

Accélération de l'extraction de règles d'association

Sami BOULECHFAR*, Quentin DUBOIS*, Isabelle PASCUAL*, Mickaël WAJNBERG†

*Étudiants à TELECOM Nancy - Promotion 2023, Villers-Lès-Nancy, France

†

Université du Québec à Montréal, Montréal, Canada

Résumé—La fouille de données est une discipline visant à extraire les tendances et régularités présentes dans les grands jeux de données. On s'intéresse ici à améliorer un processus de fouille de données, l'analyse formelle de concepts, AFC, qui permet de générer une base de règles d'associations. En effet, ce processus produit un nombre important de règles dont certaines non nécessaires. Afin de l'optimiser, on souhaite appliquer un lissage sur le jeu de données avant d'exécuter l'AFC. Le processus de prétraitement que l'on souhaite utiliser est le k Unified Nearest Neighbor, $kUNN$, un système de recommandation permettant de prédire des valeurs inconnues dans un jeu de donnée binaire.

Mots clés : Data Mining, Analyse de concepts, Règles d'association, Clustering, k Unified Nearest Neighbor, Système de recommandation, Analyse formelle de concepts

I. INTRODUCTION :

Avec la production de données qui devient de plus en plus centrale et massive au quotidien, il devient primordial d'automatiser le traitement de celles-ci. C'est ainsi qu'intervient l'extraction de connaissances [2]. Dans cette optique d'automatisation, on cherche à formuler des hypothèses par la description des tendances d'un jeu de données et à les vérifier en tant que théorie [1]. Pour cela, on utilise un processus, la *fouille de données*. Ce dernier vise à extraire du jeu de données, les *régularités* et les *tendances* sous divers formats, tels que les *règles d'association*, similitudes entre les données [2].

Toutefois, étant donné le nombre important de données, le risque d'*explosion combinatoire* est présent, résultant d'une *génération trop importante* de règles d'association. De fait, il est nécessaire de trouver une méthode qui produit moins de règles tout en conservant l'information. Il faut donc créer une *base de règles* [1]. Pour ce faire plusieurs approches existent tel que l'*analyse formelle de concepts*, AFC, permettant d'obtenir une base sans restriction. Cette méthode calcule dans un premier temps les *concepts*, ensemble de parties d'objet ou d'attribut stable par double dérivation et c'est à partir de ces derniers que l'AFC déduit les *règles d'associations* [1].

Étant donné que l'AFC permet d'obtenir une base de règles, elle est utilisée à de multiples reprises pour des tâches de *fouilles de données* [8] [9] [10]. En revanche, le nombre de règles généré reste important notamment à cause de sa sensibilité à la *variabilité*.

On propose donc une approche permettant de diminuer cette quantité de règles, en *pré traitant* les jeux de données afin d'anticiper un traitement plus compact par AFC. On propose d'appliquer un *système de recommandation* avant l'AFC, le k -Unified Nearest Neighbor, $kUNN$.

La section II introduit en détail l'AFC. La section III présente les différents algorithmes possibles permettant l'accélération de l'AFC. La section IV explique les stratégies d'implémentation, en particulier sur le $kUNN$. La section V-E évalue la performance de l'algorithme suivant différentes *métriques*. Enfin, la section VI conclut avec les analyses de l'équipe et des pistes d'amélioration.

II. MOTIVATION :

La finalité de l'AFC, en terme d'extraction, est de trouver les *règles d'association* présentant les co-occurrences entre les *attributs* du contexte [1]. Pour cela l'AFC détermine tout d'abord les ensembles d'objets ayant des attributs communs, autrement dit des *clusters* nommés *concepts formels*. Toutefois, elle permet uniquement l'étude d'attributs et non de combinaison logique d'attributs. De fait, cette méthode s'applique exclusivement à des *jeux de données unaires*.

On appelle jeu de données unaire une matrice contenant pour chaque coefficient, soit une valeur, soit aucune. On définit également un *contexte formel* comme un triplet $K = (O, A, I)$ où

- O : Ensemble des objets
- A : Ensemble des attributs
- $I \subseteq O \times A$: Relation binaire appelée la relation d'incidence

Un *contexte formel* peut être représenté par une *table unaire* comme décrit dans l'exemple 2.1 :

Exemple 2.1: On considère cinq étudiants de Telecom Nancy notées : e_a, e_b, e_c, e_d et e_e . Ces étudiants peuvent avoir validé ou non les modules notés : m_1, m_2, m_3, m_4 et m_5 . On peut alors définir le contexte formel $K = (O, A, I)$ où :

- $O = \{e_a, e_b, e_c, e_d, e_e\}$: Ensemble des objets
- $A = \{m_1, m_2, m_3, m_4, m_5\}$: Ensemble des attributs

Un tel contexte peut être représenté sous forme de table unaire, où les objets et les attributs correspondent respectivement aux lignes et aux colonnes et la relation d'incidence I est

représentée par l'ensemble des croix. Ici, la croix liant l'objet e_a et l'attribut m_1 signifie que l'étudiant e_a a validé le module m_1 . D'où la représentation du tableau I.

	m_1	m_2	m_3	m_4	m_5
e_a	X			X	
e_b		X	X		
e_c	X			X	X
e_d			X		
e_e	X				X

TABLE I
TABLEAU UNAIRE

Afin de déterminer les *concepts formels* des *contextes formels*, l'AFC utilise une paire d'opérations de dérivation qui dénote les attributs (respectivement les objets) communément portés par (respectivement portant) un ensemble d'objets (respectivement attributs) [1]. L'exemple 2.2 illustre la dérivation.

