บทที่ 4
ขั้นตอนวิธีการเรียนรู้กฎความสัมพันธ์และรูปแบบลำดับ

4.1 ขั้นตอนวิธีการเรียนรู้กฎความสัมพันธ์[4]

กฎความสัมพันธ์คือกฎที่อธิบายถึงความสัมพันธ์ของเหตุการณ์ต่างๆที่เกิดขึ้น ซึ่งสามารถใช้เป็นกฎความน่าจะเป็นได้ โดยที่ความสัมพันธ์ของเหตุการณ์ หมายถึง เหตุการณ์ที่เกิดขึ้นพร้อมกันบ่อยๆ

ตัวอย่างที่ดีตัวอย่างหนึ่งคือฐานข้อมูลทรานแซคชันการขายสินค้าซึ่งเป็นองค์ประกอบสำคัญของโครงสร้างทางการตลาดของบริษัทต่างๆ การหาความรู้ที่เกี่ยวกับสิ่งของที่ถูกซื้อด้วยกันบ่อยๆจะช่วยให้บริษัทเหล่านี้สามารถพัฒนากลยุทธ์การตลาดที่มีประสิทธิภาพยิ่งขึ้น

นิยาม 4.1 *$I=\{i_1,i_2,\ldots,i_m\}$ เป็นเซตของตัวอักษรซึ่งเรียกว่าไอเท็มเซต (itemset) D คือ เซตของทรานแซคชันซึ่งแต่ละทรานแซคชัน T คือ เซตของไอเท็มซึ่ง $T\subseteq I$ X คือ เซตของไอเท็มใน I และทรานแซคชัน T มี X เป็นส่วนประกอบถ้า $X\subseteq T$ กฎความสัมพันธ์เป็นกฎที่อยู่ในรูปของ $X\Rightarrow^{s,c}Y$ โดยที่ $X\subset I, Y\subset I$ และ $X\cap Y=\varnothing$*

- *กฎ $X\Rightarrow Y$ มีค่าซัพพอร์ต (support) เท่ากับ s ก็ต่อเมื่อ อัตราส่วนระหว่างจำนวนทรานแซคชันที่มี $X\cup Y$ เป็นส่วนประกอบ กับ จำนวนทรานแซคชันทั้งหมดมีค่าเท่ากับ s*
- *กฎ $X\Rightarrow Y$ มีค่าความเชื่อมั่น (confidence) เท่ากับ c ก็ต่อเมื่อ อัตราส่วนระหว่างจำนวนทรานแซคชันที่มี $X\cup Y$ เป็นส่วนประกอบ กับ จำนวนทรานแซคชันที่มี X ประกอบอยู่มีค่าเท่ากับ c*

ตัวอย่าง

/company/products $\rightarrow^{0.4,0.8}$ /company/products/file2.html

จากกฎสามารถอธิบายได้ว่าผู้ชม 80 เปอร์เซ็นต์ของทรานแซคชันทั้งหมดที่เยี่ยมชม /company/products จะเยี่ยมชม /company/products/file2.html ด้วย โดยที่จำนวนของทรานแซคชันที่เยี่ยมชมทั้งสองเพจคิดเป็น 40 เปอร์เซ็นต์ของทรานแซคชันทั้งหมด

เมื่อกำหนดเซตของทรานแซคชัน D งานของการขุดค้นกฎความสัมพันธ์คือการหากฎความสัมพันธ์ทั้งหมดซึ่งมีค่าซัพพอร์ตไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำ (minimum support) และมีค่าความเชื่อมั่นไม่ต่ำกว่าค่าความเชื่อมั่นขั้นต่ำ (minimum confident) โดยที่ค่าซัพพอร์ตขั้นต่ำและค่าความเชื่อมั่นขั้นต่ำเป็นค่าที่ผู้ใช้กำหนดขึ้น

งานของการค้นพบกฎความสัมพันธ์สามารถแบ่งออกได้เป็น 2 ส่วน คือ

1 หาไอเท็มเซตทั้งหมดซึ่งมีจำนวนของทรานแซคชันที่มีไอเท็มเซตนั้นเป็นส่วนประกอบมากกว่าค่าซัพพอร์ตขั้นต่ำและเราเรียกไอเท็มเซตที่มีค่าซัพพอร์ตเคานท์ (ค่าซัพพอร์ตเคานท์ของไอเท็มเซตคือจำนวนของทรานแซคชันซึ่งมีไอเท็มเซตนั้นเป็นส่วนประกอบ) มากกว่าค่าซัพพอร์ตขั้นต่ำนี้ว่าไอเท็มเซตขนาดใหญ่ (large itemset) แต่ถ้ามีค่าซัพพอร์ตเคานท์น้อยกว่าค่าซัพพอร์ตขั้นต่ำจะเรียกว่าไอเท็มเซตขนาดเล็ก (small itemset) ซึ่งจะใช้อัลกอริทึม APRIORI ทำงานในส่วนนี้

2 สร้างกฎความสัมพันธ์ทั้งหมดที่มีค่าซัพพอร์ตไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำและมีค่าความเชื่อมั่นไม่น้อยกว่าค่าความเชื่อมั่นขั้นต่ำจากไอเท็มเซตขนาดใหญ่ทั้งหมดโดยใช้อัลกอริทึมดังนี้

- สำหรับแต่ละไอเท็มเซตขนาดใหญ่ l หาเซตย่อยทั้งหมดของ l ที่ไม่ใช่เซตว่าง
- สำหรับแต่ละเซตย่อย a ของ l สร้างกฎ a$\Longrightarrow$ (l-a) ได้ก็ต่อเมื่ออัตราส่วนของค่าซัพพอร์ตเคานท์ของ l ต่อค่าซัพพอร์ตเคานท์ของ a มีค่าไม่น้อยกว่าค่าความเชื่อมั่นขั้นต่ำ

