

## Projet : Système de Recommandation Client-Serveur en C

Ce projet a pour but de développer un **système de recommandation de type client-serveur** écrit en **langage C**, capable de proposer à un utilisateur des articles ou éléments susceptibles de lui plaire, à partir de données de notations existantes.

Le projet repose sur une **architecture distribuée** dans laquelle :

- Le **serveur** centralise les données, applique les algorithmes de recommandation et retourne les résultats.
- Le **client** envoie une requête contenant un identifiant utilisateur, l'algorithme souhaité et le nombre de recommandations désirées.

La mise en place d'un système de recommandation passe par trois étapes importantes qui sont : le prétraitement du jeu de données, la construction d'un modèle pour le système de recommandation et enfin l'évaluation de ce modèle.

### Le client :

- Envoie une requête contenant :
  - L'identifiant de l'utilisateur
  - L'algorithme de recommandation souhaité (KNN, MF, ou GRAPH)
  - Le nombre de recommandations souhaitées

### Le serveur :

- Accepte plusieurs clients (via fork() ou pthread)
- Interprète la requête du client
- Utilise l'algorithme demandé (sous forme de **bibliothèque statique ou dynamique**)
- Envoie la liste de recommandations formatée vers le client

### Contenu des données

Les transactions sont stockées dans un fichier texte ratings.txt. Chaque ligne suit le format suivant :

<user\_id> <item\_id> <category\_id> <rating> <timestep>

Exemple :

249 1087 21 4.3 969215645  
83 1134 34 3.5 1000121189  
7 1231 43 2.4 99855800  
64 1042 32 3.0 952138789

### Contraintes techniques

- Implémentez la communication via : **Sockets TCP/IP**
- Le serveur peut gérer plusieurs clients simultanément
- Implémentez trois algorithmes de recommandation distincts :
  1. **KNN (K-Nearest Neighbors)** : recommandations basées sur les utilisateurs similaires
  2. **MF (Matrix Factorization)** : scores estimés via la factorisation matricielle
  3. **GRAPH** : recommandations par graphe de co-notations
- Ces algorithmes doivent être :
  1. Soit compilés en **bibliothèque statique (.a)**
  2. Soit en **bibliothèque dynamique (.so)**



## Travaux Pratiques : Systèmes de recommandation

NB : Tous les algorithmes doivent etres implémentés en langage C

La mise en place d'un système de recommandation passe par trois étapes importantes qui sont : le prétraitement du jeu de données, la construction d'un modèle pour le système de recommandation et enfin l'évaluation de ce modèle.

### 1 Traitement du jeu de données

Les données sont stockées dans un fichier .txt sous formes de transactions. Une transaction comprend l'id de l'utilisateur, de l'item, de la catégorie de l'item, la note que l'utilisateur a attribué à l'item et enfin le nommbre de seconde qui se sont écoulées depuis le 01 janvier 1970 jusqu'au moment de la transaction. À partir du fichier en entrée, vous devez écrire les fonctions suivantes :

- une fonction pour recuperer les transactions d'une période particulière (exemple : janvier 2000 à février 2000) et les mettre dans un nouveau fichier.
- une fonction pour supprimer toutes les transactions des utilisateurs et des items qui apparaissent un nombre de fois inférieur à une certaine valeur minU (pour les utilisateurs) et minI (Pour les items).
- Ayant déjà récupéré un ensemble de transactions pour l'apprentissage dans un fichier et un autre ensemble pour le test, écrire une fonction qui va nettoyer les données de test en supprimant dans le fichier de test, toutes les transactions des utilisateurs et toutes les transactions des items qui n'apparaissent pas dans les données de l'apprentissage.

### 2 Construction du modèle de recommandation

Ici, il s'agit d'utiliser les données déjà prétraitées pour construire un modèle de recommandation à partir d'un algorithme. Les algorithmes que vous devez utilisés sont les K plus proches voisins (KNN), la factorisation matricielle (MF) et les Graphes. Le fonctionnement de ces algorithmes est expliqué en détail dans la suite.

