Sorbonne Université

Projet STL
Extraction de motifs graduels fermés fréquents sous contrainte de temporalité

Composition du trinôme :
Muhannad Alabdullah
Lea El Abboud
Esma Hocini

Encadrants :
Mehdi Naima
Lionel Tabourier

18 février 2024

Table des matières

Chapitre 1

Introduction générale

1.1 Présentation du projet

Pendant leurs opérations quotidiennes, de nombreuses entreprises commerciales collectent une quantité considérables d'informations pour étudier les comportements de leurs clients. Les supermarchés, par exemple, accumulent un nombre important de données sur les achats des consommateurs en évaluant le contenu de leurs paniers, ce qui les aide à améliorer les stocks et la disposition des produits en rayon, ainsi que de créer un système de recommandations pour leur magasin en ligne.

Dans le cadre de notre projet, nous visons ainsi à développer des algorithmes efficace pour l'extraction de motifs graduels fermés fréquents. En priorité, on recherche des motifs fermés, ce qui signifie qu'il n'existe pas de motif plus grand ayant la même valeur du support.

Ce travail nécessitera la mise en place de structures de données et d'algorithmes spécifiques pour traiter efficacement les ensembles de données de grande taille et complexité.

1.2 Définition

Avant de détailler notre approche, il est primordial de se familiariser avec certaines définitions fondamentales.

Définition (Item). *Un item fait référence à tout objet appartenant à un ensemble fini d'éléments distincts* $\mathcal{I} = \{x_1, x_2, \ldots, x_m\}$.

Définition (Itemset). *Un ensemble d'éléments de* $\mathcal{I}$ *est désigné comme un itemset. Plus précisément, un itemset composé de k éléments est nommé un k-itemset.*

Définition (Transaction). *Une transaction est un itemset identifié par un identificateur unique T id.*

Définition (Support). *La fréquence d'un itemset désigne le nombre de fois que cet itemset apparaît dans un ensemble de transactions donnée.*

Définition (Support minimum). *Le support minimum représente la fréquence minimale requise pour qu'un itemset soit considéré comme fréquent.*

Définition (Itemset fréquent). *Un itemset est dit fréquent si son support est au-delà du support minimum fixé à priori.*

Chapitre 2

Algorithme d'extraction des itemsets fréquents

2.1 Algorithme Apriori

Pour réaliser ce type de problème, nous allons nous appuyer sur l'algorithme Apriori proposé par Agrawal et Srikant en 1994. Il est reconnu pour sa capacité à identifier des propriétés qui reviennent fréquemment dans un ensemble de données et d'en déduire une catégorisation.

2.1.1 Principe de base

L'algorithme Apriori implémente une approche itérative pour trouver les itemsets fréquents. Tout d'abord, il détermine le support des 1-itemsets en effectuant une première passe sur la base de données. Les itemsets dont le support ne rencontre pas le seuil minimum défini sont qualifiés de non fréquents, Par la suite, un nouvel ensemble des 2-itemsets, désignés comme les itemsets candidats, est formé. Après un autre passage sur la base de données, les supports de ces 2-itemsets candidats sont calculés. Ceux jugés non fréquents sont rejetés. Le processus décrit précédemment est maintenu jusqu'à ce que l'on ne puisse plus générer les itemsets fréquents.

La fonction Apriori-gen(L_{k-1}) est utilisée pour générer des candidats et prend L_{k-1} comme paramètre, produisant ainsi un ensemble de tous les k-itemsets candidats. Cette opération se déroule en deux étapes : d'abord la jonction de L_{k-1} avec L_{k-1}. Ensuite, lors de l'étape d'élagage, tout itemsets c dans C_k tels qu'il existe un sous-ensemble de c de taille *k-1* qui n'est pas dans L_{k-1} sont supprimés.

2.1.2 Pseudo-code

Table 2.1 – Notation

L_k	Ensemble des k-itemsets fréquents de taille k (avec support minimum). Chaque membre de cette ensemble possède deux attributs : i) itemset et ii) support count.
C_k	Ensemble de candidats k-itemsets. Chaque membre de cette ensemble possède deux attributs : i) itemset et ii) support count.