Exemple 2.2: On considère le contexte formel défini dans l'exemple 2.1 et représenté dans la table I. Alors $\{e_b, e_d\}' = \{m_3\}$ car m_3 est le seul attribut communément porté par e_b et e_d i.e. les étudiants e_b et e_d ont communément validé le module m_3 . De même, on a aussi $\{m_1\}' = \{e_a, e_c, e_e\}$, car le module m_1 , n'a été validé que par les étudiants e_a, e_c et e_e .

On appelle alors *concept formel*, une paire

$C = (X, Y) \in \mathcal{P}(O) \times \mathcal{P}(A)$ telle que $Y = X'$ et $X = Y'$. Les *concepts formels* sont intrinsèquement liés à la notion de *classe d'équivalence* et une classe d'équivalence peut contenir un ou plusieurs minimaux nommés *générateurs* [1].

Exemple 2.3: On considère le contexte formel défini dans l'exemple 2.1 et représenté dans la table I. Alors le couple $(\{e_b\}, \{m_3\})$ n'est pas un concept, car on a $\{m_3\}' = \{e_b, e_d\}$ et non pas $\{m_3\}' = \{e_b\}$.

En revanche, $(\{e_b, e_d\}, \{m_3\})$ en est un, car on a bien $\{m_3\}' = \{e_b, e_d\}$ et $\{e_b, e_d\}' = \{m_3\}$. Au total, le contexte formel définit a 7 concepts qui sont visibles dans l'annexe.

De plus, on a $\{m_2\}' = \{e_b\} = \{m_2, m_3\}'$ donc $\{m_2\}'$ et $\{m_2, m_3\}'$ font partie d'une même classe d'équivalence. Or $\{m_2\} \subseteq \{m_2, m_3\}$ donc $\{m_2\}$ est un générateur de $\{m_2, m_3\}$.

Une fois ces *concepts formels* déterminés, ils sont alors organisés dans un *treillis de concepts* [1]. On définit le *treillis de concepts* d'un contexte $K = (O, A, I)$ comme l'ensemble des concepts de K. Ce dernier peut être visualisé dans un *diagramme de Hasse* comme montré dans l'annexe.

Ce *treillis* va permettre à l'AFC d'extraire les générateurs qui sont nécessaires pour concevoir les règles d'associations,

but final de la méthode [1].

Cependant, l'AFC a un risque de produire un nombre élevé de *concepts* à cause de plusieurs raisons parmi lesquelles :

- *Information manquante* : l'absence de croix peut signifier que l'information est manquante, dans l'exemple 2.1 l'absence de croix entre e_a et m_1 peut signifier que l'on ne sait pas si l'étudiant e_a a validé le module m_1 .
- *Ressemblance des objets ou attributs* : des objets (respectivement des attributs) peuvent fortement se ressembler, dans l'exemple 2.1 les étudiants e_a et e_d ont tous les deux validés les modules m_1 et m_4 . Dans ce cas, on effectue, à quelques détails près, deux fois les mêmes calculs.
- *Données mal réparti* : le jeu de donnée n'est pas forcément uniformément réparti et peut être creux.

Pour toutes ces raisons, l'AFC génère plus de *concepts* que si les informations étaient connues et de fait plus de *règles d'associations* que nécessaire.

Afin d'éviter ces calculs inutiles, on décide de *pré traiter* les données pour *lisser* cette *variabilité*.

Exemple 2.4: En considérant la table I des contextes formels, un lissage des données recommande la suppression de la croix qui lie l'étudiant e_a et le module m_4 et mettre une croix liant l'étudiant e_a au module m_5 . Cela donne alors la table de contextes suivant :

	m_1	m_2	m_3	m_4	m_5
e_a	X				X
e_b		X	X		
e_c	X			X	X
e_d			X		
e_e	X				X

TABLE II
TABLEAU UNAIRE APRÈS LISSAGE

Ce lissage réduit le nombre de concept de 8 à 6. Les concepts sont visibles dans l'annexe.

Dans la suite, les *objets* se réfèrent aux *utilisateurs* et les *attributs* aux *items*.

III. ETAT DE L'ART :

On présente ici quelques possibilités pour *lisser* cette *variabilité*, les méthodes de *clustering*.