#### 2.1 K plus proches voisins (K Nearest Neighbors - KNN)

C'est un modèle basé sur le voisinage. De façon générale, il se sert des similarités entre utilisateurs pour faire des recommandations d'items à un utilisateur à partir d'une matrice de notes des utilisateurs aux items. De façon spécifique, les étapes à suivre pour prédire la note  $\hat{r}_{ui}$  qu'un utilisateur  $u$  va attribuer à un item  $i$  qu'il n'a pas encore noté sont les suivantes :

- Calculer la similarité entre l'utilisateur  $u$  et tous les autres utilisateurs de la base de données :

Il existe deux principales fonctions de calcul de similarité à savoir **Cosinus** et **Pearson**. La fonction Pearson est mieux adaptée pour ce modèle car elle prend en considération le fait que les utilisateurs n'ont pas la même façon de noter les items. Certains utilisateurs sont très généreux dans la manière de noter tandis que d'autres ne le sont pas; raison pour laquelle, dans la fonction de Pearson, on soustrait à chaque note  $r_{ui}$ , la moyenne des notes de l'utilisateur  $u$ , ceci pour enlever ce biais.

Soit  $I_u$  l'ensemble des indices des items que  $u$  a noté et  $I_u \cap I_v$  l'ensemble des indices des items que  $u$  et  $v$  ont noté ensemble. Le calcul de la moyenne des notes de  $u$  se fait à travers l'équation 1 :

$$\mu_u = \frac{\sum_{i \in I_u} r_{ui}}{|I_u|} \quad (1)$$

Par la suite, le coefficient de corrélation de Pearson entre deux utilisateurs  $u$  et  $v$  est donné par l'équation 2 :

$$Sim(u, v) = Pearson(u, v) = \frac{\sum_{i \in I_u \cap I_v} (r_{ui} - \mu_u) \cdot (r_{vi} - \mu_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{ui} - \mu_u)^2} \cdot \sqrt{\sum_{i \in I_u \cap I_v} (r_{vi} - \mu_v)^2}} \quad (2)$$

Avec  $r_{ui}$  et  $r_{vi}$  étant respectivement les notes que les utilisateurs  $u$  et  $v$  ont donné à l'item  $i$ .

- Retenir le top-k des utilisateurs les plus similaires à  $u$  et qui ont déjà noté l'item  $i$  (item dont on veut prédire la note que  $u$  lui accordera) :

Ces utilisateurs constituent l'ensemble  $P_u(i)$

- Calculer la prédiction  $\hat{r}_{ui}$  de la note que  $u$  va attribuer à un item  $i$  à travers l'équation 3 :

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in P_u(i)} Sim(u, v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in P_u(i)} |Sim(u, v)|} \quad (3)$$

user-ID/item-ID	1	2	3	4	5	6	Note Moy	Pearson(i,3)
1	7	6	7	4	5	4	5.5	0.894
2	6	7	?	4	3	4	4.8	0.939
3	?	3	3	1	1	?	2	1.0
4	1	2	2	3	3	4	2.5	-1.0
5	1	?	1	2	3	3	2	-0.817

TABLE 1 – Calcul de la similarité utilisateur-utilisateur entre 3 et les autres.

Considérons l'exemple d'application du tableau 1. Il présente les notes que 5 utilisateurs ont donné à 6 produits. Les notes sont inscrites dans la plage de 1 à 7 et les cases avec le point d'interrogation sont celles pour lesquelles l'utilisateur n'a pas encore acheté le produit. La colonne *Note Moy* contient la moyenne des notes de chaque utilisateur. La dernière colonne comporte la similarité de Pearson entre tous les utilisateurs et l'utilisateur 3.

Déroulons le calcul de la similarité de Pearson entre l'utilisateur 1 et l'utilisateur 3 :

$$Pearson(1, 3) = \frac{(6 - 5.5) * (3 - 2) + (7 - 5.5) * (3 - 2) + (4 - 5.5) * (1 - 2) + (5 - 5.5) * (1 - 2)}{\sqrt{0.5^2 + 1.5^2 + (-1.5)^2 + (-0.5)^2} * \sqrt{1^2 + 1^2 + (-1)^2 + (-1)^2}} = 0.894$$

Nous obtenons bien les résultats qui sont dans le tableau. A partir du tableau, nous constatons que le top-2 des utilisateurs les plus similaires à l'utilisateur 3 est l'ensemble  $\{1, 2\}$  car les similarités sont les plus grandes.

