Apprentissage automatique. Règles d'association

Mihaela JUGANARU-MATHIEU
mathieu@emse.fr

École Nationale Supérieure des Mines de St Etienne

2019-2020

Bibliographie :

- Oded Maimon, Lior Rokach (Editors) - *Data Mining and Knowledge Discovery Handbook*, Second Edition, Springer 2010
- Jiawei Han, Micheline Kamber - *Data Mining : Concepts and Techniques*, Morgan Kaufmann Publication, 2006

Contenu du cours

MINES Saint-Étienne

Définition d'une règle d'association

Les règles d'association ont fait une bonne partie du succès du Data Mining (DM).
On met en évidence des motifs fréquents qui apparaissent dans les données à explorer.
Traitements associés à l'exploration de données de type "panier d'achat".

Une **règles d'association** s'exprime comme :

$$A \Rightarrow B$$

avec une probabilité de support $P(A \cup B)$ et une probabilité de confiance $P(B|A)$.

Définition d'une règle d'association

Exemple :

$$computer \Rightarrow antivirus\_software[support = 2\%, confidence = 60\%]$$

Signification :

- *2% des achats contiennent "computer" et "antivirus_software"*
- *60% des achats qui contiennent "computer", contiennent aussi "antivirus_software"*

Une association est intéressante si son support et sa confiance dépasse les seuils pré-établis de *min_support* et *min_confiance*.

Définition d'une règle d'association

Parents :

- Pietesky et Shapiro, 1991 (règle d'association)
- Agrawal et son équipe, 1993 : l'algorithme Apriori de calcul des règles d'association

Plusieurs algorithmes de calculs connus :

- Apriori - l'algorithme le plus connu, parcours BFS
- Eclat - algorithme basé sur une recherche en profondeur en utilisant l'intersection d'ensembles
- FP-growth - algorithme qui utilise un sous-arbre de préfixes
- GUHA procedure ASSOC - génération automatique des hypothèses avec une vérification en utilisant les données en tableau
- OPUS algorithme de recherche qui travaille sans aucune hypothèse de monotonie en partant d'une conclusion donnée
- ...

Définitions formelles règles d'association

- **itemset** $I = \{i_1, i_2, \ldots, i_m\}$ un ensemble d'éléments (attributs / items).
- **transaction** t : un sous-ensemble de I
- *tid* : un identifiant de transaction qui est unique. On traitera avec (tid, t) une transaction unique.
- D un ensemble relevant des transactions :

$$D = \{t_1, t_2, \ldots, t_n\}$$

  D est appelée aussi **base de données de transaction**
- si a est un itemset $a \subset I$, a se dit k-itemset si $card(a) = k$
- le support d'un itemset a :

$$support\_count(a) = card(\{t \in D | a \subset t\})$$

$$support(a) = \frac{card(\{t \in D | a \subset t\})}{card(D)}$$

MINES Saint-Étienne

Définitions formelles règles d'association

- une **règle d'association** est de forme

$$a \rightarrow b$$

avec a et b ensembles de disjoints de itemsets.

$$a, b \subset i \text{ et } a \cap b = \phi$$

- une règle d'association $a \rightarrow b$ est dite "supportée par la transaction t" si $a \cup b \subset t$
- le support d'une règle d'association :

$$support(a \rightarrow b) = p(a \cup b) = \frac{support\_count(a \cup b)}{card(d)}$$

Définitions formelles règles d'association

- confiance de $a \to b$ est :

$$confiance(a \to b) = p(b|a)$$

$$confiance(a \to b) = \frac{p(a \cup b)}{p(a)} = \frac{support\_count(a \cup b)}{support\_count(a)}$$

La confiance est facile à calculer pour $a \to b$ et $b \to a$ si on connait les supports de itemsets a, b et $a \cup b$.

Lift

En plus du support et de la confiance pour une règle d'association on peut considérer le lift :

$$lift(a \to b) = \frac{p(b|a)}{p(b)} = \frac{support(a \cup b)}{support(a) \times support(b)}$$

Si $lift(r) < 1$ c'est une règle qui ne sert à rien.

Exemples :

$$lift(Vertbr \to Cerveau) = 1$$

bien que $confiance(Vertbr \Rightarrow Cerveau) = 100\%$, *la valeur LIFT indique une règle qui n'apporte rien.*

$$lift(fumer \to cancer) = \frac{3\%}{1\%} = 3$$

Signification : fumer augmente 3 fois le risque d'un cancer.