Une méthode dite de *clustering* est une technique *non supervisée* de partition d'un ensemble de données en un ensemble de classes visant à maximiser les similarités *intra classes* tout en minimisant les *similarités* inter classes [6] .

On présente dans cette sous-section deux catégories de méthodes, les *clusterings classiques* et les *clusterings hiérarchiques*.

A. Méthodes de clustering classique :

Le *clustering* dit *classique* est une méthode de partitionnement, c'est-à-dire qui décompose un ensemble de N objets en k *clusters* pour optimiser une fonction critère donnée. Ces méthodes se représentent graphiquement avec un centre de gravité appelé *centroïde* [12].

1) K-Means:

On présente la méthode des *K-Means* [5] consistant à positionner k *centroïdes* puis à regrouper chaque points avec le *centroïde* le plus proche. Ce regroupement itératif se poursuit jusqu'à ce que la fonction critère converge, c'est à dire que l'erreur quadratique converge.

Toutefois, en plus d'être très sensible au bruit et aux valeurs aberrantes, cette méthode dépend fortement de l'initialisation, donc de la fiabilité de l'utilisateur à fixer le bon nombre de *clusters*. On peut citer d'autres méthodes comme le *Fuzzy K-Means clustering* et le *K-Medoids* [5].

2) Fuzzy K-Means clustering:

Le *Fuzzy K-Means clustering* [6] associe à chaque point un degré d'appartenance aux *clusters* et ainsi n'appartient pas complètement à un seul *cluster*. Soit un point x , note $uk(x)$ le degré du point x d'appartenir au k^e *cluster*. L'algorithme consiste à :

- Fixer un nombre k de *cluster*
- Attribuer aléatoirement à chaque point des coefficients d'appartenances aux *clusters*
- Répéter jusqu'à ce que l'algorithme converge, c'est à dire que l'écart des coefficients entre deux itérations soit inférieur à ϵ

Le *Fuzzy K-Means clustering* reste sensible au bruit et a surtout une complexité algorithmique trop élevée [6].

3) K-Medoids:

Concernant le *K-Medoids* [5] où, contrairement aux *K-Means*, le *cluster* n'est pas représenté par son *centroïde* mais par son *médoïde*, son objet le plus centré (most centrally located object), l'algorithme se décompose ainsi :

- Sélection aléatoire, pour chaque *cluster*, d'objets en tant que *médoïde*
- Pour chaque objet non sélectionné, regrouper avec le *médoïde* auquel il est le plus similaire
- Remplacer itérativement un *médoïde* par un objet non *médoïde* améliorant la fonction coût.
- Répéter le processus jusqu'à convergence.

Cette méthode a l'avantage d'être moins sensible au bruit et aux valeurs aberrantes mais est très coûteux puisqu'il compare chaque *médoïde* à tous le reste des données, l'équipe

a donc décidé de ne pas poursuivre cette piste [5].

B. Clustering hiérarchique

Le clustering hiérarchique génère une arborescence de *clusters* au travers d'une structure appelé dendrogramme que l'on représente ci-dessous. Cette dernière peut se lire de manière ascendant ou descendante, on propose ici la deuxième version.

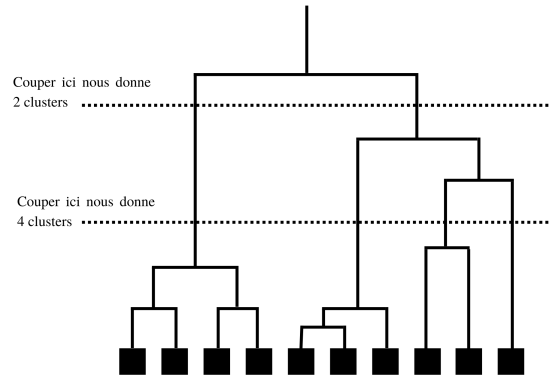


FIGURE 1. Exemple de dendrogramme

Toutefois, ces méthodes ont pour inconvénients de pouvoir présenter des erreurs irrémédiables. En effet, en cas de mauvais choix de métrique, un *cluster* peut être porteur d'une erreur et alors celle-ci est transmise à tous les *clusters* sous-jacents qui ne peuvent la corriger [12].

1) Chameleon:

Les *clusters* sont représentés par leurs nombres de points bien dispersés dans un seul *centroïde*. Ainsi, les représentations sont réduites aux *centroïdes* et ces constantes [13]. L'algorithme se décompose en trois étapes :

- Initialisation avec un graphe contenant des arrêtes entre chaque sommets et ses k voisins les plus proches.
- Partition du graphe en sous graphe non connectés.
- Chaque sous graphe est traité comme un sous *cluster* et un algorithme hiérarchique combine les deux *clusters* les plus similaires.