4.1.1 อัลกอริทึม APRIORI

APRIORI เป็นอัลกอริทึมหนึ่งที่ใช้ในการเรียนรู้กฎความสัมพันธ์ซึ่งจะหาไอเท็มเซตทั้งหมดซึ่งมีจำนวนทรานแซคชันที่มีไอเท็มเซตเป็นส่วนประกอบมากกว่าค่าซัพพอร์ตขั้นต่ำ และเรียกไอเท็มเซตที่มีค่าซัพพอร์ตเคานท์ (ค่าซัพพอร์ตเคานท์ของไอเท็มเซตคือจำนวนทรานแซคชันที่มีไอเท็มเซตนั้นเป็นส่วนประกอบ) มากกว่าค่าซัพพอร์ตขั้นต่ำว่าไอเท็มเซตขนาดใหญ่ รูปที่ 4 แสดงอัลกอริทึม APRIORI ซึ่งมีสัญลักษณ์ต่างๆดังนี้

k-itemset คือ ไอเท็มเซตที่มีสมาชิก k ตัว

L_k คือ เซตของไอเท็มเซตขนาดใหญ่ (large itemset) ที่มีสมาชิก k ตัว (มีค่าซัพพอร์ตเคานท์มากกว่าค่าซัพพอร์ตขั้นต่ำ)

C_k คือ เซตของไอเท็มเซตคู่แข่ง (candidate itemset) ที่มีสมาชิก k ตัว (คือ ไอเท็มเซตขนาดใหญ่ที่สามารถเป็นไปได้)

ในการหาไอเท็มเซตขนาดใหญ่นั้นจะนำข้อมูลมาพิจารณาหลายๆรอบ

- ในรอบแรก จะทำการนับค่าซัพพอร์ตเคานท์ของแต่ละไอเท็มและหาว่าไอเท็มใดเป็นไอเท็มเซตขนาดใหญ่บ้างโดยทำการเปรียบเทียบกับค่าซัพพอร์ตขั้นต่ำ
- ในรอบต่อๆมาจะเริ่มต้นด้วยการนำเอาเซตของไอเท็มเซตขนาดใหญ่ที่ได้จากรอบก่อนหน้ามาเป็นข้อมูลตั้งต้นเพื่อนำไปสร้างเป็นไอเท็มเซตคู่แข่งซึ่งมีสมาชิกเพิ่มขึ้นหนึ่งตัวที่สามารถเป็นไปได้อันใหม่
- นับค่าซัพพอร์ตเคานท์ของไอเท็มเซตคู่แข่งเหล่านี้

- ในตอนท้ายของรอบจะพิจารณาว่ามีไอเท็มเซตคู่แข่งตัวใดบ้างที่เป็นไอเท็มเซตขนาดใหญ่ซึ่งไอเท็มเซตขนาดใหญ่เหล่านี้จะถูกนำไปเป็นข้อมูลตั้งต้นของรอบถัดไป
- กระบวนการนี้จะดำเนินไปเรื่อยๆจนกว่าจะไม่มีไอเท็มเซตขนาดใหญ่อันใหม่เกิดขึ้น

อัลกอริทึม APRIORI

L_1={large 1-itemsets};

For ($k=2; L_{k-1} \neq \varnothing; k++$) do begin

  C_k = apriori-gen(L_{k-1}); // new candidates

  For all transactions $t \in D$ do begin

    C_t = subset(C_k,t); // Candidates contained in t

    For all candidates $c \in C_t$ do

      c.count++;

  end

  $L_k = \{c \in C_k \mid c.count \geq minsup\}$

End

Answer =$U_k L_k$

รูปที่ 4 อัลกอริทึม APRIORI

4.1.2 วิธีการสร้างไอเท็มเซตคู่แข่ง

ฟังก์ชัน apriori-gen จะสร้างไอเท็มเซตคู่แข่งที่มีสมาชิก k ตัวโดยใช้ค่า L_{k-1} (L_{k-1} คือ เซตของไอเท็มเซตขนาดใหญ่ที่มีสมาชิก k-1 ตัว) เป็นอาร์กิวเมนต์และคืนค่าไอเท็มเซตคู่แข่งที่มีสมาชิก k ตัวที่เป็นไปได้ทั้งหมด ฟังก์ชันทำงานดังแสดงในรูปที่ 5 ซึ่งอธิบายได้ดังนี้

ขั้นตอนแรกคือขั้นตอนการประสาน (join step) ซึ่งจะประสาน L_{k-1} กับ L_{k-1}ซึ่งมีสมาชิก k-2 ตัวแรกเหมือนกันเข้าด้วยกันเป็นซุปเปอร์เซตที่มีสมาชิก k ตัว ซึ่งโดยปกติแล้วเมื่อไอเท็มเซตมีขนาดใหญ่ขึ้นก็จะมีค่าซัพพอร์ตเคานท์น้อยลง

Insert into C_k

Select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$

From L_{k-1} p , L_{k-1} q

Where $p.item_1=q.item_1, \ldots p.item_{k-2}=q.item_{k-2}, p.item_{k-1}<q.item_{k-1}$

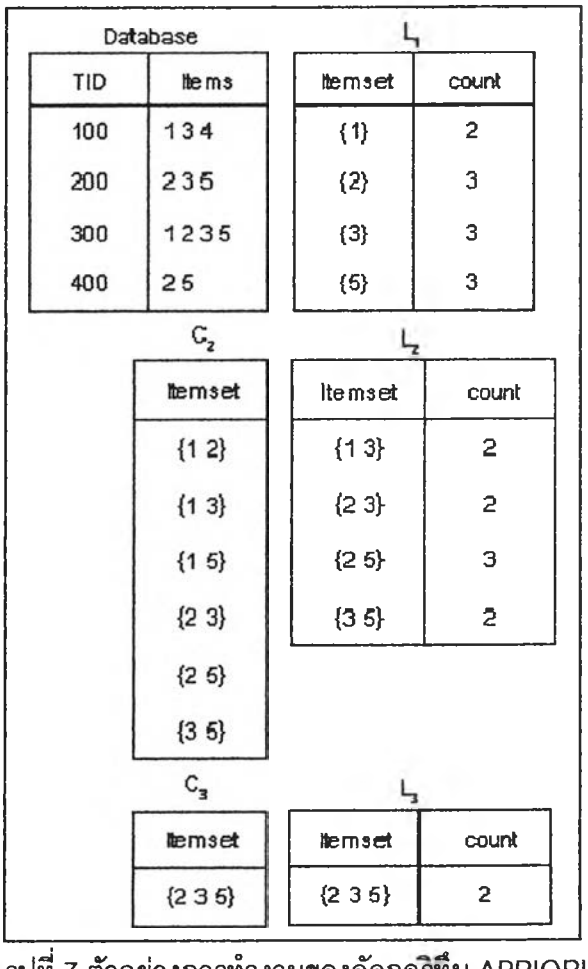
รูปที่ 5 ฟังก์ชัน apriori-gen (join step)