Objectifs de la fouille de règles d'association :

On est intéressé par des itemsets dit **fréquents** : ayant un support supérieur à une limite *min_sup*.

On est aussi intéressé à découvrir des règles d'association dites **fortes** : ayant une confiance supérieure à une limite *min_conf*.

Recherche de règles d'association

La recherche de règles d'associations peut être vue comme un processus à deux pas :

- recherche des itemsets fréquents : **tous** les itemsets X qui satisfont $support(X) > min\_support$ difficile !
- recherche des règles d'association fortes : **toutes** les règles R avec $confiance(R) > min\_conf$ plus facile ! si on dispose des itemsets fréquents

Combinatoire

Pour un quelconque nombre de transaction utilisant d attributs il y a :

$$2^d \text{ itemsets}$$

Le nombre de règles d'association possibles avec partie gauche et droite non nulles est :

$$3^d - 2^{d+1} + 1$$

Une stratégie gloutonne (greedy) de génération de tous les itemsets n'est pas envisageable.

Recherche de règles d'association

Déduction des règles à partir des itemsets :

- prendre chaque I itemset fréquent ($support(I) > min\_support$)
- décomposer I de toutes les manières possibles

$$I = X \cup Y \text{ avec } X \cap Y = \phi$$

et calculer $confiance(X \Rightarrow Y)$ et $confiance(Y \Rightarrow X)$. Retenir les règles si elles sont fortes.

Remarque : si le itemset $X \cup Y$ est fréquent alors X et Y sont aussi des itemsets fréquents.

Itemsets fréquents

Dans certains cas ce sont les itemset fréquents qui nous intéressent. Au lieu de tout afficher, on peut s'en tenir au itemset fréquent fermés ou maximaux.

Un **itemset fréquent** X est **fermé** (*closed frequent itemset*) s'il n'y a pas de itemset X' le contenant $X \subset X'$ qui ait la même fréquence.

Un **itemset fréquent** X est **maximal** (*maximal frequent itemset*) s'il n'y pas de plus large itemset fréquent le contenant.

Monotonie - itemsets et règles d'association

Monotonie : Si X fréquent et $Y \subset X$ (Y est un sous-ensemble), alors Y est fréquent.
Anti-monotonie : Si X n'est pas fréquent et $X \subset Z$ (Z est un sur-ensemble), alors Z n'est pas, non plus, fréquent.

Si la règle $R = X \Rightarrow Y$ n'est pas forte et $X \subset X'$, $Y \subset Y'$ avec $X \cap Y = X' \cap Y' = \emptyset$, alors $R' = X' \Rightarrow Y'$ n'est pas forte non-plus.

Cette remarque permet de générer plus rapidement les règles d'association fortes à partir d'un itemset fréquent.

Recherche de règles d'association

Idée, avoir tous les itemsets fréquents peut être très utile.

Remarques :

- le nombre des k-itemsets de I est de $\binom{m}{k}$ (parce que $card(I) = m$)
- le nombre total de itemsets est de $2^m - 1$

Une génération exhaustive de tous les itemsets (fréquents ou pas) suivi d'une vérification est beaucoup trop couteuse.

Algorithme Apriori

1994 : Agrawal et son équipe.

Idées :

- générer itérativement tous les k-itemsets fréquents
- à partir de $k = 1$ jusqu'à un k' pour lequel il n'y a plus de k'-itemset fréquent
- pour chaque niveau k on se base sur les calculs faits au niveau $k - 1$
- on se base sur les propriétés de monotonie et anti-monotonie.

Algorithme Apriori

Notations :

- L_k les k-itemsets fréquents (tous)
- C_k un ensemble de k-itemset candidats avec la propriété que $L_k \subset C_k$

Algorithme Apriori :

- Pas 1 : générer L_1 (tous les 1-itemsets fréquents) et $k \leftarrow 1$
- Tant que $L_{k-1} \neq \phi$ faire (Pas k) :
  - générer C_k par jointure $L_{k-1} \bowtie L_{k-1}$
  - parcourir la base D des transactions afin de calculer le support de chaque itemset de C_k.
    Élagage dans C_k des itemsets qui ne sont pas fréquents.
    $L_k \leftarrow elagage(C_k)$; $k \leftarrow k + 1$

Algorithme Apriori

On suppose que les éléments de I sont ordonnés et que dans chaque transaction et itemset calculé les items apparaissent en ordre croissante.

