Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Informatique

Module: Fouille de donnée

Rapport du TP

L'Algorithme Apriori

Réalisé par:
FELLAH Wassim 202031031557
BOUSSIS Rafiq 202033029578

Version améliorée de l'algorithme Apriori

1)-Introduction:

Dans le domaine de la fouille de données, l'algorithme Apriori est un outil fondamental pour découvrir des ensembles d'éléments fréquents dans les bases de données transactionnelles. Cependant, l'algorithme Apriori traditionnel souffre souvent d'inefficacités, en particulier dans la génération d'ensembles d'éléments candidats, ce qui peut entraîner une surcharge de calcul importante. Dans ce rapport, nous proposons une méthode améliorée qui rationalise le processus de génération d'itemsets candidats, réduisant ainsi le temps de calcul et les ressources.

2)-Définition des termes :

Avant d'aborder l'approche proposée, définissons les termes clés pour plus de clarté :

1. Ensemble de transactions (T) : Désigné par $T = \{T1, T2, \dots, Tm\}$, où $m \geq 1$, T représente un ensemble de transactions. Chaque transaction $Ti = \{I1, I2, \dots, In\}$, où $n \geq 1$, est un ensemble d'items.
2. k-itemset : Désigné par $\{i1, i2, \dots, ik\}$, où $k \geq 1$, un k-itemset représente un ensemble de k items, et k-itemset $\subseteq$ I.
3. Nombre de supports (σ) : Le nombre de supports d'un itemset fait référence à la fréquence de son occurrence dans les transactions.
4. Itemset candidat (Ck) : Ck représente l'itemset candidat de taille k.
5. Itemset fréquent (Lk) : Lk désigne l'itemset fréquent de taille k.

3)-Approche proposée:

A. Définitions:

Dans le processus de l'Apriori, les définitions suivantes sont nécessaires :

Définition 1: Supposons que T = {T1, T2, … , Tm},(m ≥ 1) est un ensemble de transactions, Ti = {I1, I2, … , In},(n ≥ 1) est un ensemble d'items, et k-itemset = {i1, i2, … , ik},(k ≥ 1) est également un ensemble de k items, et k-itemset ⊆ I.

Définition 2: Supposons que σ (itemset), est le nombre de supports de l'itemset ou la fréquence d'occurrence d'un itemset dans les transactions.

Définition 3: Supposons que Ck est l'itemset candidat de taille k, et Lk est l'itemset fréquent de taille k.