ขั้นตอนต่อมาคือขั้นตอนการตัดเล็ม (prune step) ซึ่งจะลบไอเท็มเซตทั้งหมด ($c \in C_k$) ซึ่งมีเซตย่อยที่มีสมาชิก k-1 ตัวบางเซตย่อยของ c ซึ่งไม่อยู่ใน L_{k-1} เพราะไอเท็มเซตที่มีสมาชิก k ตัวจะเป็นไอเท็มเซตขนาดใหญ่ถ้าเซตย่อยที่มีสมาชิก k-1 ตัวทั้งหมดของมันเป็นไอเท็มเซตขนาดใหญ่ ดังแสดงในรูปที่ 6

```
for all itemsets c ∈ C_k do
    for all (k-1)-subsets s of c do
        if (s ∉ L_{k-1}) then
            delete c from C_k;
```

รูปที่ 6 ฟังก์ชัน apriori-gen (prune step)

ตัวอย่าง



รูปที่ 7 ตัวอย่างการทำงานของอัลกอริทึม APRIORI

รูปที่ 7 แสดงตัวอย่างการทำงานของอัลกอริทึม APRIORI ซึ่งสามารถอธิบายได้ดังนี้ ขั้นแรกจะนับค่าซัพพอร์ตเคานท์ของแต่ละไอเท็มจากทรานแซคชันทั้งหมดซึ่งแสดงอยู่ในตารางฐานข้อมูลแล้วกำหนดว่ามีไอเท็มใดบ้างที่เป็นไอเท็มเซตขนาดใหญ่โดยพิจารณาจากค่าซัพพอร์ตเคานท์ซึ่งต้องไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนด คือ 2 ซึ่งทำให้ได้ไอเท็มเซตขนาดใหญ่ที่มีสมาชิก 1 ตัว 4 ไอเท็มเซต คือ {1} {2} {3} และ {5} ดังแสดงในตาราง L_1 ในรอบต่อมาจะนำไอเท็มขนาดใหญ่จากใน L_1 มาสร้างเป็นไอเท็มเซตคู่แข่งที่มีสมาชิก 2 ตัว 6 ไอเท็มเซต คือ {1 2} {1 3} {1 5} {2 3} {2 5} และ {3 5} ดังแสดงในตาราง C_2 และต่อมาจะกำหนดไอเท็มที่เป็นไอเท็มเซตขนาดใหญ่โดยพิจารณาจากค่าซัพพอร์ตเคานท์ซึ่งต้องไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนดซึ่งทำให้ได้ไอเท็มเซตขนาดใหญ่ที่มีสมาชิก 2 ตัว ดังแสดงในตาราง L_2 ในรอบต่อมาจะนำไอเท็มขนาดใหญ่จากใน L_2 มา สร้างเป็นไอเท็มเซตคู่แข่งที่มีสมาชิก 3 ตัว ดังแสดงในตาราง C_3 และต่อมาจะกำหนดไอเท็มที่เป็นไอเท็มเซตขนาดใหญ่โดยพิจารณาจากค่าซัพพอร์ตเคานท์ซึ่งต้องไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนดซึ่งทำให้ได้ไอเท็มเซตขนาดใหญ่ที่มีสมาชิก 3 ตัว 1 ไอเท็มเซต คือ {2 3 5} ดังแสดงในตาราง L_3 ในรอบสุดท้ายจะนำไอเท็มขนาดใหญ่จากใน L_3 มาสร้างเป็นไอเท็มเซตคู่แข่งที่มีสมาชิก 4 ตัว แต่เนื่องจากไม่สามารถสร้างไอเท็มเซตคู่แข่งที่มีสมาชิก 4 ตัวได้และทำให้ไม่สามารถสร้างไอเท็มเซตขนาดใหญ่ที่มีสมาชิก 4 ตัวได้ด้วย ดังนั้นอัลกอริทึม APRIORI จึงหยุดการทำงาน

4.1.3 วิธีการสร้างกฎ

ในการสร้างกฎจะทำการหาเซตย่อยทั้งหมดที่ไม่ใช่เซตว่างของไอเท็มเซตขนาดใหญ่ l ทุกตัว ดังนั้นสำหรับเซตย่อย a ทุกๆตัวของ l จะสร้างกฎที่อยู่ในรูป $a \Rightarrow (l-a)$ ถ้าอัตราส่วนของค่าซัพพอร์ตเคานท์ของ l ต่อค่าซัพพอร์ตเคานท์ของ a มีค่าไม่ต่ำกว่าค่าความเชื่อมั่นขั้นต่ำ

เราสามารถปรับปรุงกระบวนการข้างบนได้โดยสร้างเซตย่อยของไอเท็มเซตขนาดใหญ่ในแบบรีเคอร์ซีพโดยไปในทางลึกก่อน ตัวอย่างเช่น เมื่อกำหนดไอเท็มเซต ABCD มาให้ ขั้นแรกพิจารณาเซตย่อย ABC และตัวถัดไปคือ AB ฯลฯ และต่อมาถ้าเซตย่อย a ของไอเท็มเซตขนาดใหญ่ l ไม่สามารถสร้างกฎได้ก็ไม่จำเป็นต้องพิจารณาเซตย่อยอื่นของ a เพื่อมาใช้ในการสร้างกฎที่ใช้ l อีก