Nous allons donc utiliser ces deux utilisateurs pour calculer les prédictions des notes de l'utilisateur 3 pour les produits 1 et 6 :

$$r_{31} = 2 + \frac{1.5 * 0.894 + 1.2 * 0.939}{0.894 + 0.939} = 3.35$$

$$r_{36} = 2 + \frac{-1.5 * 0.894 - 0.8 * 0.939}{0.894 + 0.939} = 0.86$$

Nous remarquons que le résultat de la prédiction  $r_{36} = 0.86$  qui est inférieur à 1 (la plus petite note de la plage). Dans ce genre de situations, on peut ramener la note à la plus petite de la plage (1).

Les fonctions principales que vous devez implémenter sont les suivantes :

- **Pearson(train-data)** : prend en entrée l'ensemble des transactions pour l'entraînement et retourne une matrice carrée de similarité entre les utilisateurs.
- **Predict(u,i)** : prend en entrée l'identifiant d'un utilisateur et d'un item et donne en sortie la note que l'utilisateur va attribuer à l'item.
- **Predict-all(test-data)** : prend en entrée l'ensemble des transactions du jeu de données de test et donne en sortie l'ensemble de toutes les notes prédites. À partir de ces notes, vous pourrez évaluer le modèle de KNN construit.

## 2.2 Factorisation matricielle

La factorisation matricielle est un modèle utilisé dans les systèmes de recommandation pour la prédiction des notes que les utilisateurs vont accorder aux items. Initialement on dispose d'une matrice de notes  $R \in \mathbb{R}^{m*n}$  concernant  $m$  utilisateurs et  $n$  items.

L'idée derrière la factorisation matricielle est d'apprendre un modèle de facteurs latents d'utilisateurs  $U \in \mathbb{R}^{m*k}$  et d'items  $V \in \mathbb{R}^{n*k}$  de sorte que la reconstruction  $R' = U * V^T$  entre les deux matrices estime au mieux la matrice initiale  $R$  et permette par conséquent de prédire les notes initialement inconnues de la matrice de départ.  $k$  est le nombre de facteurs latents par utilisateur et par item, c'est donc un paramètre de ce modèle.

Pour la suite, on va noter  $u_i$  le vecteur représentant les  $k$  facteurs latents de l'utilisateur  $i$  et  $v_j$  le vecteur représentant ceux de l'item  $j$ .

À partir de ce qui a été dit plus haut, nous pouvons donc comprendre que le modèle de factorisation matricielle repose sur les valeurs des matrices  $U$  et  $V$ . Il est donc important de mettre sur pied une astuce pour trouver les bonnes valeurs (facteurs latents) de ces deux matrices. Pour le faire, ce modèle utilise un algorithme d'optimisation appelé la **descente de gradient stochastique**[2].

Ce modèle a pour but de trouver les facteurs latents qui minimisent la fonction 4 :

$$\min J = \frac{1}{2} \sum_{(i,j) \in S} (r_{ij} - \hat{r}_{ij})^2 + \frac{\lambda}{2} \left( \sum_{i=1}^m \|u_i\|^2 + \sum_{j=1}^n \|v_j\|^2 + \sum_{i=1}^m o_i^2 + \sum_{j=1}^n p_j^2 \right) \quad (4)$$

avec

$$\hat{r}_{ij} = \sum_{s \in \{1, \dots, k\}} (u_{is} \cdot v_{js}) + o_i + p_j$$

Cette fonction  $J$  est composée de 2 grandes parties (la deuxième commence avec  $\frac{\lambda}{2}$ ). La seconde partie est elle aussi composée de 4 parties (dont chacune est séparée de l'autre par le signe d'addition). Dans la suite, nous allons expliquer chacune de ces différentes parties l'une après l'autre :