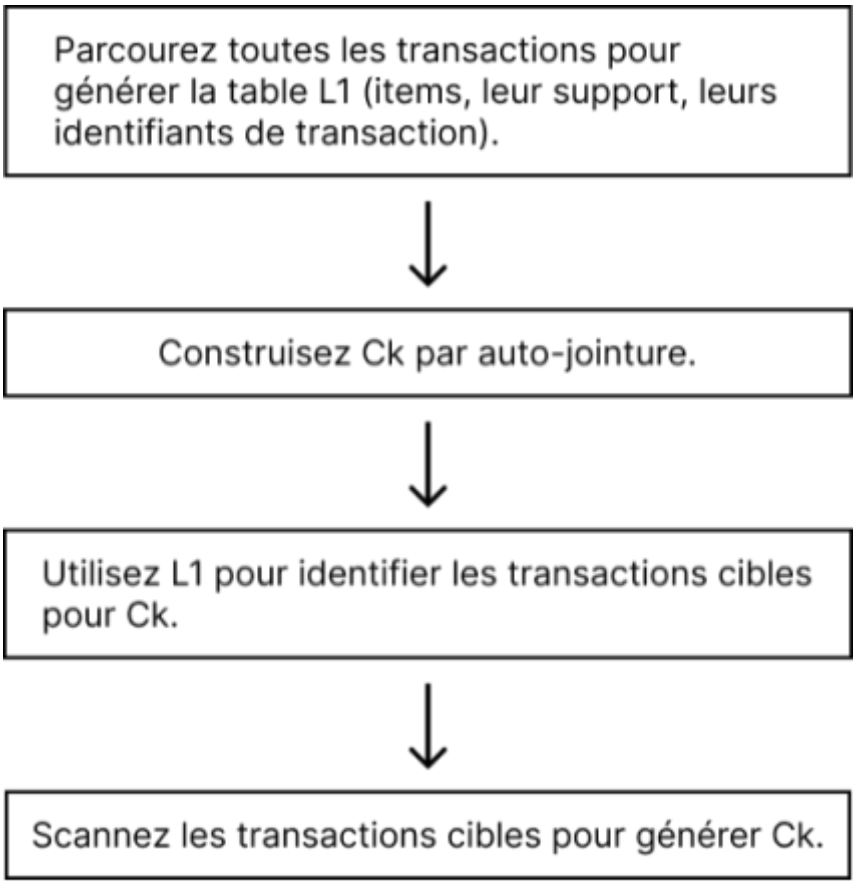

Figure 1: Steps for Ck generation

Dans notre approche proposée, nous améliorons l'algorithme Apriori pour réduire le temps nécessaire à la génération de candidats d'itemsets. Nous commençons par parcourir toutes les transactions pour générer L1, qui contient les items, leur nombre de supports, et l'ID de transaction où les items sont trouvés. Ensuite, nous utilisons L1 comme aide pour générer L2, L3... Lk. Pour générer C2, nous effectuons une auto-jonction L1 * L1 pour construire un 2-itemset C (x, y), où x et y sont les items de C2. Avant de scanner tous les enregistrements de transactions pour compter le nombre de supports de chaque candidat, nous utilisons L1 pour obtenir les IDs de transaction de la valeur minimale de support entre x et y, et donc pour scanner C2 uniquement dans ces transactions spécifiques. La même méthode est appliquée pour C3, construisant un 3-itemset C (x, y, z), où x, y et z sont les items de C3, et utilisant L1 pour obtenir les IDs de transaction de la valeur minimale de support entre x, y et z, puis pour scanner C3 uniquement dans ces transactions spécifiques. Ces étapes sont répétées jusqu'à ce qu'aucun nouvel itemset fréquent ne soit identifié. L'ensemble du processus est montré dans la **Figure 1**.

B. L'algorithme Apriori amélioré:

L'amélioration de l'algorithme peut être décrite comme suit :

//Générer les items, leurs supports, leurs ID de transaction
(1) L1 = find_frequent_1_itemsets (T);

(2) Pour (k = 2; Lk-1 ≠Φ; k++) {
//Générer le Ck à partir du LK-1

(3) Ck = candidats générés à partir de Lk-1;
//Obtenir l'item Iw avec le support minimal dans Ck en utilisant L1, (1≤w≤k).

(4) x = Get_item_min_sup(Ck, L1);

//Obtenir les IDs de transaction cibles contenant l'item x.]

(5) Tgt = get_Transaction_ID(x);

(6) Pour chaque transaction t dans Tgt
(7) Augmenter le décompte de tous les items dans Ck qui sont trouvés dans Tgt;

(8) Lk = items dans Ck ≥ min_support;

(9) Fin; }

C. An example of the improved Apriori:

Supposons que nous ayons un ensemble de transactions (T) D avec 9 transactions, 5 items (I), et le support minimal = 3. L'ensemble de transactions est présenté dans **Table 1**.

TID	Items
T_1	I_1, I_2, I_5
T_2	I_2, I_4
T_3	I_2, I_4
T_4	I_1, I_2, I_4
T_5	I_1, I_3
T_6	I_2, I_3
T_7	I_1, I_3
T_8	I_1, I_2, I_3, I_5
T_9	I_1, I_2, I_3

Table 1: les transactions

Items	Support	
I_1	6	
I_2	7	
I_3	5	
I_4	3	
I_5	2	supprime

Table 2: The candidate 1-itemset

Premièrement, parcourez toutes les transactions pour obtenir le 1-itemset fréquent L1, qui contient les items, leur nombre de supports et les IDs des transactions contenant ces items. Ensuite, éliminez les candidats qui sont peu fréquents ou dont le support est inférieur au minimum requis. Le 1-itemset fréquent est présenté dans **Table 3**.