ตัวอย่างเช่น ถ้า $ABC \Rightarrow D$ มีค่าความเชื่อมั่นไม่เพียงพอแล้วก็ไม่จำเป็นต้องเช็ค $AB \Rightarrow CD$ อีก ซึ่งวิธีการนี้จะไม่ทำให้พลาดกฎใดๆเพราะว่าค่าซัพพอร์ตเคานท์ของเซตย่อย a^- ของ a จะมีค่ามากกว่า a ซึ่งทำให้ค่าความเชื่อมั่นของกฎ $a^- \Rightarrow (l- a^-)$ จะมีค่าไม่มากไปกว่าค่าความเชื่อมั่นของ $a \Rightarrow (l-a)$ นั่นหมายความว่าถ้า a ไม่สามารถสร้างกฎจาก l ได้แล้วก็จะไม่สามารถสร้างกฎจาก a^- ได้เช่นกัน ซึ่งแนวความคิดนี้สามารถสร้างเป็นอัลกอริทึมได้ดังแสดงในรูปที่ 8 ด้านล่างนี้

```
// อัลกอริทึมเริ่มต้น
for all large itemsets l_k , k ≥ 2 do
        call genrules(l_k, l_k);
// ฟังก์ชัน genrules สร้างกฎ b⇒(l_k- a~) โดยที่ a~⊂a_m
procedure genrules (l_k: large k-itemset, a_m: large m-itemset)
        A={(m-1)-itemsets : a_m-1 | a_m-1⊂a_m};
        For all a_m-1∈A do begin
           conf = support (l_k) / support(a_m-1);
           if (conf≥minconf) then begin
              output the rule a_m-1⇒(l_k-a_m-1) , with confidence = conf and support = support(l_k);
              if (m-1>1) then
                 call genrules(l_k ,a_m-1); // to generdate rules with subsets of a_m-1 as the antecedents
           end
        end
```

รูปที่ 8 ฟังก์ชัน genrules

จากฟังก์ชัน genrules ในรูปที่ 8 สามารถอธิบายขั้นตอนการทำงานได้ดังนี้

- ขั้นแรกฟังก์ชัน genrules จะนำไอเท็มเซตขนาดใหญ่ที่มีจำนวนสมาชิก m ตัวซึ่งเป็นอาร์กิวเมนต์ a_m มาหาเซตย่อยซึ่งมีจำนวนสมาชิก m-1 ตัวที่เป็นไปได้ทั้งหมดเก็บเอาไว้ใน A
- ขั้นต่อมาจะนำสมาชิกแต่ละตัว a_{m-1} ใน A มาหาค่าความเชื่อมั่น conf โดยคำนวณได้จากค่าซัพพอร์ตของ l_k หารด้วยค่าซัพพอร์ตของ a_{m-1} จากนั้นจึงนำค่า conf ที่ได้มาพิจารณาว่ามีค่าต่ำกว่าค่าความเชื่อมั่นขั้นต่ำที่กำหนดหรือไม่ ถ้าหากมีค่าไม่ต่ำกว่าค่าความเชื่อมั่นขั้นต่ำจะสร้างกฎ $a_{m-1} \Rightarrow (l_k - a_{m-1})$ โดยที่ค่าซัพพอร์ตของกฎจะมีเท่ากับค่าซัพพอร์ตของ l_k และมีค่าความเชื่อมั่นเท่ากับ conf หลังจากนั้นจะตรวจสอบว่าค่า m-1 คือ จำนวนสมาชิกของ a_{m-1}ยังมีค่ามากกว่า 1 หรือไม่ ถ้ายังมีค่ามากกว่า 1 จะเรียกฟังก์ชัน genrules แบบรีเคอร์ซีฟโดยส่งพารามิเตอร์คือ l_k และ a_{m-1} เพื่อนำไปประมวลผลต่อ

4.2 ขั้นตอนวิธีการเรียนรู้รูปแบบลำดับ[5]

เมื่อกำหนดฐานข้อมูลทรานแซคชันการซื้อสินค้าของลูกค้า D ซึ่งในแต่ละทรานแซคชันประกอบด้วยฟิลด์ต่างๆดังนี้ รหัสประจำตัวลูกค้า เวลาที่เกิดทรานแซคชัน และสิ่งของที่ซื้อ โดยที่ไม่มีลูกค้าคนใดมีทรานแซคชันมากกว่าหนึ่งทรานแซคชันภายในเวลาเดียวกัน

นิยาม 4.2 *ลำดับ (sequence) คือ ลิสต์ของไอเท็มเซต*

สมมุติให้ไอเท็มเซตอยู่ในรูปของเซตของตัวเลข แล้วไอเท็มเซต i เขียนแทนด้วย $(i_1,i_2,\ldots,i_m)$ โดยที่ i_j คือ ไอเท็ม และ ลำดับ s เขียนแทนด้วย $<s_1,s_2\ldots s_n>$โดยที่ s_j คือ ไอเท็มเซต

นิยาม 4.3 *ลำดับ* $<a_1,a_2,\ldots,a_n>$ *เป็นลำดับย่อยของ* $<b_1,b_2,\ldots,b_m>$ *ถ้ามีตัวเลข* $i_1<i_2<\ldots<i_n$ *ซึ่ง* $a_1\subseteq b_{i1},a_2\subseteq b_{i2},\ \ldots\ ,a_n\subseteq b_{in}$

ตัวอย่าง

ลำดับ <(3) (4 5) (8)> เป็นลำดับย่อยของ <(7) (3 8) (9) (4 5 6) (8)> เพราะ (3)$\subseteq$(3 8), (4 5)$\subseteq$(4 5 6) และ (8)$\subseteq$(8)

แต่<(3 5)>ไม่เป็นส่วนประกอบของ<(3) (5)>รวมทั้งในทางกลับกันด้วย เพราะ <(3) (5)> หมายถึง 3 และ 5 ปรากฏในคนละครั้งกัน แต่<(3 5)>หมายถึง 3 และ 5 ปรากฏในครั้งเดียวกัน

เราสามารถมองทรานแซคชันทั้งหมดของลูกค้าคนหนึ่งเป็นลำดับได้โดยที่ทรานแซคชันแต่ละทรานแซคชันคือไอเท็มเซต และลิสต์ของทรานแซคชันที่จัดเรียงตามเวลา คือ ลำดับ ซึ่งเราเรียกลำดับนี้ว่าลำดับของลูกค้า (customer-sequence) ถ้าทรานแซคชันของลูกค้าที่จัดเรียงตามเวลาเป็น T_1 , T_2 ,..., T_n และไอเท็มเซตใน T_i เขียนแทนด้วย itemset(T_i) แล้วจะได้ว่าลำดับของลูกค้าคือลำดับ <itemset(T_1),itemset(T_2), ... ,itemset(T_n)>