- $\frac{1}{2} \sum_{(i,j) \in S} (r_{ij} - \hat{r}_{ij})^2$  : ceci représente la somme des carrés des erreurs entre la note réelle  $r_{ij}$  et celle prédite  $\hat{r}_{ij}$  par la factorisation matricielle, ceci pour toutes les notes observées de la matrice initiale.  $S$  est l'ensemble des couples  $(i, j)$  de notes observées dans la matrice  $R$  entre l'utilisateur  $i$  et l'item  $j$ .
- Le facteur de régularisation **lambda** ( $\lambda$ ) permet d'éviter le sur-apprentissage lors de l'entraînement pour obtenir les bons facteurs latents. En effet, dans la vie réelle la matrice de note est souvent éparse, ce qui augmente le risque de sur-apprentissage de l'algorithme, c'est donc pour éviter cela qu'on ajoute ce facteur lambda. Ce dernier prend en paramètres 4 éléments que nous allons détailler dans la suite.
- $\frac{\lambda}{2} (\sum_{i \in \{1, \dots, m\}} \|u_i\|^2 + \sum_{j \in \{1, \dots, n\}} \|v_j\|^2)$  : ceci représente les deux premiers éléments qui sont pris en paramètre. Il s'agit de  $u_i$  et  $v_j$  qui sont respectivement les vecteurs des facteurs latents de l'utilisateur  $i$  et de l'item  $j$ .
- $\frac{\lambda}{2} (\sum_{i \in \{1, \dots, m\}} o_i^2 + \sum_{j \in \{1, \dots, n\}} p_j^2)$  : ceci représente les deux derniers éléments qui sont pris en paramètre. Il s'agit des termes  $o_i$  et  $p_j$  qui symbolisent respectivement le biais de l'utilisateur  $i$  et de l'item  $j$ . Le biais utilisateur représente une valeur ajoutée à la prédiction, caractérisant la tendance globale d'un utilisateur à attribuer des évaluations plus élevées ou plus basses que la moyenne. De manière similaire, le biais item constitue un ajustement à la prédiction, reflétant la tendance globale d'un item à recevoir des évaluations plus élevées ou plus basses que la moyenne.

$\sum_{s \in \{1, \dots, k\}} (u_{is} \cdot v_{js}) + o_i + p_j$  : ceci représente la note prédictive  $\hat{r}_{ij}$  entre  $i$  et  $j$ . C'est la somme de 3 éléments à savoir : le produit scalaire entre les vecteurs  $u_i$  et  $v_j$ , le biais de l'utilisateur concerné ( $o_i$ ) et celui de l'item concerné ( $p_j$ ).

Pour résumer, l'algorithme de descente de gradient ajuste les quatre variables suivantes, notamment  $u_{is}$  (le  $s$ -ème facteur latent de l'utilisateur  $i$ ),  $v_{js}$  (le  $s$ -ème facteur latent de l'item  $j$ ),  $o_i$  et  $p_j$ , jusqu'à atteindre la convergence. La convergence se produit lorsque ces variables atteignent des valeurs stables et que la variation entre les itérations successives devient suffisamment faible.

Pour le calcul de la formule de mise à jour de chacun de ces paramètres à l'aide de la descente de gradient stochastique, il faut dans un premier temps calculer l'expression du gradient de la fonction  $J$  par rapport à la variable concernée.

— Pour  $u_{is}$  : après avoir calculé la dérivée de la fonction  $J$  par rapport à  $u_{is}$ , on a,

$$\frac{\partial J}{\partial u_{is}} = -v_{js} \cdot e_{ij} + \lambda \cdot u_{is} \quad (5)$$

avec

$$e_{ij} = r_{ij} - \hat{r}_{ij}$$

Ensuite, nous pouvons calculer la formule de mise à jour de  $u_{is}$  par le développement suivant :

$$u_{is} = u_{is} + \alpha \cdot d_{u_{is}} \text{ avec } \alpha \text{ le pas de déplacement et } d_{u_{is}} \text{ la direction de descente or}$$

$$d_{u_{is}} = -\frac{\partial J}{\partial u_{is}} = v_{js} \cdot e_{ij} - \lambda \cdot u_{is} \text{ donc,}$$

$$u_{is} = u_{is} + \alpha \cdot (v_{js} \cdot e_{ij} - \lambda \cdot u_{is}) \quad (6)$$

— De même, pour  $v_{js}$ , on a  $\frac{\partial J}{\partial v_{js}} = -u_{is} \cdot e_{ij} + \lambda \cdot v_{js}$  et donc,

$$v_{js} = v_{js} + \alpha \cdot (u_{is} \cdot e_{ij} - \lambda \cdot v_{js}) \quad (7)$$