Algorithm 1 Apriori

Entrées : Base de transaction D, Support minimum S
Sorties : Ensemble L_k des itemsets fréquents
$L_1 \leftarrow$ {1–itemsets fréquents}
for $(k \leftarrow 2, L_{k-1} \neq \emptyset; k \leftarrow k+1)$ **do**
    $C_k \leftarrow$ Apriori-Gen(L_{k-1}); // nouveaux candidats
    for all transaction $t \in D$ **do**
        $C_t \leftarrow$ Subset(C_k, t);
        for all candidat $c \in C_t$ **do**
            c.support $\leftarrow c$.support $+ 1$;
        end for
    end for
    $L_k \leftarrow \{c \in C_k | c.\text{support} \geq \text{S}\}$;
end for
fin
retourner $\bigcup_k L_k$

Algorithm 2 Apriori-Gen

Entrées : Ensemble L_{k-1} de $(k-1)$-itemsets fréquents
Sorties : Ensemble C_k k-itemsets candidats
insérer dans C_k ;
sélectionner $p[1], p[2], ..., p[k-1], q[k-1]$;
de L_{k-1}, L_{k-1} ;
où $p[1] = q[1], ..., p[k-2] = q[k-2], p[k-1] < q[k-1]$;
for all itemset candidat $c \in C_k$ **do**
    for all sous-ensemble s de c de taille $k-1$ **do**
        if $s \notin L_{k-1}$ **then**
            supprimer c de C_k ;
        end if
    end for
end for
fin
retourner C_k

2.1.3 Exemple

Pour démontrer les étapes successives de l'algorithme Apriori, nous exécuterons un petit exemple en choisissant un support minimum S = 2.
Dans la première étape, chaque item de transaction T est un 1-itemset de C_1 Un premier passage sur la base de donnée D permet d'obtenir le support de chaque 1-itemset, ceux qui atteignent ou dépassent le support minimum sont considérés comme fréquents et conservés dans L_1. Ensuite, une jointure de L_1 x L_1 est réalisée pour déterminer l'ensemble C_2. Un deuxième parcours de D est effectué pour trouver le support de 2-itemsets candidats. Les 2-itemsets jugés fréquents sont retenus dans C_2 et on utilise une jointure de L_2 x L_2 pour obtenir les candidats C_3. Une troisième passe sur D permet de trouver les 3-itemsets fréquents. Les itemsets n'ayant pas le support supérieur ou égal ou support minimum sont supprimés pour construire L_3. De nouveau, on réalise une jointure de L_3 x L_3 pour trouver les candidats C_4. Cet ensemble est vide, car on n'a plus qu'un seul élément de taille 3.

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

TABLE 2.2 – Base de données des transactions

C_1	
Itemsets	Support
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1	
Itemsets	Support
{1}	2
{2}	3
{3}	3
{5}	3

TABLE 2.3 – Premier passage sur la base de données pour trouver des ensembles d'éléments uniques

C_2	
Itemsets	Support
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2	
Itemsets	Support
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

TABLE 2.4 – Deuxième parcours pour trouver des ensembles d'éléments de taille 2

C_3	
Itemsets	Support
{2 3 5}	2

L_3	
Itemsets	Support
{2 3 5}	2

TABLE 2.5 – Troisième parcours pour trouver des ensembles d'éléments de taille 3

2.2 Les structures de données

Pour optimiser l'extraction de motifs fréquents à partir de grandes bases de données transactionnelles, nous avons conçu des structures de données spécifiques pour soutenir les algorithmes que nous avons implémentés.

2.2.1 Structure des Itemsets (item_set)

La structure *item_set* est utilisée pour représenter les itemsets identifiés lors du processus de fouille de données. Elle contient un tableau dynamique pour stocker les items de cet ensemble, leur nombre (k) et un compteur d'occurrences (count) utilisé pour XXXXXXXXX enregistrer le nombre d'apparition de cet itemset dans l'ensemble de données.

2.2.2 Table de hachage

Afin de gérer efficacement les grandes quantités d'itemsets et de faciliter les opérations fréquentes, nous avons intégré une structure de table de hachage, une composante clé pour notre algorithme.

2.2.2.1 Structure et fonctionnement de la table de hachage

La table de hachage est composée d'un tableau de pointeurs *pointers*, où chaque entrée pointe vers un *hash_node*, une structure de donnée qui sert de conteneur pour les *item_set*, offrant ainsi un accès aux itemsets et leur données associées.

