute left  to create subtree

  build_subtree([ V | Vs], A, InsIDlist, ParentNode, RestAttr, MinInstance) :-
                          % create root of subtree
                          % get subset of instances with attribute A=V
       findall(InsID, (member(InsID, InsIDlist), instance(InsID, _, L),
                          member(A=V, L) ), Inslist),
       getNodeID(NodeID),
       assertz(node(NodeID, A=V, ParentNode)),

            % recursively build left subtree
       induce_tree(NodeID, Inslist, RestAttr, MinInstance), !,

            % build  right subtree based on Vs
       build_subtree(Vs, A, InsIDlist, ParentNode, RestAttr, MinInstance).

        %% supporting module getNodeID(-NodeID)
        %%
    getNodeID(M) :-
                    retract(counter(N)),       % check current counter N
                    M is N + 1,                % increment N by 1
                    assert(counter(M)), !.     % then record the new counter

    getNodeID(1) :-
                    assert(counter(1)). % if counter does not exist, then create one

%% ----------------------------------------------------------------

%% Module print_tree_model:
%% =============
 print_tree_model :-
          print_tree_model(root, 0),     % start from root node at position zero
          nl, nl,
          write('Size of tree: '),
          retract(counter(N)),
          write(N),
          write(' internal nodes and '),
          findall(Node, node(leaf, _,Node), NL),
          length(NL, M),
          write(M),
          writeln(' leaf nodes.').

 print_tree_model(ParentNode, _) :-        % the case for printing leaf node
      node(leaf, Class, ParentNode), !,
      write(' => '), write(Class).
```

```
  print_tree_model(ParentNode, Position) :-
      findall(Son, node(Son, _, ParentNode), L),
      Position1 is Position+2,
      childList(L, Position1), !.

  childList([], _) :- !.
  childList([N|Child], Pos) :-
      node(N, NodeLabel, _),
      nl, tab(Pos), write(NodeLabel),
      print_tree_model(N, Pos),
      childList(Child, Pos).

%======================END BUILD ROBUST-TREE==========
%==== Test TREE =========
%
test :-
     write('Test-data file name (e.g. data-sample-test.) ==> '),
     read(D),
     consult(D),
     get_time(Start),
                    % get all instance ID of test data
     findall(TestIns, instance(TestIns, _, _), TestInsList),
     length(TestInsList, NumTestCase),
                    % send all test cases to test_accuracy module
                    %      with initial correct case = 0
     test_accuracy(TestInsList, 0, Totalcorrect), !,
     Accuracy is Totalcorrect / NumTestCase,

     nl,write('Predicting correctly: '), write(Totalcorrect),
     write(' from '), write(NumTestCase), write(' cases ==> '),
     write('Accuracy = '), writeln(Accuracy),
     get_time(Finish),
     Time is Finish- Start,
      nl, tab(5), write('Model Test Time = '), write(Time), writeln('  sec.').

 %% Module test_accuracy
 %%  get all test cases, and
 %%    start evaluating correctness of prediction one case at a time,
 %%    stop when the lest of test cases is empty,
 %%    then report the total number of cases predicted correctly

test_accuracy([], C, C) :- !.

test_accuracy([Case| Rest], Correct, NextCorrect) :-
       instance(Case, Trueclass, AttList), % get current test case
                                    % search tree for predicted class start from root node
       search_decision(root, AttList, Prediction),
                                    % compare Trueclass and PredictedClass
                                    % and count correct prediction
       evaluate(Case, Trueclass, Prediction, Correct, NewCorrect),
                                     % recursively do the same for other cases
       test_accuracy(Rest, NewCorrect, NextCorrect).


search_decision(StartNode, _, Prediction) :-
        node(leaf, Prediction, StartNode), !.
                % return Prediction once leaf node has been found
```

```
search_decision(StartNode, AttList, Prediction) :-
        node(NextNode, TestAtt, StartNode),
        member(TestAtt, AttList), !,
        search_decision(NextNode, AttList, Prediction).

evaluate(_, Trueclass, Prediction, Correct, NewCorrect) :-
                        % Prediction might be a mixture such as
                        %       [(class=positive)/2, (class=negative)/1]
                        % thus, PredictedClass should be the majority class
        maximum(Prediction, PredictedClass/_),
       (Trueclass == PredictedClass,  NewCorrect is Correct +1;
        NewCorrect = Correct).