— Pour  $o_i$  : après avoir calculé la dérivée de la fonction  $J$  par rapport à  $o_i$ , on a,

$$\frac{\partial J}{\partial o_i} = -e_{ij} + \lambda \cdot o_i \quad (8)$$

Ensuite, nous pouvons calculer la formule de mise à jour de  $o_i$  par le développement suivant :

$$o_i = o_i + \alpha \cdot d_{o_i} \text{ avec } d_{o_i} \text{ la direction de descente or } d_{o_i} = -\frac{\partial J}{\partial o_i} = e_{ij} - \lambda \cdot o_i \text{ donc,}$$

$$o_i = o_i + \alpha \cdot (e_{ij} - \lambda \cdot o_i) \quad (9)$$

— De même, pour  $p_j$ , on a  $\frac{\partial J}{\partial p_j} = -e_{ij} + \lambda \cdot p_j$  et donc,

$$p_j = p_j + \alpha \cdot (e_{ij} - \lambda \cdot p_j) \quad (10)$$

Les équations 6, 7, 9 et 10 ci-dessus constituent les formules de mises à jour des différentes variables de l'algorithme de descente de gradient stochastique.

Dans un premier temps, les matrices  $U$  et  $V$  des facteurs latents des utilisateurs et des items, conjointement aux vecteurs  $O$  et  $P$  représentant les biais respectifs des utilisateurs et des items, sont initialisés de manière aléatoire. Par la suite, à l'aide des équations formulées ci-dessus, les différentes variables sont itérativement mises à jour pour chaque note observée  $r_{ij}$  présente dans la matrice initiale. Cette procédure est répétée sur un nombre défini d'*epochs*, suivant un processus d'optimisation visant à converger vers des estimations plus précises des paramètres du modèle. L'algorithme 1 présente la factorisation matricielle.

Considérons la figure 1 : À gauche, nous avons une matrice de notes que les utilisateurs {A,B,C,D} ont attribués aux produits {W,X,Y,Z}. S constitue l'ensemble des notes qui existent actuellement dans

---

**Algorithme 1 : Factorisation matricielle**

---

**Entrées :** matrice de notes  $R$ , ensemble  $S$  de notes  $r_{ij} \neq 0$ ,  $k$  le nombre de facteur latents, paramètres  $\alpha$  et  $\lambda$ , nombre d'epochs  $N$

**Sorties :** Matrice des facteurs latents  $U$  des utilisateurs et  $V$  des items; Vecteur  $O$  des biais des utilisateurs et  $P$  des items

```

1 initialisation aléatoire de  $U$ ,  $V$ ,  $O$  et  $P$  ;
2  $l \leftarrow 0$  ;
3 tant que  $l < N$  faire
4    $\forall r_{ij} \in S$  ;
5     mettre à jour le vecteur  $u_i$  par  $u_{is} = u_{is} + \alpha.(v_{js}.e_{ij} - \lambda.u_{is}) \forall s \in \{1, \dots, k\}$  ;
6     mettre à jour le vecteur  $v_j$  par  $v_{js} = v_{js} + \alpha.(u_{is}.e_{ij} - \lambda.v_{js}) \forall s \in \{1, \dots, k\}$  ;
7     mettre à jour le biais utilisateur  $o_i$  par  $o_i = o_i + \alpha.(e_{ij} - \lambda.o_i)$  ;
8     mettre à jour le biais item  $p_j$  par  $p_j = p_j + \alpha.(e_{ij} - \lambda.p_j)$  ;
9    $l \leftarrow l + 1$ 
10 fin

```

---

		Item			
		W	X	Y	Z
User	A	4.5	2.0		
	B	4.0		3.5	
	C		5.0		2.0
	D		3.5	4.0	1.0

Rating Matrix

		User			
		W	X	Y	Z
Item	A	1.2	0.8		
	B	1.4	0.9		
	C	1.5	1.0		
	D	1.2	0.8		

User Matrix

		Item			
		W	X	Y	Z
User	A	1.5	1.2	1.0	0.8
	B	1.7	0.6	1.1	0.4
	C				
	D				

Item Matrix

FIGURE 1 – Matrice de notes et facteurs latents des utilisateurs et item .

la matrice. Chaque note de  $S$  est représenté par le triplet  $(i, j, r_{ij})$  avec  $i$  étant l'utilisateur,  $j$  l'item et  $r_{ij}$  la note que  $i$  a donné à  $j$ . À droite, nous avons deux matrices, la verticale qui représente les facteurs latents des utilisateurs obtenus après execution de l'algorithme et l'horizontale qui correspond aux facteurs latents des produits. À partir de ces facteurs latents et des biais utilisateurs et items, on peut reconstruire la matrice initiale et avoir les valeurs des notes qui étaient initialement inconnues.

