République Algérienne Démocratique et Populaire

Ministère de l’Enseignement Supérieur et de la Recherche Scientifique

**Université des Sciences et de la Technologie Houari Boumediene**

Département Informatique

# Module : Data Mining

**Résumé du livre « Rapport sur Apriori et FP-growth »**

**Réalisation par :**

**-CHABANE Nouar 201400007379**

**-BEZGALI Meriem 201300005476**

**Groupe : 01**

1. **Apriori**  
    L'algorithme Apriori est un algorithme classique dans l'exploration de données, Il est utilisé pour extraire les objets fréquents et des règles d'association, il est nommé Apriori car il utilise des connaissances préalables sur les propriétés fréquentes des ensembles d'éléments, il applique une approche itérative ou une recherche par niveau dans laquelle les ensembles d'éléments k-fréquents sont utilisés pour trouver k + 1 éléments.

*Remarque :* une règle d’association peut être définie comme une implication, X⟶Y où X et Y.

* Le pseudo-code de l'algorithme est donné ci-dessous comme suit : pour une base de données de transactions « T », ainsi qu'un seuil de support « minSupport », un ensemble de candidats Ck initialisé à vide au départ, et une liste d’item fréquent Lk initialisée a vide on commence par calculer à partir de « T » L1 en incrémentant de 1 la fréquence de chaque Item si on le trouve dans la base « T » ensuite on élimine les items qui ont un support inférieur  à «minSupport », aprés à partir de la liste L1 on génère les candidats C1 et on incrémente leurs fréquence en consultant la base de donnée « T » puis on recalcule L2 en prenant tout les éléments de C1 qui ont un support > ou = « minsupport », et ainsi de suite jusqu'à ce qu’on ne puisse plus générer de nouveau candidats.

|  |
| --- |
| Procedure-Apriori(T,minSupport)  {  Ck : candidate itemset of size k   Lk :frequent itemset of size k  L1={frequent items} ;  For(k=1 ;Lk != empty ; k++)  Do begin   Ck+1=candidates generated from Lk ;   for each transaction t in Database T Do begin   increment the count of all candidates in Ck+1 that contained in t Lk+1=candidates in Ck+1with min support   } end   Return Lk ; |

* Déroulement avec un support minimum = 2

*(1< support\_min) L1*

|  |  |
| --- | --- |
| Transaction id | items |
| 1 | 1/3/4 |
| 2 | 2/3/5 |
| 3 | 1/2/3/5 |
| 4 | 2/5 |

|  |  |
| --- | --- |
| Item set | Support |
| {1} | 2 |
| {2} | 3 |
| {3} | 3 |
| {4} | 1 |
| {5} | 3 |

|  |  |
| --- | --- |
| Item set | Support |
| {1} | 2 |
| {2} | 3 |
| {3} | 3 |
| {5} | 3 |

*(1< support\_min) L2 L3*

|  |  |
| --- | --- |
| Item set | Support |
| {1 2} | 1 |
| {1 3} | 2 |
| {1 5} | 1 |
| {2 3} | 2 |
| {2 5} | 3 |
| {3 5} | 2 |

|  |  |
| --- | --- |
| Item set | Support |
| {1 3} | 2 |
| {2 3} | 2 |
| {2 5} | 3 |
| {3 5} | 2 |

|  |  |
| --- | --- |
| Item set | Support |
| {2 3 5} | 2 |

*Remarque :* {1,2,3} {1,2,5} et {1,3,5} ne sont pas dans C3, et la solution est donnée par l’union de L1, L2 et L3.

* L’inconvénient majeur de cet algorithme est le nombre considérable de l’accès à la base de données « T » et son avantage est qu’il est simple.

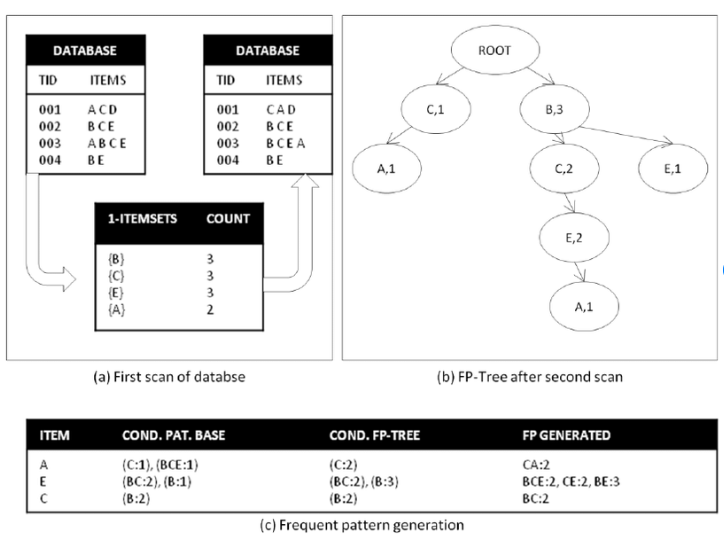
1. **FP-Growth**

Une méthode efficace et évolutive pour extraire les objets fréquents et des règles d'association sans la génération des ensembles des candidats (le cas d’Apriori).

* Pseudo code : il faut d’abord générer la structure appelé Fp-Tree qui est une représentation de la base de donnée compressée et qui montre la fréquence des items, après on peut faire l’extraction des items directement sur le FP-tree construit comme le montre l’algorithme suivant :

|  |
| --- |
| Algorithme Fp-growth 1. Balayer la base de transactions T une premeiere fois   * Pour créer L, la liste des items fréquents avec leur support * Trier L en ordre décroissant du support   2. Créer l’arbre N contenant une racine étiquetée « Null »  3. Procédure Fp\_growth(fp-tree,Null)  Si Fp-tree contient un seul chemain P alors  Pour chaque combinaison Beta de p faire  generer l’itemset Beta union alpha de support =minimum des supports des   nœuds des Beta  Sinon pour chaque Alpha dabs l’indexe de fp-tree faire  générer l’itemset Beta=A i union alpha support = A i.support   Construire l’itemset condition de base de Beta  Construire fp-tree(Beta)  si fp-tree(Beta)= vide  Fp-growth(fp-tree(Beta,Beta) Fin. |

* Déroulement sur un exemple :



* Les Inconvénients de FP-Growth :
* la FP-Tree consomme trop de mémoire si elle est volumineuse et coute cher à construire.
* Avantages de FP-Growth :
* On ne fait que 2 passages sur le jeu de données pour construire la Fp-tree.
* On compresse l’ensemble des données.
* Pas de génération de candidats.
* Est beaucoup plus rapide qu'Apriori.