Items	Support	T_IDs	
I_1	6	$T_1, T_4, T_5, T_7, T_8, T_9$	
I_2	7	$T_1, T_2, T_3, T_4, T_6, T_8, T_9$	
I_3	5	T_5, T_6, T_7, T_8, T_9	
I_4	3	T_2, T_3, T_4	
I_5	2	T_1, T_8	supprime

Table 3: Frequent 1_itemset

L'étape suivante consiste à générer un candidat de 2-itemset à partir de L1. Pour obtenir le nombre de supports pour chaque itemset, divisez chaque itemset de 2-itemset en deux éléments, puis utilisez la table L1 pour déterminer les transactions où vous pouvez trouver l'itemset, plutôt que de les rechercher dans toutes les transactions. Par exemple, prenons le premier item dans le tableau 4 (I1, I2). Dans l'Apriori original, nous parcourons les 9 transactions pour trouver l'item (I1, I2) ; mais dans notre algorithme amélioré proposé, nous divisons l'item (I1, I2) en I1 et I2 et obtenons le support minimum entre eux en utilisant L1, ici I1 a le support minimum le plus bas. Ensuite, nous recherchons l'itemset (I1, I2) uniquement dans les transactions T1, T4, T5, T7, T8 et T9.

Items	Support	Min	T_IDs	
I_1, I_2	4	I_1	$T_1, T_4, T_5, T_7, T_8, T_9$	
I_1, I_3	4	I_3	T_5, T_6, T_7, T_8, T_9	
I_1, I_4	1	I_4	T_2, T_3, T_4	supprime
I_2, I_3	3	I_3	T_5, T_6, T_7, T_8, T_9	
I_2, I_4	3	I_4	T_2, T_3, T_4	
I_3, I_4	0	I_4	T_2, T_3, T_4	supprime

Table 4: Frequent 2_itemset

La même méthode s'applique pour générer un 3-itemset en fonction de la table L1, comme illustré dans **table 5**.

Items	Support	Min	T_IDs	
I_1, I_2, I_3	2	I_3	$T_1, T_4, T_5, T_7, T_8, T_9$	supprime
I_1, I_2, I_4	1	I_4	T_2, T_3, T_4	supprime
I_1, I_3, I_4	0	I_4	T_2, T_3, T_4	supprime
I_2, I_3, I_4	0	I_4	T_2, T_3, T_4	supprime

Table 5: Frequent 3_itemset

Nous avons testé la différence entre l'implémentation de l'Apriori original et notre Apriori amélioré, et nous avons rassemblé 5 groupes différents de transactions comme suit :

- T1 : 555 transactions.
- T2 : 900 transactions.
- T3 : 1230 transactions.
- T4 : 2360 transactions.
- T5 : 3000 transactions.

La première expérience compare le temps nécessaire à l'exécution de l'Apriori original et de notre algorithme amélioré en appliquant les cinq groupes de transactions dans l'implémentation.