Les principales fonctions à implémenter ici sont :

- **MF(train-data)** : prend en entrée l'ensemble des transactions du jeu de données d'apprentissage ( $S$ ) et donne en sortie une matrice de notes pleines.
- **Predict-all-MF(Full-Matrix, test-data)** : prend en entrée la matrice de notes pleines et l'ensemble des transactions du jeu de données de test et donne en sortie l'ensemble de toutes les notes prédites. À partir de ces notes, vous pourrez évaluer le modèle de MF construit.

### 2.3 Graphes

Les méthodes basées sur le voisinage qui n'utilisent pas les graphes, notamment les K plus proches voisins ont l'inconvénient de ne pas être efficaces lorsque la matrice de notes est creuse. Ce scénario se caractérise par la présence de nombreuses cases pour lesquelles les notes n'ont pas encore été renseignées. Supposons par exemple qu'on veuille calculer la similarité entre deux utilisateurs ( $u_1$  et  $u_2$ ) qui n'ont jamais achetés d'items ensemble. Le calcul de cette similarité ne pourra pas se faire de manière significative car ils n'ont aucun lien direct. C'est à ce moment qu'interviennent les modèles de graphes car à l'aide d'une représentation des interactions sous forme de graphes, il sera toujours possible de trouver un chemin partant de  $u_1$  à  $u_2$ , même s'il faudra passer par un grand nombre d'arêtes.

Les graphes fournissent une bonne représentation des interactions entre utilisateurs et items. Ces graphes peuvent être construits à partir des utilisateurs uniquement, des items uniquement, ou alors

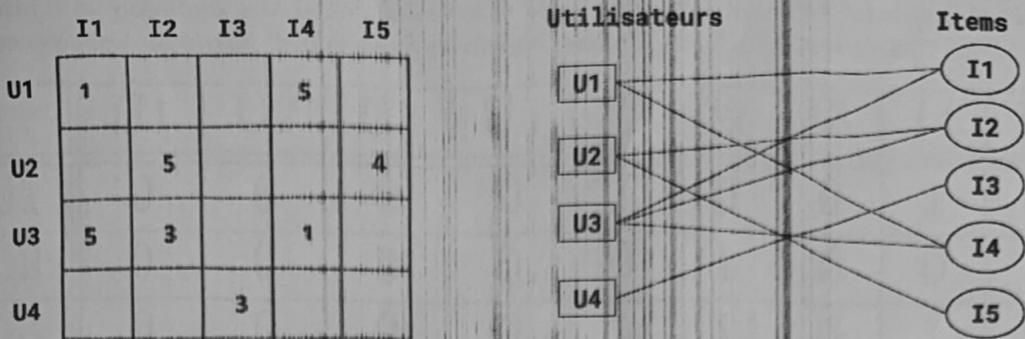


FIGURE 2 – Matrice de notes et graphe biparti classique associé.

des deux. Dans le cadre de notre travail, nous nous concentrons sur le graphe bipartie classique, avec d'un coté les noeuds utilisateurs et de l'autre les noeuds items.

### 2.3.1 Construction du graphe bipartie classique Utilisateurs-Items

Le graphe utilisateur-item est défini comme étant un graphe bipartie et non orienté  $G = (N_u \cup N_i, A)$  avec  $N_u$  étant l'ensemble des noeuds représentant les utilisateurs et  $N_i$  celui des noeuds représentant les items.  $A$  est l'ensemble des arêtes du graphe. Toutes ces arêtes existent uniquement entre utilisateurs et items. Une arête de  $A$  existe entre l'utilisateur  $u$  et l'item  $i$  si et seulement si  $u$  a noté l'item  $i$ .