2) Balanced iterative reducing and clustering using hierarchies, BIRCH:

Cette méthode utilise une structure d'arbre pour compresser d'avantages les données dans de petit *sous-cluster*. Chaque *sous-cluster* étant représenté par une feuille et chaque noeud par un ensemble de sous *clusters* [14].

Cette séparation se fait selon deux critères, d'abord le nombre d'enfant par noeud non terminal, puis le diamètre

maximal de chaque *cluster*, c'est à dire de chaque feuille.

BIRCH est assez rapide, mais est très sensible à l'ordre des données. Cet algorithme à aussi du mal avec les *clusters non-circulaires*, au sens topologique du terme, et de tailles variables car il repose sur leurs diamètres pour contrôler les limites d'un cluster [14].

C. Système de recommandation, filtrage collaboratif :

Les *systèmes de recommandation* sont des algorithmes utilisés dans de nombreuses applications telles que les réseaux sociaux, les sites d'achats, les plateformes de divertissement type Youtube, Netflix etc. Ces méthodes exploitent un ensemble d'informations pour en prédire d'autres, plus précisément pour recommander à un utilisateur, un produit susceptible de lui plaire [4].

En guise d'exemple, on s'intéresse au cas Netflix. Soit un jeu de données représentant les préférences d'un ensemble d'utilisateurs concernant un ensemble de séries. L'algorithme de *recommandation* vise à prédire, pour un utilisateur donné, quel contenu pourrait lui plaire. On schématise cela avec des données *unaires* comme dans la figure 2.

Exemple 3.1: Dans cet exemple, la présence de croix signifie que l'utilisateur a aimé le film ou la série en question et l'absence de croix signifie que l'on ne connaît pas l'avis de l'utilisateur concernant ce film ou cette série.











					
	X		X	X	X
			X	X	
	X	X			
				X	
	X	X	X	X	X

FIGURE 2. Tableau unaire avant le filtrage collaboratif

On retrouve deux approches, soit une comparaison d'utilisateurs, soit une comparaison d'items [4].

Soit un *utilisateur* u et un *item* i , l'algorithme se décompose en trois étapes :

- *Calcul de similarité* :
- *Approche utilisateur* : On associe à chaque utilisateur $u' \neq u$, un réel traduisant la ressemblance entre ces deux utilisateurs. Plus ce réel est haut, plus les utilisateurs sont proches.

— *Approche item* : On adapte la méthode précédente en permutant utilisateur par item.

— *Sélection des voisins* : On restreint l'ensemble des utilisateurs/items à ceux jugés suffisamment ressemblant.

— *Calcul de prédiction* : A partir des voisins retenus, on estime l'avis qu'aurait l'utilisateur u sur l'item i .

On obtient ainsi une prédiction entre l'utilisateur u et l'item i . Cette prédiction est une valeur entre 0 et 1.

Exemple 3.2: On reprend la figure 2, on note les utilisateurs $\{u_1, u_2, u_3, u_4, u_5\}$ et les films/séries $\{i_1, i_2, i_3, i_4, i_5\}$. On propose ici une version du filtrage collaboratif simplifié, avec un seul voisin. On présente d'abord l'approche utilisateur puis l'approche item.

On cherche à répondre à la question suivante, est-ce que le premier utilisateur, u_1 peut aimer le deuxième film, i_2 ? u_1 a aimé tous les films sauf i_2 , on regarde donc parmi les autres utilisateurs, lequel a les mêmes avis. On remarque que u_5 a également aimé i_1, i_3, i_4 et i_5 . Ces deux utilisateurs ont donc, en dehors de i_2 , une ressemblance de 100%. Ainsi, l'approche utilisateur prédit que u_1 aura le même avis que u_5 concernant i_2 . Donc le premier utilisateur va aimer le deuxième film, d'où la croix rouge dans la figure 3.

De la même manière, est-ce que le quatrième film/série, i_4 peut plaire au troisième utilisateur, u_3 ? La série ayant le plus de ressemblances avec i_4 est i_3 qui partagent une similarité de 75%. Donc, l'approche item prédit que i_4 sera autant apprécié que i_3 par u_3 , mais ne connaissant pas ce dernier avis, on laisse une absence de croix. Ici, on met un cercle vert dans la figure 3 uniquement pour illustrer.











					
	X	X	X	X	X
			X	X	
	X	X			
				X	
	X	X	X	X	X

FIGURE 3. Tableau unaire après le filtrage collaboratif

D. K Unified Nearest Neighbor, kUNN :

Dans le *filtrage collaboratif* seul une approche est utilisée, que ce soit l'*approche utilisateur* ou l'*approche item*. Cependant, ces deux approches se ressemblent particulièrement.

En effet, comme il a été vu dans la partie III-C, les formules calculant la prédiction avec l'approche utilisateur et l'approche item, qui se retrouvent respectivement dans les équations 1 et 2 de la partie IV, diffèrent seulement par la somme. Dans l'une on fait la somme sur les voisins de l'*utilisateur* et dans l'autre on fait la somme sur les voisins de l'*item*.