ลำดับของลูกค้าจะมีลำดับ s เป็นส่วนประกอบ ถ้าลำดับ s เป็นลำดับย่อยของลำดับของลูกค้า และค่าซัพพอร์ตของลำดับจะมีค่าเท่ากับจำนวนลูกค้าที่ซัพพอร์ตลำดับนี้

เมื่อกำหนดฐานข้อมูลทรานแซคชันการซื้อสินค้าของลูกค้า D มาให้ ปัญหาของการขุดค้นรูปแบบลำดับคือการหาลำดับทั้งหมดซึ่งมีค่าซัพพอร์ตไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนด โดยที่ความยาว (length) ของลำดับคือ ค่าจำนวนของไอเท็มเซตในลำดับ และเรียกลำดับที่มีความยาว k ว่า k-sequence และเราเรียกลำดับซึ่งมีค่าซัพพอร์ตไม่น้อยกว่าค่าซัพพอร์ตขั้นต่ำนี้ว่า ลำดับ

ขนาดใหญ่ (large sequence) และเรียกลำดับแต่ละลำดับนี้ว่ารูปแบบลำดับ (sequential pattern)

เนื่องจากค่าซัพพอร์ตเคานท์ของไอเท็มเซต i คือค่าจำนวนของลูกค้าซึ่งซื้อของทั้งหมดในไอเท็มเซตดังนั้นไอเท็มเซต i และลำดับที่มีสมาชิกหนึ่งตัว (1-sequence) <i> จึงมีค่าซัพพอร์ตเท่ากันและเรียกไอเท็มเซตที่มีค่าซัพพอร์ตเคานท์ไม่น้อยกว่าค่าซัพพอร์ตขั้นต่ำว่า ไอเท็มเซตขนาดใหญ่ (large itemset) เนื่องจากแต่ละไอเท็มเซตในลำดับขนาดใหญ่ต้องมีค่าซัพพอร์ตเคานท์ไม่น้อยกว่าค่าซัพพอร์ตขั้นต่ำดังนั้นจึงทำให้ลำดับขนาดใหญ่เป็นลิสต์ของไอเท็มเซตขนาดใหญ่ด้วย

นิยาม 4.4 *$I=\{i_1,i_2,\ldots,i_m\}$ เป็นเซตของตัวอักษรซึ่งเรียกว่าไอเท็ม (item) D คือ เซตของทรานแซคชันซึ่งแต่ละทรานแซคชัน T คือเซตของไอเท็มซึ่ง $T\subseteq I$ X คือ เซตของไอเท็มใน I และทรานแซคชัน T มี X เป็นส่วนประกอบถ้า $X\subseteq T$ Y คือ เซตของไอเท็มใน I และทรานแซคชัน T มี Y เป็นส่วนประกอบถ้า $Y\subseteq T$ เมื่อกำหนดลำดับ S=X.Y (โดยที่ " . " แทนการต่อกันของลำดับ 2 ลำดับ)*

รูปแบบลำดับเป็นลำดับที่อยู่ในรูปของ $X \overset{s,c}{\Rightarrow} Y$ โดยที่ $X\subset I, Y\subset I$ และ $X\cap Y=\varnothing$

- *ลำดับ $X\Rightarrow Y$ มีค่าซัพพอร์ต (support) เท่ากับ s ก็ต่อเมื่อ อัตราส่วนระหว่างจำนวนทรานแซคชันที่มี X . Y เป็นส่วนประกอบ กับ จำนวนทรานแซคชันทั้งหมดมีค่าเท่ากับ s*
- *ลำดับ $X\Rightarrow Y$ มีค่าความเชื่อมั่น (confidence) เท่ากับ c ก็ต่อเมื่อ อัตราส่วนระหว่างจำนวนทรานแซคชันที่มี X . Y เป็นส่วนประกอบ กับ จำนวนทรานแซคชันที่มี X ประกอบอยู่มีค่าเท่ากับ c*

ตัวอย่าง

0.4,0.8

/company/products -> /company/products/file2.html

จากลำดับสามารถอธิบายได้ว่าทรานแซคชันจำนวน 80 เปอร์เซ็นต์ของทรานแซคชันทั้งหมดที่เยี่ยมชม /company/products แล้วจะเยี่ยมชม /company/products/file2.html ด้วย โดยที่จำนวนของทรานแซคชันที่เยี่ยมชมทั้งสองเพจคิดเป็น 40 เปอร์เซ็นต์ของทรานแซคชันทั้งหมด

4.2.1 อัลกอริทึม Apriori All

Apriori All เป็นอัลกอริทึมหนึ่งที่ใช้ในการเรียนรู้รูปแบบลำดับซึ่งจะหาลำดับทั้งหมด ซึ่งมีทรานแซคชันซัพพอร์ตมากกว่าค่าซัพพอร์ตขั้นต่ำ โดยที่ค่าซัพพอร์ตของลำดับ คือ จำนวน ทรานแซคชันที่มีลำดับนั้นเป็นส่วนประกอบ และเรียกลำดับที่มีค่าซัพพอร์ตมากกว่าค่าซัพพอร์ตขั้นต่ำว่าลำดับขนาดใหญ่

```
L1={large 1-sequences}; // result of large itemset phase
For (k=2; Sk-1≠∅; k++)
   Begin
      Ck= apriorialI-gen(Sk-1); // New candidate generated from Sk-1
      for each customer sequence c in database do
         Increment the count of all candidates in Ck that are contained in c
      Sk = Candidates in Ck with minimum support.
   End
Answer = Maximal Sequences in Uk Sk
```

รูปที่ 9 อัลกอริทึม Apriori All

จากรูปที่ 9 เราสามารถอธิบายได้ว่า ในแต่ละรอบเราจะใช้ลำดับขนาดใหญ่จากรอบก่อนมาสร้างเป็นลำดับคู่แข่งแล้วทำการนับค่าซัพพอร์ตจากฐานข้อมูลทรานแซคชัน และที่ตอนท้ายของรอบจะทำการกำหนดลำดับขนาดใหญ่โดยพิจารณาจากค่าซัพพอร์ตของลำดับคู่แข่ง แต่ยกเว้นในรอบแรกที่เซตของลำดับขนาดใหญ่ที่มีสมาชิกหนึ่งตัว (1-large sequence) จะหาได้โดยทำการนับค่าซัพพอร์ตของแต่ละไอเท็มจากในฐานข้อมูลแล้วจึงทำการเปรียบเทียบค่าซัพพอร์ตของแต่ละไอเท็มกับค่าซัพพอร์ตขั้นต่ำ