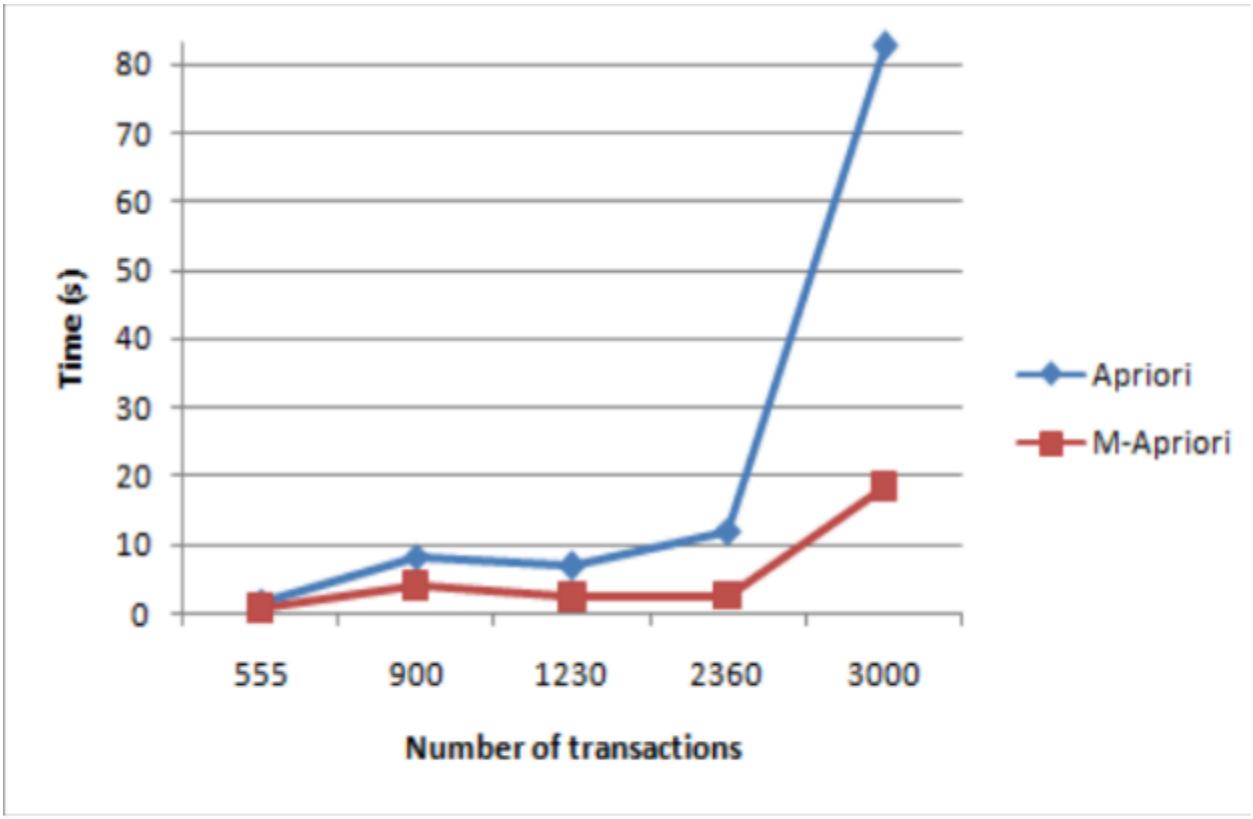

Figure 2: Time consuming comparison for different groups of transactions

La deuxième expérience compare le temps nécessaire à l'exécution de l'Apriori original et de notre algorithme proposé en appliquant un groupe de transactions avec différentes valeurs de support minimum dans l'implémentation. Le résultat est présenté dans la **Figure 3**.