D'autre part, ces deux approches prises individuellement donnent à peu près les mêmes résultats [3]. En effet, Verstrepen et Goethals [4] montrent que ces deux approches sont deux modèles incomplets qui peuvent enlever des informations importantes du jeu de données. C'est pour cela qu'ils introduisent le *kUNN*, une méthode combinant ces deux approches.

Comme dit en section III-C, une prédiction est une valeur entre 0 et 1. On compare cette valeur à un seuil fixé au préalable afin de fixer la prédiction à 0 ou à 1. Si cette prédiction est de un, on propose le produit *i* à l'utilisateur *u*.

La recherche de ce seuil sera plus développée dans la section IV-A.

Dans l'optique d'améliorer la précision des prédictions, on combine les approches utilisateur et item [4].

$$s(u, i) = \frac{s_u(u, i) + s_i(u, i)}{\sqrt{c(u)c(i)}}$$

Où :

- $s(u, i)$: Prédiction entre l'utilisateur *u* et l'item *i*
- $s_u(u, i)$: Prédiction via l'approche utilisateur
- $s_i(u, i)$: Prédiction via l'approche item
- $c(u)$: Nombre de 1 fourni par l'utilisateur *u*
- $c(i)$: Nombre de 1 fourni par l'item *i*

On détaille le calcul de $s_u(u, i)$ et $s_i(u, i)$ dans la section IV-A. On divise la somme par $\sqrt{c(u)c(i)}$ pour pondérer par l'absence de renseignement concernant l'utilisateur *u* et/ou l'item *i*. Ainsi, si l'on souhaite évaluer une prédiction entre un utilisateur *u* et un item *i*, dans le cas où *u* ou *i* n'ont fourni aucune donnée à la base, alors on ignore tout d'eux et il est donc difficile de prédire leur avis. Dans ce cas, $c(u) * c(i) = 0$ et on ne calcule pas la prédiction. De même, si l'utilisateur ou l'item n'ont qu'une note, on peut estimer ne pas les connaître suffisamment pour émettre une prédiction. Dans ce cas, on impose une condition tel que si l'utilisateur (respectivement l'item) n'ont pas noté (respectivement été noté) au moins un certain nombre de fois, alors on ne calcule pas la prédiction [4].

On peut désormais développer l'implémentation de ces algorithmes.

IV. IMPLÉMENTATION :

A. Calcul de prédiction :

On étudie désormais le calcul de prédiction selon l'approche *utilisateur*, l'approche *item* s'en déduit aisément en modifiant

les noms des variables et des méthodes.

On commence par la méthode `userSimilarity(int u, int v)` renvoyant le calcul de similarité entre les *utilisateurs* *u* et *v*. Le jeu de données étant unaire, on l'évalue à partir de la similarité cosinus :

$$\cos(u, v) = \frac{\sum_{j \in I} R_{uj} R_{vj}}{\sqrt{c(u)c(v)}}$$

Avec :

- $\cos(u, v)$: Similarité cosinus entre les *utilisateurs* *u* et *v*.
- $R_{uj} \in \{0, 1\}$: Coefficient entre l'*utilisateur* *u* et l'item *j*.
- $c(u) \in \mathbb{N}$: Nombre de coefficient à 1 de l'*utilisateur* *u*.

On définit ensuite `buildUserSimilarities(int u)` permettant d'obtenir toutes les similarités entre l'*utilisateur* *u* et les autres *utilisateurs* en rappelant `userSimilarity` en boucle. On obtient ainsi un vecteur de similarités entre *u* et les autres *utilisateurs*.

On sélectionne dans ce vecteur les *k* voisins les plus proches via la méthode `selectNeighbor`, c'est-à-dire les *k* *utilisateurs* aux plus hautes similarités.

On définit enfin la méthode `userPredict(int u, int i, int k)` renvoyant la prédiction du point de vue *utilisateur* entre l'*utilisateur* *u* et l'item *i* à partir des *k* voisins les plus proches de *u*. Ce calcul correspond à la formule suivante :

$$s_u(u, i) = \sum_{v \in KNN(u)} \frac{R_{vi}}{\sqrt{c(i)}} \text{sim}(u, v) \quad (1)$$

Où :

- $s_u(u, i)$: Prédiction du point de vue *utilisateur* entre l'*utilisateur* *u* et l'item *i*.
- $KNN(u)$: Ensemble des *k* voisins les plus proches de l'*utilisateur* *u*.
- $c(i) \in \mathbb{N}$: Nombre de coefficients à 1 de l'item *i*.

De manière très similaire, on obtient la prédiction du point de vue *item* en utilisant la formule :

$$s_i(u, i) = \sum_{j \in KNN(i)} \frac{R_{uj}}{\sqrt{c(u)}} \text{sim}(u, v) \quad (2)$$

On obtient enfin la prédiction finale en calculant :

$$s(u, i) = \frac{s_u(u, i) + s_i(u, i)}{\sqrt{c(u)c(i)}} \quad (3)$$

Cette valeur est un double supérieur ou égal à 0 et doit donc ensuite être comparé à un seuil pour être fixé à 0 ou à 1.