4.2.2 วิธีการสร้างลำดับคู่แข่ง

ฟังก์ชัน apriorialI-gen ในรูปที่ 10 นำ S_{k-1} คือเซตของลำดับขนาดใหญ่ที่มีสมาชิก k-1 ตัว มาเป็นอาร์กิวเมนต์และคืนค่าลำดับคู่แข่งที่มีสมาชิก k ตัว ที่เป็นไปได้ทั้งหมด ซึ่งฟังก์ชันมีการทำงานดังนี้

ขั้นแรกจะทำการประสาน (join step) S_{k-1} กับ S_{k-1} ซึ่งมีสมาชิก k-2 ตัวแรกเหมือนกันเข้าด้วยกันเพื่อสร้างลำดับคู่แข่งที่มีสมาชิก k ตัวและต่อมาจะทำการลบลำดับ $s \in S_k$ ทั้งหมดซึ่งมีลำดับย่อยบางตัวของ s ที่มีสมาชิก k-1 ตัวที่ไม่อยู่ใน S_{k-1} ซึ่งจะเห็นได้ว่าวิธีการสร้างลำดับคู่แข่งนี้จะเป็นการจัดลำดับไอเท็มซึ่งถือว่าลำดับก่อนหลังมีความแตกต่างกัน แต่ในวิธีการสร้างไอเท็มเซตคู่แข่งที่ได้กล่าวไปแล้วใน 4.1.2 จะไม่ถือว่าลำดับก่อนหลังมีความแตกต่างกัน

insert into C_k

select p.itemset $_1$, ... , p.itemset $_{k-1}$, q.itemset $_{k-1}$

from S_{k-1} p , S_{k-1} q

where p.itemset $_1$ = q.itemset $_1$,..., p.itemset $_{k-2}$ = q.itemset $_{k-2}$

รูปที่ 10 ฟังก์ชัน apriorialI-gen (join step)

ขั้นตอนต่อมาคือขั้นตอนการตัดเล็ม (*prune step*) ดังแสดงในรูปที่ 11 จะลบลำดับทั้งหมด $s \in S_k$ ซึ่งมีลำดับย่อยที่มีสมาชิก k-1 ตัวบางลำดับย่อยของ s ซึ่งไม่อยู่ใน L_{k-1} เพราะลำดับที่มีสมาชิก k ตัวจะเป็นลำดับขนาดใหญ่ ถ้าลำดับย่อยที่มีสมาชิก k-1 ตัวทั้งหมดของมันเป็นลำดับขนาดใหญ่

for all itemsets $c \in C_k$ do

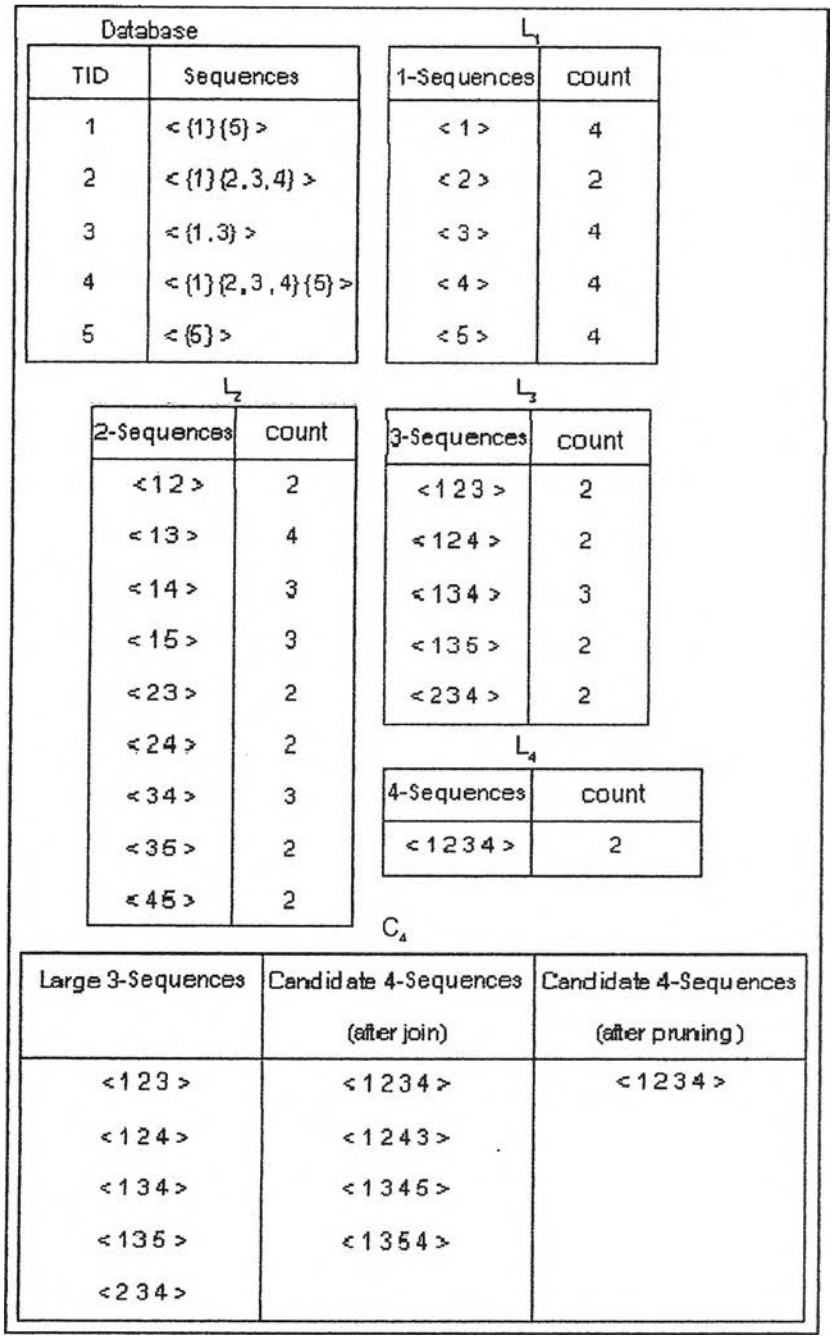
for *all* (k-1)-*sub*sets *s of* c do

if ($s \notin L_{k-1}$) then

delete c from C_k;