%% ======== END Test-Tree======================
%%
%% Module Clustering
%% ==============
clustering(Ins, K, Clusters, Means) :-
    length(Ins, N),
    initialized_means(N, K, [], MeanPoints),
                        % e.g. MeanPoints = [2/1, 3/2]
                        % get attributes of initial MeansPoints
    findall(MeanAttr/Cluster, (member(P/Cluster, MeanPoints),
                                instance(P, _, MeanAttr)),
            MeansAttrList),
                        % e.g. [(size=small,color=red,shape=circle)/1,
                        %       (size=large,color=blue,shape=circle)/2]
    assign_clusters(MeansAttrList, Ins, K, Clusters,Means).
%
%
assign_clusters(MeansAttr, Ins, K, Clusters,Means) :-
    find_clusters(MeansAttr, Ins, [], InsClusterList),
    find_means(InsClusterList, K, [], TempMeans),
    getRepresentatives(TempMeans,Ins,[],RepList),
    getMeans(RepList,[],NewMeans),
    find_clusters(NewMeans, Ins, [], NewInsClusterList),
    entropy(InsClusterList, K, PreEntropy),
    entropy(NewInsClusterList, K, PostEntropy),
    average(PreEntropy, PreEn),
    average(PostEntropy, PostEn),
    (PostEn >= PreEn, Clusters = InsClusterList, Means = NewMeans, ! ;
       assign_clusters(NewMeans, Ins, K, Clusters,Means),!).

%%
initialized_means(_, 0, Means, Means) :- !.
initialized_means(N, K, Means, NewMeans) :-
     MeanIns is random(N-1)+1,
     NewK is K-1,
     initialized_means(N, NewK, [MeanIns/K|Means], NewMeans).
%%
find_clusters(_, [], List, List) :- !.
find_clusters(MeanAttrList, [Ins|Rest], CurrentList, NewList) :-
      findall(Cluster/Score, (instance(Ins,_, InsAtt),
                              member(MAtt/Cluster, MeanAttrList),
                              similarity(MAtt, InsAtt, 0, Score) ),
              ClusterScoreList),
      maximum(ClusterScoreList, Cluster/_),
      find_clusters(MeanAttrList, Rest, [Ins/Cluster|CurrentList], NewList).
```

```
similarity([],[],S,S) :- !.
similarity([A | RestA1], [A | RestA2], Score, NewS) :-
                NewScore is Score + 1, !,
                similarity(RestA1, RestA2, NewScore, NewS).
similarity([A1 | RestA1], [A2 | RestA2], Score, NewS) :-
                A1 \= A2,
                similarity(RestA1, RestA2, Score, NewS).
%%
minimum([ClusterScore], ClusterScore) :- !.
minimum([C/S | Rest], Cluster/Score) :-
        minimum(Rest, Clus/Sc),
        ( Sc > S, Cluster/Score = C/S ;
                Cluster/Score = Clus/Sc), !.
%%
find_means(_, 0, List, List) :- !.
find_means(InsClusterList, K, CurrentList, NewList) :-
        findall(Ins, member(Ins/K, InsClusterList), InsList),
        findall(Name=Vlist, (attribute(Name,Values),
                                Name \= class,
                                findall(V/0, member(V,Values), Vlist)),
                AttValueList),
        common_attributes(InsList, AttValueList, AttrList),
        NewK is K - 1,
        find_means(InsClusterList, NewK, [AttrList/K | CurrentList], NewList).

%%
common_attributes([], AttValueList, AttList) :- !,
        findall(A=V, (member(A=VList, AttValueList),
                        maximum(VList, V/_) ),  AttList).

common_attributes([Ins | Rest], AttValueList, AttList) :-
        instance(Ins,_, AttValue),
        count_value(AttValue, AttValueList, NewAttValueList),
        common_attributes(Rest, NewAttValueList, AttList).

%%
count_value([], AVList, AVList) :- !.
count_value([A=V | Rest], AttValueList, NewAttValueList) :-
        member(A=VList, AttValueList),
        delete(AttValueList, A=VList, TempAttValueList),
        member(V/Count, VList),
        delete(VList, V/Count, TempVList),
        NewCount is Count + 1,
        append([V/NewCount], TempVList, NewVList),
        append([A=NewVList], TempAttValueList, NewAVList),
        count_value(Rest, NewAVList, NewAttValueList).

%%
entropy(_, 0, []) :- !.
entropy(InsCluster, K, Entropy) :-      K>0,
        findall(Ins, member(Ins/K, InsCluster), InsList),
        length(InsList, InsLen),
        (InsLen >0,
             compute_info(InsList, InsLen, Info),
             Entropy = [K/Info | RestEntropy];
        Entropy = [K/1 | RestEntropy]),
        NewK is K-1,
        entropy(InsCluster, NewK, RestEntropy).
```