La fonction de hachage attribue à chaque itemset un indice dans la table, qui déterminé ou le noeud correspondant doit être stocké.

2.2.2.2 Gestion des collisions

La gestion des collisions est un aspect important de la performance de la table de hachage. Lors de l'insertion d'un nouvel itemset , la fonction de hachage calcule un indice qui détermine où le *hash_ node* correspondant devrait être place dans le tableau de pointeurs. L'indice calculé sera également conservé dans le *hash_ node* pour une récupération efficace. En cas de collision, quand plusieurs itemsets sont attribués le même indice de hachage, les *hash_ node* sont reliés en liste chaînée grâce au pointeur *next*. Chaque noeud ajouté est connecté au suivant, permettant ainsi de résoudre les collisions tout en préservant l'accès rapide aux itemsets.

2.2.2.3 Avantages de la table de hachage

L'utilisation de la table de hachage offre une amélioration des performances de notre algorithme, surtout lorsqu'il est appliqué à de grandes bases de données transactionnelles. XXXXTEMPS CONSTANTXXX

2.3 Fonctions et Opérations

2.3.1 Opérations sur les Itemsets

Plusieurs opérations sont implémentées afin de manipuler les itemsets dans l'algorithme. Les fonctions clés incluent l'initialisation d'itemsets en créant un nouvel itemset vide (*init_ item_ set*), l'ajout d'éléments à un itemset existant tout en assurant qu'il n'y a pas de doublons (*add_ item_ to_ item_ set*), et la vérification de la présence d'un item dans un itemset (*item_ in_ item_ set*) ou dans la table de hachage (*item_ set_ in_ hash_ table*)

2.3.2 Construction

2.4 Algorithme AprioriTID

2.4.1 Principe de base

2.4.2 Pseudo-code

Notation. $\overline{C_1}$.

Algorithm 3 AprioriTID

$L_1 \leftarrow$ {1-itemsets fréquents}
$\mathcal{C}_1 \leftarrow$ Base de donnée D;
for $(k \leftarrow 2; L_{k-1} \neq \emptyset; k++)$ **do**
    $C_k \leftarrow$ Apriori-Gen(L_{k-1}); // Nouveaux candidats
    $\overline{C_k} \leftarrow \emptyset$;
    for all entrée $t \in \overline{C}_{k-1}$ **do**
        $\overline{C}_t \leftarrow \{c \in C_k | (c - c[k]) \in t.\text{set-of-itemsets} \wedge$
                $(c - c[k-1]) \in t.\text{set-of-itemsets}\}$;
        for all candidats $c \in \overline{C}_t$ **do**
            c.count $++$;
            if $(\overline{C}_t \neq \emptyset)$ **then**
                $\overline{C}_k \leftarrow \overline{C}_k \cup \{t.TID, \overline{C}_t\}$;
            end if
        end for
    end for
    $L_k \leftarrow \{c \in \overline{C}_k | c.\text{count} \geq \text{S}\}$
end for
Answer $\leftarrow \bigcup_k L_k$;

2.4.3 Exemple

Database	
TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

$\overline{C_1}$	
TID	Set-of-Itemsets
100	{ {1}, {3}, {4} }
200	{ {2}, {3}, {5} }
300	{ {1}, {2}, {3}, {5} }
400	{ {2}, {5} }

L_1	
Itemset	Support
{1}	2
{2}	3
{3}	3
{5}	3

C_2	
Itemset	Support
{1, 2}	1
{1, 3}	2
{1, 5}	1
{2, 3}	2
{2, 5}	3
{3, 5}	2

$\overline{C_2}$	
TID	Set-of-Itemsets
100	{ {1, 3} }
200	{ {2, 3}, {2, 5} {3, 5} }
300	{ {1, 2}, {1, 3} {1, 5} {2, 3}, {2, 5} {3, 5} }
400	{ {2, 5} }

L_2	
Itemset	Support
{1, 3}	2
{2, 3}	2
{2, 5}	3
{3, 5}	2

C_3	
Itemset	Support
{2, 3, 5}	2

$\overline{C_3}$	
TID	Set-of-Itemsets
200	{ {2, 3, 5} }
300	{ {2, 3, 5} }

L_3	
Itemset	Support
{2, 3, 5}	2

Chapitre 3

Etude expérimentale

Chapitre 4

Bilan