รูปที่ 11 ฟังก์ชัน apriorialI-gen (prune step)

ตัวอย่าง



Database

TID	Sequences
1	<{1}{5}>
2	<{1}{2,3,4}>
3	<{1,3}>
4	<{1}{2,3,4}{5}>
5	<{5}>

L_1

1-Sequences	count
<1>	4
<2>	2
<3>	4
<4>	4
<5>	4

L_2

2-Sequences	count
<12>	2
<13>	4
<14>	3
<15>	3
<23>	2
<24>	2
<34>	3
<35>	2
<45>	2

L_3

3-Sequences	count
<123>	2
<124>	2
<134>	3
<135>	2
<234>	2

L_4

4-Sequences	count
<1234>	2

C_4

Large 3-Sequences	Candidate 4-Sequences (after join)	Candidate 4-Sequences (after pruning)
<123>	<1234>	<1234>
<124>	<1243>	
<134>	<1345>	
<135>	<1354>	
<234>		

รูปที่ 12 ตัวอย่างการทำงานของอัลกอริทึม Apriori All

รูปที่ 12 แสดงตัวอย่างการทำงานของอัลกอริทึม Apriori All ซึ่งสามารถอธิบายได้ดังนี้ ขั้นแรกจะนับค่าซัพพอร์ตของแต่ละไอเท็มจากทรานแซคชันทั้งหมดซึ่งแสดงอยู่ในตารางฐานข้อมูล แล้วกำหนดว่ามีไอเท็มใดบ้างที่เป็นลำดับขนาดใหญ่โดยพิจารณาจากค่าซัพพอร์ตซึ่งต้องไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนด คือ 2 ซึ่งทำให้ได้ลำดับขนาดใหญ่ที่มีสมาชิก 1 ตัว 5 ลำดับ คือ <1> <2> <3> <4> และ <5> ดังแสดงในตาราง L_1 ในรอบต่อมาจะนำลำดับขนาดใหญ่จากใน

L_1 มาสร้างเป็นลำดับคู่แข่งที่มีสมาชิก 2 ตัวและจะกำหนดลำดับที่เป็นลำดับขนาดใหญ่โดยพิจารณาจากค่าซัพพอร์ตซึ่งต้องไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนดซึ่งทำให้ได้ลำดับขนาดใหญ่ที่มีสมาชิก 2 ตัว 9 ลำดับ ดังแสดงในตาราง L_2 ในรอบต่อมาจะนำลำดับขนาดใหญ่จากใน L_2 มาสร้างเป็นลำดับคู่แข่งที่มีสมาชิก 3 ตัว และจะกำหนดลำดับที่เป็นลำดับขนาดใหญ่โดยพิจารณาจากค่าซัพพอร์ตซึ่งต้องไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนดซึ่งทำให้ได้ลำดับขนาดใหญ่ที่มีสมาชิก 3 ตัว ดังแสดงในตาราง L_3 ในรอบต่อมาในขั้นตอนการประสานจะนำลำดับขนาดใหญ่จากใน L_3 มาสร้างเป็นลำดับคู่แข่งที่มีสมาชิก 4 ตัว 4 ลำดับ ดังแสดงในสดมภ์ที่ 2 ของตาราง C_4 และต่อมาในขั้นตอนการตัดเล็มจะตัดทอนลำดับคู่แข่งทุกตัวซึ่งมีลำดับย่อยที่มีสมาชิก 3 ตัวที่ไม่อยู่ใน L_3 ออกไปซึ่งทำให้เหลือลำดับคู่แข่งเพียงหนึ่งลำดับ ดังแสดงในสดมภ์ที่ 3 ของตาราง C_4 และต่อมาจะกำหนดลำดับคู่แข่งว่าเป็นลำดับขนาดใหญ่หรือไม่ โดยพิจารณาจากค่าซัพพอร์ตซึ่งต้องไม่ต่ำกว่าค่าซัพพอร์ตขั้นต่ำที่กำหนดซึ่งทำให้ได้ลำดับขนาดใหญ่ที่มีสมาชิก 4 ตัว 1 ลำดับ ดังแสดงในตาราง L_4 ในรอบสุดท้ายจะนำลำดับขนาดใหญ่จากใน L_4 มาสร้างเป็นไอเท็มเซตคู่แข่งที่มีสมาชิก 5 ตัว แต่เนื่องจากไม่สามารถสร้างลำดับคู่แข่งที่มีสมาชิก 5 ตัวได้และทำให้ไม่สามารถสร้างลำดับขนาดใหญ่ที่มีสมาชิก 5 ตัวได้ด้วย ดังนั้นอัลกอริทึม Apriori All จึงหยุดการทำงาน

4.2.3 วิธีการสร้างรูปแบบลำดับ

ในการสร้างรูปแบบลำดับจะทำการหาลำดับย่อยทั้งหมดที่มีสมาชิกมากกว่าหนึ่งตัวของลำดับขนาดใหญ่ s ทุกตัว ดังนั้นสำหรับลำดับย่อย a ทุกๆตัวของ s จะสร้างรูปแบบลำดับที่อยู่ในรูป $a \Rightarrow (s-a)$ ถ้าอัตราส่วนของค่าซัพพอร์ตเคานท์ของ s ต่อค่าซัพพอร์ตเคานท์ของ a มีค่าไม่ต่ำกว่าค่าความเชื่อมั่นขั้นต่ำ

เราสามารถปรับปรุงกระบวนการข้างบนได้โดยสร้างลำดับย่อยของลำดับขนาดใหญ่โดยทำซ้ำไปในทางลึกก่อน ตัวอย่างเช่น เมื่อกำหนดลำดับ ABCD มาให้ ขั้นแรกพิจารณาลำดับย่อย ABC และตัวถัดไปคือ AB ฯลฯ และต่อมาถ้าลำดับย่อย a ของลำดับขนาดใหญ่ s ไม่สามารถสร้างรูปแบบลำดับได้ก็ไม่จำเป็นต้องพิจารณาลำดับย่อยอื่นของ a เพื่อมาใช้ในการสร้างรูปแบบลำดับที่ใช้ s อีก