On note *max* la valeur prédite maximale.

On choisit ainsi pour seuil un intervalle en dehors duquel on va prédire les valeurs. Pour cela, on fixe une valeur $n \in \mathbb{N}$ tel que :

- Si la valeur prédite est inférieure ou égale à $\frac{max}{n}$, on la fixe à 0

- Si la valeur prédite est supérieur ou égale à $\frac{(n-1)*max}{n}$ on la fixe à 1.
- Sinon, la valeur fixée est celle présente dans la matrice initiale.

Ce choix de formule s'explique par la volonté d'accepter une prédiction à partir d'un certains degré de certitude. Ainsi, plus n est grand, moins on modifie notre jeu de données.

V. RÉSULTATS :

Les prédictions ont été réalisé grâce au kUNN sur un jeu de donnée pharmacologique. Ainsi, l'équipe a récupéré différents résultats en fixant des valeurs n , tels que $n \in [2, 6]$. Afin d'analyser ces résultats, différentes mesures de qualités ont été réalisé. [3].

A. Précision

La première mesure à réaliser est celle de l'écart entre les valeurs du jeu de données initial et celles des jeux de données prédits. En effet, on cherche à produire un nombre minimal de *concepts*, tout en modifiant le moins que possible le jeu de données initial. Ainsi, la précision indique le pourcentage de données qui n'ont pas été modifiées par les calculs de prédiction. La figure 4 donne la précision des jeux de données en fonction du seuil choisi.

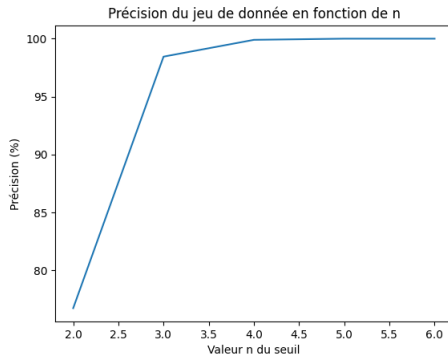


FIGURE 4. Métrique de précision du kUNN en fonction du seuil

B. Nombre de concepts

Afin d'accélérer l'AFC, on souhaite générer le moins de concepts possible. On indique donc dans la figure 5 le nombre de concepts générés pour chaque valeur de seuil.

C. Hit rate

On définit le *hit set* Hu d'un utilisateur u comme l'ensemble des *items* que celui-ci a noté. Ici, il s'agit d'une liste contenant tous les *items* prédits à 1 pour cet *utilisateur*. De plus, on définit U_t comme la liste d'*utilisateur* qui ont noté au moins un *item*, tel que $U_t = \{u \in U | |Hu| > 0\}$ [3]. On définit également le hit rate à 10, $HR@10$, tel que :

$$HR@10 = \frac{1}{|U_t|} \sum_{u \in U_t} |Hu \cap top10(u)|$$

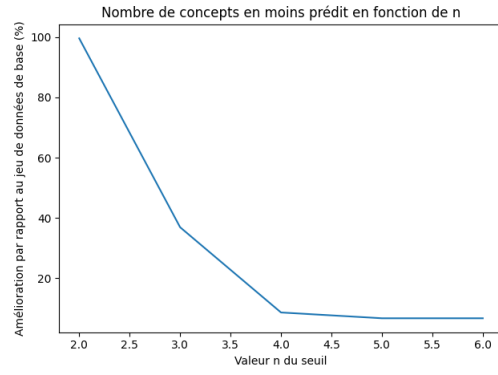


FIGURE 5. Pourcentage de concepts générés en moins par rapport au jeu de donnée de base du kUNN en fonction du seuil

avec $top10(u)$ les 10 *items* les mieux notés par u . Par conséquent, $HR@10$ donne le pourcentage d'*utilisateurs* pour lesquels une recommandation est dans leur 10 premières recommandations favorites [3]. La figure 6 montre $HR@10$ pour chaque valeur de seuil.

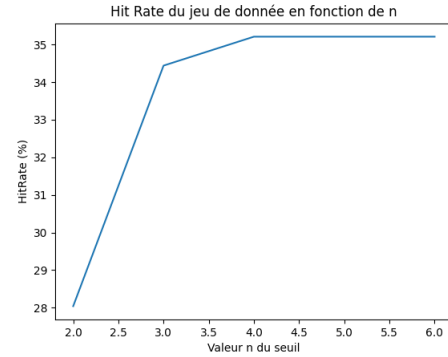


FIGURE 6. Métrique $HR@10$ du kUNN en fonction du seuil

D. Average reciprocal hit rate

La métrique Average reciprocal hit rate à 10 ($ARHR@10$) prend en compte le rang de la préférence de l'*utilisateur* dans son top10 [3], dénoté comme $r(hu)$. $ARHR@10$ est donné par la formule :

$$ARHR@10 = \frac{1}{|U_t|} \sum_{u \in U_t} |Hu \cap top10(u)| \cdot \frac{1}{r(hu)}$$

Ainsi, la figure 7 montre $ARHR@10$ pour chaque valeur de seuil.

