Retour sur les itemsets: justification du codage + comment travailler les TPs

Retour sur les itemsets et l'algorithme Apriori

Un *itemset* est un ensemble d'items. Dans le cours du 02.02.23, un certain nombre de définitions ont été fournies que nous allons revisiter

- Le **compteur de support** d'un *itemset* X c'est simplement le nombre de transactions où X apparaît
- Un *itemset* est dit **fréquent** si son compteur est supérieur au compteur support minimum
- Un *itemset* est dit **fréquent maximal** si aucun sur-ensemble n'est fréquent
- Un *itemset* X est dit **fermé** si aucun *itemset* de la forme $X \cup \{i_p\}$ n'a un compteur de support aussi grand que lui.

L'objectif de l'algorithme **Apriori** est de calculer les itemsets fréquents maximaux. Dit autrement, il s'agit de trouver les co-occurrences d'items les plus grandes possibles. La semaine dernière nous avons vu la méthode naïve qui consiste à chercher les itemsets de taille k fixée, et à ne garder que ceux qui possédaient un compteur de support plus grand que la valeur fixée a priori.

L'algorithme Apriori est basé sur une approche *incrémentale* qui consiste à repérer les itemsets fréquents de taille 1, puis ceux de taille 2 et ainsi de suite. Afin de rendre l'algorithme efficace, nous avons vu que l'on maintenait 2 tables. La première était indexée par les itemsets et contenait comme valeurs les transactions qui englobaient ces itemsets. La seconde était indexée par les transactions et contenait comme valeur les itemsets de longueur k.
Ainsi pour l'exemple suivant pour lequel on suppose que l'on fixe le support minimum à 2 transactions:

TID	items
100	[1,3,4]
200	[2,3,5]
300	[1,2,3,5]
400	[2,5]

L'algorithme construit les deux tables

TID	itemsets taille 1	itemsets	ensembles
100	{ {1}, {3}, {4}}	{1}	{100, 300}
200	{ {2}, {3}, {5}}	{2}	{200, 300, 400}
300	{{1}, {2}, {3}, {5}}	{3}	{100, 200, 300}
400	{{2}, {5}}	{4}	{100}
		{5}	{200, 300, 400}

Suit une phase d'élagage qui va supprimer les itemsets dont le compteur < 2 et va construire tous les itemsets possibles de taille **2**

- On supprime *4*. L'ensemble des itemsets fréquents de taille 1, noté $L_1 =$ { {1}, {2}, {3}, {5}}
- On construit les itemsets de taille 2

TID	itemsets taille 2
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}}

Puis la table

itemsets	ensembles	compteurs
{1,2}	{100}	1
{1,3}	{100, 300}	2
{1,5}	{300}	1
{2,3}	{200, 300}	2
{2,5}	{200, 300, 400}	3
{3,5}	{200, 300}	2

- On supprime {1,2} et {1,5}. L'ensemble des itemsets fréquents de taille 2, noté L_2 est { {1,3}, {2,3}, {2,5}, {3,5}}
- On construit les itemsets de taille 3

TID	itemsets taille 3
200	{{2,3,5}}

TID	itemsets taille 3
300	{{2,3,5}}

Puis la table

itemsets	ensembles	compteurs
{2,3,5}	{200,300}	2

Et l'algorithme s'arrête, il n'est plus possible d'étendre l'itemset de taille 3.

Codage

Pour représenter un objet, nous lui attribuons un numéro > 0. Deux objets différents ont un numéro différent, 2 numéros distincts font référence à 2 objets différents. Mathématiquement, on parle d'une bijection.
Pour représenter un itemset, nous allons utiliser des ***tuples*** python ordonnés par ordre croissant. De même pour représenter des ensembles d'itemsets nous allons utiliser un ordre croissant ainsi l'ensemble des itemsets { {3,2}, {5,2}, {5,3} } sera représenté par la ***liste*** python $[(2,3),(2,5),(3,5)]$. L'intérêt de ce codage est de pouvoir exploiter la propriété selon laquelle :

"Pour être fréquent, un itemset X doit vérifier que tout $Y \subset X$ est fréquent".

On va montrer que si L_k est correct, alors $c_1 = \{x_1, x_2, \dots x_k\}$ et $c_2 = \{y_1, y_2, \dots y_k\}$ permettront de construire $c = \{x_1, x_2, \dots, x_{k-1}, x_k, y_k\}$ si $\forall 1 \leq i < k, x_i = y_i$ et $x_k < y_k$ un itemset fréquent si et seulement si $\forall u \subset c, \mid u \mid = k, u \in L_k$ et qu'il n'existe pas d'itemsets de longueur $k+1$ qui soit fréquent et qui ne soit pas dans L_{k+1}

1. Il est évident, sous réserve que L_k soit correct, que si on peut trouver un $u \subset c, u \notin L_k$, c n'est pas fréquent (c'est la définition d'un itemset fréquent)
2. c est bien de taille $k+1$, par définition tout sous-ensemble de taille k est fréquent, sous réserve que L_k est correct, on doit avoir $\{x_1, x_2, \dots, x_{k-1}, x_k\}$ et $\{x_1, x_2, \dots, x_{k-1}, y_k\}$ dans L_k. Ces deux itemsets correspondent à c_1 et c_2
3. Supposons qu'il existe un itemset fréquent de longueur p n'appartenant pas à L_p, et prenons p le plus petit possible. $p > 1$, en effet, par construction L_1 contient tous les itemsets fréquents de taille 1. On sait, par hypothèse que L_{p-1} contient tous les itemsets fréquents de taille $p-1$ et que c est un itemset fréquent de taille p n'appartenant pas à L_p. Puisque l'itemset c est constitué de nombres entiers tous différents, on peut les trier par ordre croissant, on a donc un unique représentant trié qui correspond à c, notons le $a_1, a_2, \dots, a_p$. Puisque ce représentant n'est pas dans L_p, c'est que, par construction l'un, au moins des 2 itemsets $a_1, a_2, \dots, a_{p-2}, a_{p-1}$ et $a_1, a_2, \dots, a_{p-2}, a_p$ n'est pas dans L_p, ce qui est contraire à notre hypothèse, donc c n'existe pas.

Travailler les TPs

Dans cette partie, nous supposerons que vous avez récupéré et décompressé le fichier `panier.zip`.
Nous allons manipuler 3 fichiers

- `main_tests.py` le fichier qui contient les tests de chaque classe
- `apriori.py` le fichier dans lequel vous allez développer la classe `Apriori`
- `sortie_apriori.py` le fichier qui va permettre de commencer la mise au point de votre code

`main_tests`

Rien de spécial à part l'utiliser régulièrement pour vérifier que vous faites ce qui est demandé

`sortie_apriori`

C'est juste le copié-collé des bouts de code de la fiche projet

`apriori`

Là où vous placer le code qui vous est demandé. C'est aussi le fichier que vous m'envoyez chaque semaine.

La suite de la présentation se déroulera "en live" pendant la séance de jeudi matin

A suivre `>>>`