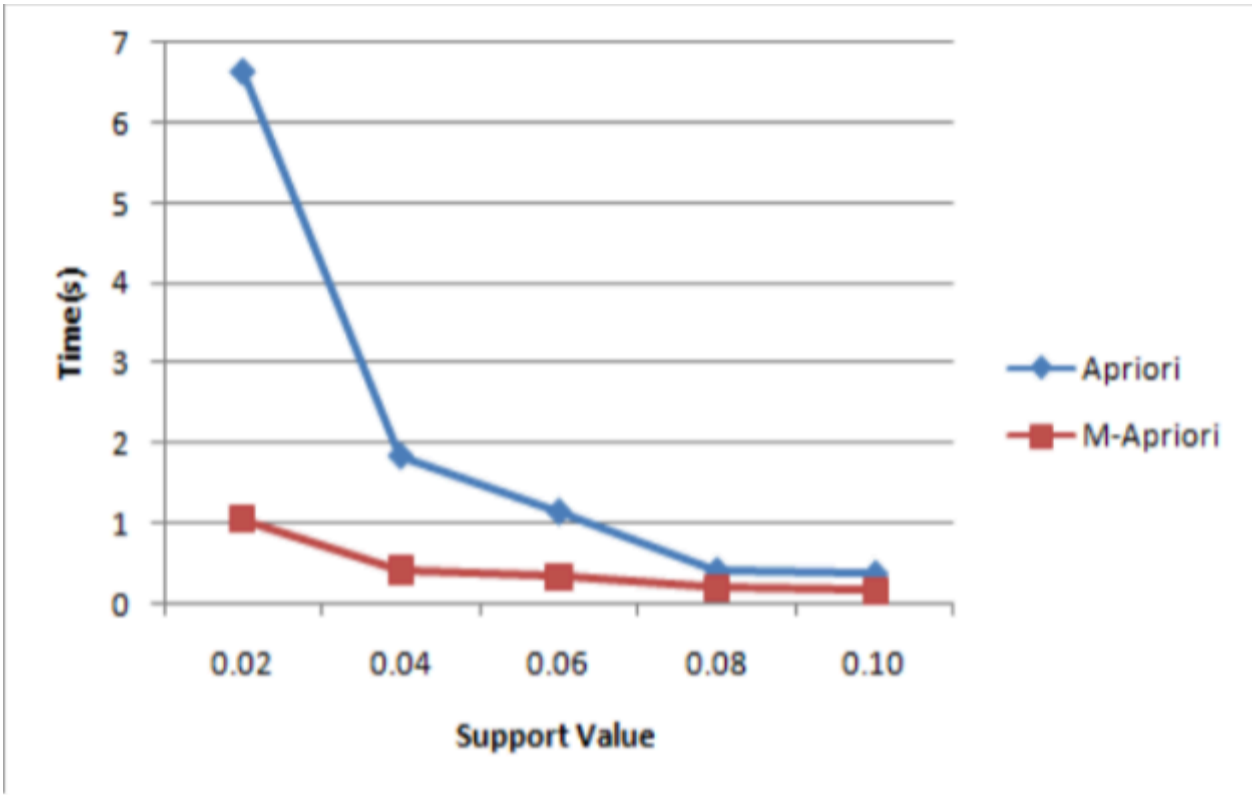

Figure 3: Time consuming comparison for different values of minimum support

4)-Support minimum dynamique:

A. Définitions:

Dans le processus de génération de support minimum dynamique pour l'Apriori amélioré, les définitions suivantes sont nécessaires :

I. Variables:

Interest_items:

La variable "interest_items" représente le nombre d'items jugés intéressants dans le jeu de données. Elle est calculée en fonction du ratio du nombre de transactions au nombre d'items distincts dans le jeu de données.

La variable interest_items est calculée à l'aide des équations suivantes :

Donné:

- Nb_transactions: est le nombre total de transactions dans le jeu de données
- Nb_items: est le nombre total d'items distincts dans le jeu de données

L'équation pour calculer interest_items est :

Interest_items = Nb_transactions / Nb_items

II. Fonctions:

get_items_support:

La fonction get_items_support calcule les valeurs de support pour chaque item donné dans le jeu de données et les stocke dans une liste. Les valeurs de support sont ensuite triées par ordre décroissant..

- Python code:

```
def items_support(item_list, transactions):
    list_supp = []
    for item in range(1, len(item_list) + 1):
    support = get_support(transactions, {item})
    list_supp.append(support)
    list_supp.sort(key = None, reverse = True)

    return list_supp
```

get_interest_items:

La fonction get_interest_items calcule le nombre d'items intéressants en fonction de la longueur des transactions (Nb_transactions) et de la liste d'items (Nb_items).

- Python code:

```
def get_interest_items(item_list, transactions):
    num_interest_items = len(transactions) /
len(item_list)
    num_interest_items =
math.ceil(num_interest_items)

    return num_interest_items
```

get_min_support:

La fonction get_min_support calcule le seuil de support minimum pour l'algorithme Apriori en utilisant la liste des supports d'items et le nombre d'items intéressants. Elle vérifie d'abord si la longueur de list_support_items est supérieure ou égale au nombre d'items intéressants. Si c'est le cas, elle calcule la valeur moyenne du support à partir des interest_items les plus élevés en additionnant leurs supports et en divisant par interest_items. La fonction renvoie cette moyenne comme valeur de support minimum. Si la liste est trop courte, un message indique un manque d'items suffisants.

- Python code:

```
def get_min_support(interest_items,
list_support_items):

     if len(list_support_items) >=  interest_items:

     for index in range(interest_items):
          min_supp += list_support_items[index]

     min_supp = float(min_supp / interest_items )

     return min_supp
     else:
          print("No more items inside
list_support_items")
```

B. Algorithme de support minimum dynamique :

L'amélioration de l'algorithme peut être décrite comme suit :

// Générer le support des items et le stocker dans une liste ordonnée par ordre décroissant list_support_items :
(1) - list_support_items = items_support(item_list, transactions)

// Calculer le nombre d'items intéressants pour le jeu de données fourni :
(2) - interest_items = get_interest_items(item_list, transactions)

// Calculer la valeur de min_support :
(3) - min_support = get_min_support(interest_items, list_support_items)

// Exécuter l'algorithme Apriori amélioré :
(4) - frequent_item_sets = apriori(min_support)

// Générer les règles d'associations :
(5) - association_rules = association_rule(min_confidence, item_support_dict)

// Choisir une valeur seuil pour évaluer la qualité du support minimum en comparant le nombre de règles d'associations à la valeur seuil :
(6) threshold = int(len(transactions) * 40 / 100) // notez que 40 % peut être modifié en fonction du choix de l'utilisateur de la valeur seuil