```
sum_list([], 0) :- !.
sum_list([H|T], Value) :-
        sum_list(T, NewValue),
        Value is H + NewValue.
%%
getRepresentatives([],_, List, List) :- !.
getRepresentatives([Mean/Cluster | Rest], InsList, Current, NewList) :-
        findall(Ins/Score, (member(Ins, InsList),
                            instance(Ins,_,InsAtt),
                            similarity(InsAtt, Mean, 0, Score)),  InsScoreList),
        maximum(InsScoreList, Instance/_),
        delete(InsList, Instance, NewIns),
        getRepresentatives(Rest,NewIns, [Instance/Cluster | Current], NewList).
%%
getMeans([], List, List) :- !.
getMeans([Ins/Cluster | Rest], Current, NewMeans) :-
        instance(Ins,_, InsAtt),
        getMeans(Rest, [InsAtt/Cluster | Current], NewMeans).

removed_Data(DataSample, InstList ,RemovedData ) :-
        findall( D, (member(D, InstList),
                    not(member(D, DataSample))),
                RemovedData).
%%
select_DataSample(_, 0, _, DataSample, DataSample) :- !.
select_DataSample(Clusters, K, Means, TempData, DataSample) :-
        findall(Ins, member(Ins/K, Clusters), InsKList),
        length(InsKList, Len), Len > 0,
        findall(I/Score, (member(I, InsKList),
                          instance(I, _, InsAtt),
                          member(MAtt/K, Means),
                          similarity(InsAtt, MAtt, 0,Score)),
                IScoreList),
        average(IScoreList, Average),
        variance(IScoreList, Average,Variance),
        Threshold is (2 * Variance),          % a heuristic for stopping tree induction
        findall(Inst, (member(Inst/Sc, IScoreList),
                       Sc >= Threshold), InstList),
        append(InstList, TempData, NewData),
        NewK is K-1,
        select_DataSample(Clusters, NewK, Means, NewData, DataSample).
%%
average(ValueList, E) :-
        findall(S, member(_/S, ValueList), SList),
        sum_list(SList, SValue),
        length(SList, Len),
        (Len=0, E = 0; E is SValue / Len).
%%
variance(ValueList, Avg, Var) :-
        findall(Diff, (member(_/S, ValueList),
                       Diff is abs(S-Avg)),
                DiffList),
        sum_list(DiffList, DValue),
        length(DiffList, DLen),
        D is DLen-1,
        (D=0, Var = 0; Var is DValue / D).

% ===== End Clustering ============
```

ประวัติผู้วิจัย

รองศาสตราจารย์ ดร.นิตยา เกิดประสพ สำเร็จการศึกษาในระดับปริญญาเอกสาขา Computer Science จาก Nova Southeastern University เมือง Fort Lauderdale รัฐฟลอริดา สหรัฐอเมริกา เมื่อปีพุทธศักราช 2542 (ค.ศ. 1999) ด้วยทุนการศึกษาของกระทรวงวิทยาศาสตร์ฯ โดยทำวิทยานิพนธ์ระดับปริญญาเอกในหัวข้อเรื่อง "The application of inductive logic programming to support semantic query optimization" หลังสำเร็จการศึกษาได้ปฏิบัติราชการในตำแหน่งอาจารย์ ประจำสาขาคอมพิวเตอร์ ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ต่อมาในปีพุทธศักราช 2543 ได้มาปฏิบัติงานในตำแหน่งอาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จนถึงปัจจุบัน งานวิจัยที่ทำในขณะนี้คือ การพัฒนาระบบเหมืองข้อมูลประสิทธิภาพสูงที่สามารถทนต่อข้อมูลรบกวน และการเพิ่มความสามารถในการจัดการความรู้ของระบบเหมืองข้อมูล

รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ สำเร็จการศึกษาในระดับปริญญาเอกสาขา Computer Science จาก Nova Southeastern University เมือง Fort Lauderdale รัฐฟลอริดา สหรัฐอเมริกา เมื่อปีพุทธศักราช 2542 (ค.ศ. 1999) ด้วยทุนการศึกษาของทบวงมหาวิทยาลัย (หรือสำนักงานคณะกรรมการอุดมศึกษาในปัจจุบัน) โดยทำวิทยานิพนธ์ระดับปริญญาเอกในหัวข้อเรื่อง "Active database rule set reduction by knowledge discovery" หลังสำเร็จการศึกษาได้ปฏิบัติงานในตำแหน่งอาจารย์ ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ปัจจุบันดำเนินการวิจัยเกี่ยวกับการพัฒนาระบบเหมืองข้อมูลประสิทธิภาพสูงที่สามารถทนต่อข้อมูลรบกวน และการวิจัยพื้นฐานเกี่ยวกับเทคนิคการจัดกลุ่มข้อมูล และการวิเคราะห์ข้อมูลโดยวิธีอัตโนมัติ โดยมีผลงานวิจัยตีพิมพ์ในวารสารวิชาการและเอกสารการประชุมวิชาการ จำนวนมากกว่า 30 เรื่อง ในสาขาฐานข้อมูลแอคทีฟ ฐานข้อมูลนิรนัย การทำเหมืองข้อมูลและการค้นหาความรู้