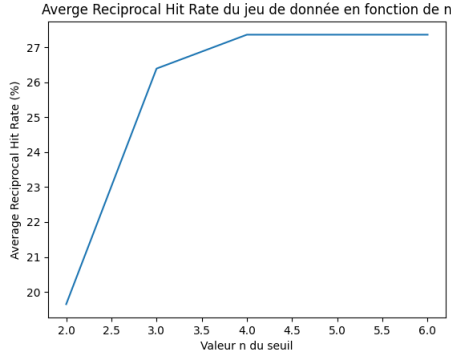


FIGURE 7. Métrique ARHR@10 du kUNN en fonction du seuil z

E. All missing as negative

La métrique All missing as negative (*AMAN*) [3] permet de mesurer le pourcentage de préférences manquantes qui ont été évaluées comme un 0 dans la recommandation. Celle-ci prend en compte le rang des recommandations pour une liste d'*utilisateurs* et le nombre d'*items* du jeu de données $|I|$. *AMAN* est donnée par la formule :

$$AMAN = \frac{1}{|U|} \sum_{u \in U} \frac{|I| - r(hu)}{|I| - 1}$$

La figure 7 montre le pourcentage de valeurs manquantes évaluées comme négatives.

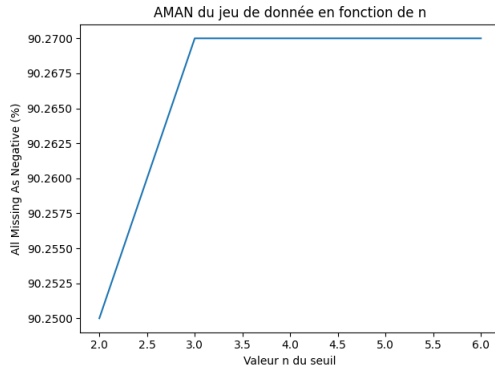


FIGURE 8. Métrique *AMAN* du kUNN en fonction du seuil

VI. CONCLUSION :

Finalement, les résultats montrent que la solution optimale qui permet d'obtenir le moins de concepts possibles, donc de *règles d'association*, tout en gardant de bonnes métriques de qualité est de fixer la valeur du seuil à $n=3$. Cela permet de garder une précision de 98.45% et de réduire le nombre de concepts générés de 36.93%, ce qui permet de diminuer de 75% le nombre de règles générées par notre jeu de donnée.

Pour assurer la qualité des jeux de données, on a d'abord mesuré le *hit rate*. Ce dernier vaut, pour $n=3$, 34.44%, contre 35.21% pour les valeurs supérieurs. On montre ainsi qu'en

moyenne une recommandation sur trois est pertinente pour un *utilisateur*, lorsque le seuil est supérieur à 3. Malgré cela, le caractère binaire du jeu de donnée de départ et sa forte lacunarité fait que plus de 90% des données manquantes sont considérées comme négatives

Initialement avec ce jeu de données, l'AFC produisait près de 4,5 millions de règles. Désormais, on restreint ce nombre à 2,7 millions soit une réduction de 40%. Mais bien que les résultats soient prometteurs, une version parallèle du système pourrait permettre d'avoir une meilleure complexité temporelle. En effet, les différents calculs tels que la prédiction ou encore la sélection des voisins peuvent se faire en parallèle pour chaque *utilisateur* et *item*.

REMERCIEMENTS :

On souhaite essentiellement remercier Mickael Wajnberg pour son implication dans notre projet et pour son aide précieuse. Bien qu'à distance, on a pu pleinement apprécier ce projet de recherche dont le sujet était très intéressant. Durant ces quatre derniers mois, il n'a pas manqué une seule réunion et a pris le temps à chaque fois d'expliquer en détails les points qui posaient problème au groupe, sans compter les nombreux échanges par messages.

On le remercie également de nous avoir présenté à d'autres collègues à lui avec qui il a été enrichissant de discuter et de nous avoir invité à une conférence portant sur le Machine Learning qui a été, tout comme ce projet, l'occasion d'élargir notre culture de la Big Data.