La **jointure** $L_{k-1} \bowtie L_{k-1}$ se calcule de la manière suivante :

- on prend toutes les paires P, Q de L_{k-1} qui ont $k-2$ éléments en commun et $p[k-1] < q[k-1]$

$$P = p[1], p[2], \cdots, p[k-1]$$

$$Q = p[1], p[2], \cdots, p[k-2], q[k-1]$$

- on construit $PQ = p[1], p[2], \cdots, p[k-1], q[k-1]$, on inclut PQ dans C_k

La propriété de anti-monotonie nous assure que ce procédé est correct et complet : tout k-itemset fréquent se retrouve dans C_k.

Algorithme Apriori

Le **parcours** de D pour le calcul de $support(C)$ avec $C \in C_k$, k-itemset candidat :

- pour chaque T transaction en D
  - pour chaque C k-itemset candidat dans C_k :
    si $C \subset T$, alors $compteur(C) + +$

Lors de l'élagage on élimine les $C \in C_K$ avec $compteur(C) \leq min\_support$

Algorithme Apriori

Le **parcours** est fait selon une stratégie en largeur d'abord.

Une autre version consiste à considérer non pas $L_{k-1} \bowtie L_{k-1}$, mais $L_{k-1} \bowtie L_1$, à savoir calculer les extensions possibles de chaque itemset de L_k avec un élément de L_1.
Problème supplémentaire : éviter le calcul redondant.

Dans R on peut utiliser la librairie `arules` pour l'application de l'algorithme Apriori ou le calcul des itemsets et aussi la librairie `arulesViz`.
Datasets intéressants : Titanic et Groceries. Usage data(Titanic) et data(Groceries)

Si toutes les transaction sont dans la **mémoire interne** on peut utiliser :

- tables de hachage
- B-arbres pour garder les transactions et les TID des transactions
- un arbre de préfixe pour garder les L_k successifs
- files de priorité (tas)
- un arbres de tables de hachage pour garder les candidats (candidats dans les feuilles et des tables de tables hachage dans les feuilles)

MINES Saint-Étienne

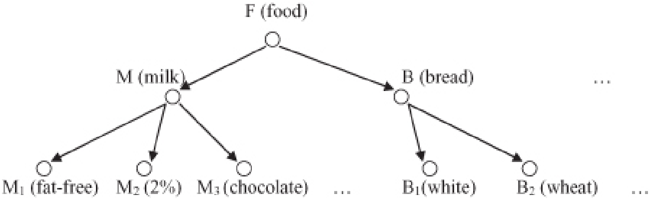
Si les transaction sont dans la **mémoire externe** dans une base de données, on prend dans une table normalisée de préférence.

On doit à chaque génération de candidats de taille k **faire un scan complet de toute la base.**

Le comptage se réalise avec SQL via des opérations d'autojointure. Un scan complet de la base est très coûteux.

Prise en compte des niveaux

Parfois des règles d'association fréquentés mais non-intéressantes (exemple : *pain* $\rightarrow$ *lait* à 80%). On peut bâtir une hiérarchies de niveaux et appliquer à chaque hiérarchie un support décroissant.



TID Apriori

Le but de cette version est de réduire le nombre de scans de la base de transactions.

Rappel : TID est l'identifiant (unique) d'une transaction

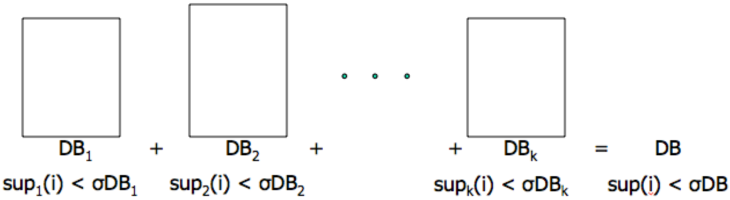
- Lors de la construction de chaque itemset de C_1 on leur associe aussi la liste des transactions qui le supporte
- pour chaque nouvel élément de C_k on fait l'intersection des listes des éléments de L_{k-1} qui l'ont généré

Avantage : un seul scan de la base.

Défaut : on manipule des listes qui peuvent être très longues.

Partition Apriori

Idée : partitionner en sous-ensemble disjoints qui tiennent en mémoire, appliquer **Apriori** sur chaque morceaux et consolider après les motifs (règles ou itemsets) par un scan complet de la base.