ตัวอย่างเช่น ถ้า $ABC \Rightarrow D$ มีค่าความเชื่อมั่นไม่เพียงพอแล้วก็ไม่จำเป็นต้องเช็ค $AB \Rightarrow CD$ อีก ซึ่งวิธีการนี้จะไม่ทำให้พลาดรูปแบบลำดับใดๆเพราะว่าค่าซัพพอร์ตเคานท์ของลำดับย่อย $\tilde{a}$ ของ a จะมีค่ามากกว่า a ซึ่งทำให้ค่าความเชื่อมั่นของรูปแบบลำดับ $\tilde{a} \Rightarrow (s-\tilde{a})$ จะมีค่าไม่มากไปกว่าค่าความเชื่อมั่นของ $a \Rightarrow (s-a)$ นั่นหมายความว่าถ้า a ไม่สามารถสร้างรูปแบบลำดับจาก s ได้แล้วก็จะไม่สามารถสร้างรูปแบบลำดับจาก $\tilde{a}$ ได้เช่นกัน ซึ่งแนวความคิดนี้สามารถสร้างเป็นอัลกอริทึมได้ดังแสดงในรูปที่ 13

```
// อัลกอริทึมเริ่มต้น
for all large sequences s_k , k ≥ 2 do
        call genpatterns(s_k);
// ฟังก์ชัน genpatterns สร้างรูปแบบลำดับ a⟹(s_k- a)
procedure genpatterns (s_k: large k-sequence)
        i = k-1
        while i>=1  begin
        a= subsequence of s_k that is first i elements of s_k
         conf = support (s_k) / support(a);
           if (conf≥minconf) then
              output the rule a ⟹(s_k- a) , with confidence = conf and support = support(s_k);
           else
              exit loop
        i = i –1
        end
end
```

รูปที่ 13 ฟังก์ชัน genpatterns

จากฟังก์ชัน genpatterns ในรูปที่ 13 สามารถอธิบายขั้นตอนการทำงานได้ดังนี้

- ขั้นแรกฟังก์ชัน genpatterns จะนำลำดับขนาดใหญ่ที่มีจำนวนสมาชิก k ตัวซึ่งเป็นอาร์กิวเมนต์ s_k มาหาลำดับย่อยซึ่งเป็นสมาชิก i ตัวแรกของ s_k เก็บเอาไว้ใน a
- ขั้นต่อมาจะนำ a มาหาค่าความเชื่อมั่น conf โดยคำนวณได้จากค่าซัพพอร์ตของ s_k หารด้วยค่าซัพพอร์ตของ a จากนั้นจึงนำค่า conf ที่ได้มาพิจารณาว่ามีค่าต่ำกว่าค่าความเชื่อมั่นขั้นต่ำที่กำหนดหรือไม่ ถ้าหากมีค่าไม่ต่ำกว่าค่าความเชื่อมั่นขั้นต่ำจะสร้างรูปแบบลำดับ a ⟹ (s_k-a) โดยที่ค่าซัพพอร์ตของรูปแบบลำดับจะมีเท่ากับค่าซัพพอร์ตของ s_k และมีค่าความเชื่อมั่นเท่ากับ conf แต่ถ้า conf มีค่าต่ำกว่าค่าความเชื่อมั่นขั้นต่ำจะหยุดการทำซ้ำทันที โดยการทำซ้ำจะทำตั้งแต่ขั้นตอนแรกโดยกำหนดให้ค่า i เริ่มต้นที่ k-1 และลดค่า i ลงทีละหนึ่งจนกว่าค่า i จะมีค่าน้อยกว่า 1

4.3 สรุป

ขั้นตอนวิธีการเรียนกฎความสัมพันธ์จะใช้อัลกอริทึม Apriori ซึ่งมีขั้นตอนการทำงานดังนี้

ในรอบแรกจะทำการนับค่าซัพพอร์ตเคานท์ของแต่ละไอเท็มและหาว่าไอเท็มใดเป็นไอเท็มเซตขนาดใหญ่บ้างโดยเปรียบเทียบกับค่าซัพพอร์ตขั้นต่ำ และในรอบต่อๆมาจะเริ่มต้นด้วยการนำเอาเซตของไอเท็มเซตขนาดใหญ่ที่ได้จากรอบก่อนหน้ามาเป็นข้อมูลตั้งต้นเพื่อนำไปสร้างเป็นไอเท็มเซตคู่แข่ง (candidate itemset) ซึ่งมีสมาชิกเพิ่มขึ้นหนึ่งตัวที่สามารถเป็นไปได้ และต่อมาจะนับค่าซัพพอร์ตเคานท์ของไอเท็มเซตคู่แข่งเหล่านี้ และในตอนท้ายของรอบจะพิจารณาว่ามีไอเท็มเซตคู่แข่งตัวใดบ้างที่เป็นไอเท็มเซตขนาดใหญ่ (large itemset) ซึ่งไอเท็มเซตขนาดใหญ่เหล่านี้จะถูกนำไปเป็นข้อมูลตั้งต้นของรอบถัดไป โดยที่กระบวนการนี้จะดำเนินไปเรื่อยๆจนกว่าจะไม่มีไอเท็มเซตขนาดใหญ่อันใหม่เกิดขึ้น หลังจากได้ไอเท็มเซตขนาดใหญ่ทั้งหมดที่เป็นไปได้แล้วจะนำไอเท็มเซตขนาดใหญ่เหล่านี้ไปสร้างเป็นกฎความสัมพันธ์โดยใช้อัลกอริทึม genrules

ขั้นตอนวิธีการเรียนรู้รูปแบบลำดับจะใช้ Apriori All ซึ่งจะมีขั้นตอนการทำงานคล้ายกับอัลกอริทึม Apriori แต่จะมีความแตกต่างกันที่วิธีการสร้างลำดับคู่แข่งโดยจะถือว่าลำดับก่อนหลังมีความแตกต่างกัน หลังจากได้ลำดับขนาดใหญ่ทั้งหมดที่เป็นไปได้แล้วจะนำลำดับขนาดใหญ่เหล่านี้ไปสร้างเป็นรูปแบบลำดับโดยใช้อัลกอริทึม genpatterns