// Si le nombre de règles d'associations est inférieur à la valeur seuil, répétez l'étape (4) à (5), en supprimant chaque fois une valeur de list_support_items, jusqu'à ce que nous trouvions le meilleur support minimum possible pour le jeu de données fourni :

```
(7) while( is_optimal_min_supp == False){
        Si le nombre de règles d'associations < seuil){
                list_support_items.pop(premier de la liste)
                Répétez l'étape (4)
                Répétez l'étape (5)
        }

        Sinon{
                Is_optimal_min_supp = True// support minimum
                optimal trouvé
              }
}
```

(8) imprimer les règles d'associations ;
(9) Fin;

Step (7) python code:

```
is_optimal_min_supp = False
threshold = int(len(transactions) * 20 / 100)
print(f"The value of Threshold is: {threshold}")

while not is_optimal_min_supp:

    if len(association_rules) < threshold:
    if len(list_support_items) > interest_items:
        list_support_items.pop(0)
        min_support = get_min_support(interest_items,
list_support_items)
        print(f"The value of min_support is
{min_support}")

        frequent_item_sets = apriori(min_support)

        print(frequent_item_sets)
        item_support_dict = dict()
        items_list = list()

        key_list = list(item_dict.keys())
        val_list = list(item_dict.values())

        for level in frequent_item_sets:
            for set_support_pair in
frequent_item_sets[level]:
                for i in set_support_pair[0]:

items_list.append(key_list[val_list.index(i)])
                item_support_dict[frozenset(items_list)]
```

```
= set_support_pair[1]
                    items_list = list()

          association_rules =
association_rule(min_confidence, item_support_dict)
          print("Number of rules: ", len(association_rules),
"\n")
     else:
          print("Threshold was higher than anticipated")
          print(f"Best possible min support is:
{min_support}")
          break

     else:
     print(f"Optimal min support is {min_support}")
     is_optimal_min_supp = True
```

C. Un exemple de support dynamique :

Dans cet exemple, nous avons fourni un très petit ensemble de transactions contenant les items : téléphone portable, ordinateur portable, chargeur, batterie externe et tablette. Tout d'abord, nous calculons le support de chaque item et stockons ces valeurs dans une liste. La liste des supports des items est ensuite triée par ordre décroissant, comme illustré dans la **Figure 1**.

```
item list is ['handphone', 'laptop', 'charger', 'powerbank', 'tablet']
The list_support_items is:  [0.8, 0.6, 0.6, 0.4, 0.4]
```

Figure 1: list of items with corresponding support

Ensuite, nous calculons le nombre d'items intéressants, comme illustré dans la **Figure 2**.

```
def get_interest_items(item_list, transactions):
    num_interest_items = len(transactions) / len(item_list)
    num_interest_items = math.ceil(num_interest_items)

    return num_interest_items


interest_items = get_interest_items(item_list, transactions)
print(f"The number of interest items is:  {interest_items}")
```

```
The number of interest items is:  2
```

Figure 2: Calculate number of interest items in the dataset

Ensuite, nous calculons la valeur du support minimum en utilisant la fonction get_min_support, ce qui nous donne la valeur initiale du support minimum.

```
min_support = get_min_support(interest_items, list_support_items)

print(f"The value of min_support is {min_support}")
frequent_item_sets_per_level = apriori(min_support)
```

```
The value of min_support is 0.7
```

Il ne reste plus qu'un seul item avec un support supérieur au support minimum, qui est le téléphone portable avec 0,8. Cela aboutit à 0 règle d'association, ce qui n'est pas idéal et suggère que le seuil est trop bas. Pour trouver une meilleure valeur de support minimum, nous recalibrons en supprimant le premier item de list_support_items et en répétant le processus jusqu'à atteindre la valeur de support minimum optimale de 0,4, comme illustré dans la **Figure 4**.

Figure 4: Optimal minimum support for the given dataset alongside the association rules.

5)-Conclusion:

En conclusion, notre méthode améliorée offre une approche simplifiée de la génération de candidats d'itemsets dans le cadre de l'algorithme Apriori. En exploitant judicieusement les informations préexistantes, en optimisant le processus de balayage et en ajustant dynamiquement le seuil de support minimum, nous réduisons efficacement les coûts de calcul. Cette approche améliore considérablement l'efficacité et la capacité à traiter de grands volumes de données dans l'extraction d'itemsets fréquents dans les bases de données transactionnelles.