RÉFÉRENCES

- [1] Mickael Wajnberg. Analyse relationnelle de concepts : une méthode polyvalente pour l'extraction de connaissance. Informatique [cs]. Université du Québec à Montréal ; Université de Lorraine, 2020. Français. fffnt : 2020LORR0136ff. ffftel-03042085f
- [2] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3) :37–37, 1996.
- [3] Koen Verstrepen and Bart Goethals, Unifying Nearest Neighbors Collaborative Filtering, 2014.
- [4] Koen Verstrepen, Bart Goethals, Boris Cule, Kanishka Bhaduri, Collaborative Filtering for Binary, Positive-only Data, 2017.
- [5] Preeti Arora, Dr. Deepali and Shipra Varshney, Analysis of K-Means and K-Medoids Algorithm For Big Data, 2015.
- [6] Deepti Sisodia, Lokesh Singh, Sheetal Sisodia and Khushboo Saxena, Clustering Techniques : A Brief Survey of Different Clustering Algorithms, 2012.
- [7] Dan Pelleg and Andrew Moore, X-Means : Extending K-Means with efficient estimation of the number of clusters, 2002.
- [8] P. Valtchev et al. Formal Concept Analysis for Knowledge Discovery and Data Mining : The New Challenges. In Proc. of ICFA 2004, volume 2961 of LNCS, pages 352–371, 2004.
- [9] Marianne Huchard. Formal concept analysis, a framework for knowledge structuring and exploration. applications to service directories and product lines. 2019.
- [10] Perna Kapoor, Prem Kumar Singh, and Aswani Kumar Cherukuri. Crime data set analysis using formal concept analysis (fca) : A survey. In Advances in Data Sciences, Security and Applications, pages 15–31. Springer, 2020.
- [11] Pranav Shetty and Suraj Singh , Hierarchical Clustering : A Survey, 2021.

- [12] Deepti Sisodia Lokesh Singh Sheetal Sisodia Khushboo Saxena, A Brief Survey of Different Clustering Algorithms, 2012.
- [13] George Karypis , Eui-Hong (Sam) Han , Vipin Kumar, Chameleon : Hierarchical Clustering Using Dynamic Modeling, Computer, v.32 n.8, p.68-75, August 1999 [doi:10.1109/2.781637]
- [14] He Zengyou , Xu Xiaofei , Deng Shengchun, Squeezer : an efficient algorithm for clustering categorical data, Journal of Computer Science and Technology, v.17 n.5, p.611-624, May 2002

ANNEXE

A. Tracé de toutes les métriques

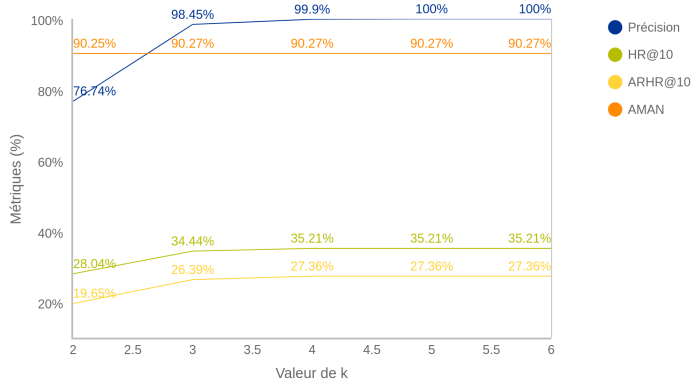


FIGURE 9. Métriques de qualité du kUNN en fonction du seuil

B. Diagrammes de Hasse

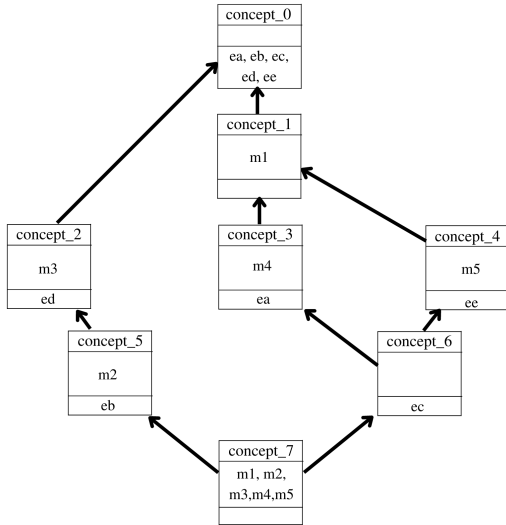


FIGURE 10. Treillis de concepts du contexte représenté par la table I

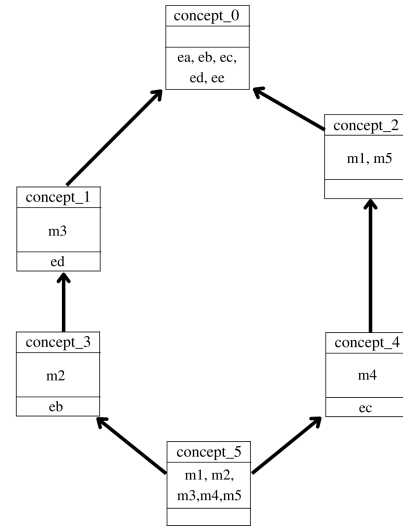


FIGURE 11. Treillis de concepts du contexte représenté par la table I après lissage des données