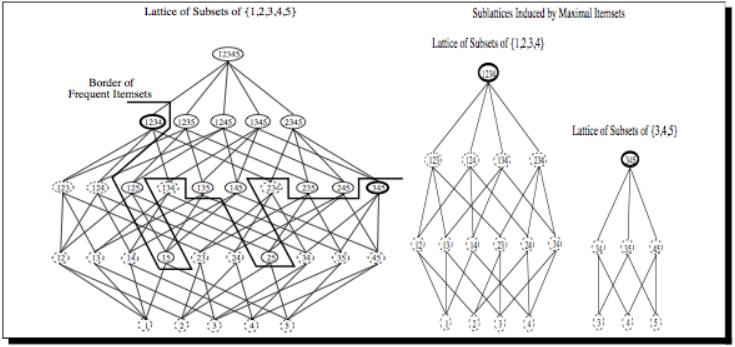


MINES Saint-Étienne

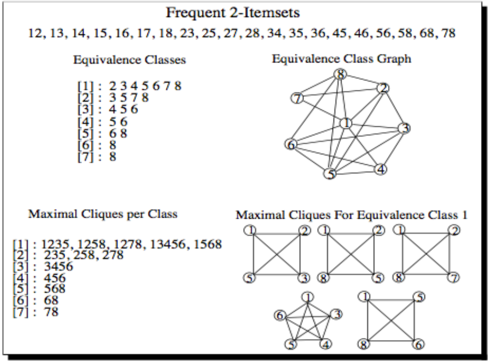
ECLAT : 1997, Zaki - algorithme pour trouver les itemsets fréquents.

Idée de base : dans la lattice des itemsets les itemsets fréquents (et maximaux) sont générés par des itemsets de taille plus petites et qui forment des classe d'équivalence en forme de clique (graphe complet).

ECLAT


Lattice of Subsets of {1,2,3,4,5}
Sublattices Induced by Maximal Itemsets
Lattice of Subsets of {1,2,3,4}
Lattice of Subsets of {3,4,5}
Border of
Frequent Itemsets
12345
1234
1235
1245
1345
2345
123
124
125
134
135
145
234
235
245
345
12
13
14
15
23
24
25
34
35
45
1
2
3
4
5

ECLAT


Frequent 2-Itemsets
12, 13, 14, 15, 16, 17, 18, 23, 25, 27, 28, 34, 35, 36, 45, 46, 56, 58, 68, 78
Equivalence Classes
[1] : 2 3 4 5 6 7 8
[2] : 3 5 7 8
[3] : 4 5 6
[4] : 5 6
[5] : 6 8
[6] : 8
[7] : 8
Equivalence Class Graph
Maximal Cliques per Class
[1] : 1235, 1258, 1278, 13456, 1568
[2] : 235, 258, 278
[3] : 3456
[4] : 456
[5] : 568
[6] : 68
[7] : 78
Maximal Cliques For Equivalence Class 1

ECLAT

Principes :

- On garde pour chaque itemset fréquent X l'ensemble de d'identifiants *tid* des transactions qui le supportent $t(X)$.
- On suppose aussi qu'on sait garder les itemsets sous une forme qui permet d'établir une relation d'ordre (ordre topologique, par exemple).
- On construit d'abord dans un scan de base l'ensemble des itemsets fréquents de taille 1 :

$$P_{init} = \{(i, t(i)) | i \in I, card(t(i)) \geq min\_support\}$$

- Au fur et à mesure de l'exécution de l'algorithme on cumule dans un ensemble résultat les itemsets fréquents $(Y, support(Y))$.
- Appel initial $ECLAT(P_{init}, min\_sup, \phi)$.

ECLAT

$ECLAT(P, min\_sup, F)$

foreach $\underline{(X_a, t(X_a)) \in P}$ **do**
- $F \leftarrow F \cup (X_a, support(X_a))$
- $P_a \leftarrow \phi$
- **foreach** $\underline{(X_b, t(X_b)) \in P \text{ with } X_b > X_a}$ **do**
  - $X_{ab} \leftarrow X_a \cup X_b$
  - $t(X_{ab}) \leftarrow t(X_a) \cap t(X_b)$
  - **if** $\underline{support(X_{ab}) \geq min\_support}$ **then**
    - $P_a \leftarrow P_a \cup (X_{ab}, support(X_{ab}))$
  - **end**
- **end**
- **if** $\underline{P_a \neq \phi}$ **then**
  - $ECLAT(P_a, min\_sup, F)$
- **end**

end

ECLAT

Caractéristiques :

- c'est une approche verticale (et on garde pour chaque itemset sa liste de transactions support)
- on procède à un parcours en profondeur et buttom-up (depuis des itemsets de petite taille)
- on considère de manière conceptuelle la classe d'équivalence et on cherche de manière récursive les sous-graphes complets

L'algorithme est implémenté dans R, librairie `arules` : `eclat`.