Par exemple la figure 2 présente une matrice de notes et son graphe utilisateur-item associé :

### 2.3.2 Calcul des recommandations top-N : en utilisant le PageRank

Après avoir construit le graphe biparti classique, on applique un algorithme de marche aléatoire. L'hypothèse de recommandation repose ici sur la transitivité des goûts des utilisateurs et la proximité de l'utilisateur cible par rapport aux items qui correspondent à ses préférences. Ainsi, l'objectif est de recommander N items que l'utilisateur cible n'a pas encore acheté et qui sont les plus proches de lui et de son voisinage.

Le PageRank est un algorithme de marche aléatoire proposé par les cofondateurs de Google dont l'objectif initial était de classer les pages web par ordre d'importance. Plus tard, une version adaptée du PageRank (le PageRank personnalisé), a été proposée pour le calcul des recommandations des items aux utilisateurs [1]. Cet algorithme permet d'identifier les N items les plus susceptibles d'intéresser l'utilisateur cible, par une marche aléatoire dans le graphe qui a pour point de départ le nœud  $u$  associé à l'utilisateur cible. A l'issue de cette marche aléatoire, le poids de chaque nœud est stockée dans le vecteur  $PR$  calculé de manière itérative en appliquant l'équation suivante :

$$PR = \alpha * M * PR + (1 - \alpha) * d \quad (11)$$

Où  $M$  est la matrice d'adjacence du graphe construit,  $\alpha \in [0, 1]$  est le facteur d'atténuation de la personnalisation et  $d$  le vecteur de personnalisation du PageRank. Pour avoir la liste des N items à proposer à l'utilisateur cible, on récupère les scores dans  $PR$  des items que cet utilisateur n'a pas encore sélectionnés et on les trie dans l'ordre décroissant. Par la suite, les N items ayant les scores les plus grands sont retenus et recommandés à l'utilisateur cible.

En considérant la matrice de la figure 2, on obtient la matrice d'adjacence  $M$  dans le tableau 2 : le coefficient de la i-eme ligne et j-eme colonne est 1 s'il existe une arête allant du sommet i au sommet j, et 0 sinon. Le vecteur  $PR$  initial est de taille  $N = (\text{nombre d'utilisateurs} + \text{nombre d'items})$  et chaque valeur de ce vecteur initial contient la valeur  $1/N$ .  $\alpha$  est une valeur comprise entre 0 et 1.  $d$  est également un vecteur de taille  $N$  mais toutes les cases sont à 0 sauf la case correspondante à la l'utilisateur à qui on voudrait recommander les produits qui est à 1 au début de l'algorithme. On comprend donc que chaque fois qu'on veut recommander les produits à un utilisateur, on applique

l'algorithme du pageRank et à la fin on obtient le vecteur PR dans lequel est renseigné des flottants qui correspondent au niveau de préférence de chaque produit pour l'utilisateur cible.

	U1	U2	U3	U4	I1	I2	I3	I4	I5
U1	0	0	0	0	1	0	0	1	0
U2	0	0	0	0	0	1	0	0	1
U3	0	0	0	0	1	1	0	1	0
U4	0	0	0	0	0	0	1	0	0
I1	1	0	1	0	0	0	0	0	0
I2	0	1	1	0	0	0	0	0	0
I3	0	0	0	1	0	0	0	0	0
I4	1	0	1	0	0	0	0	0	0
I5	0	1	0	0	0	0	0	0	0

TABLE 2 – Matrice d'adjacence.

Les principales fonctions à implémenter ici sont les suivantes :

- **PageRank(Train-data, ID-user)** : prend en entrée l'ensemble des observations du jeu de données d'apprentissage et l'id d'un utilisateur et retourne en sortie la liste des 100 premiers produits recommandés à cet utilisateur.
- **Test-PageRank(Train-data, Test-data)** : prend le jeu de données d'apprentissage et de test et retourne en sortie une liste de 100 produits pour chaque utilisateur contenu dans le jeu de test. Ces différentes listes seront par la suite utilisées pour évaluer le modèle construit à partir du pageRank.

### 3 Evaluation du modèle construit

Evaluer le modèle consiste à le soumettre aux données de test afin de mesurer l'écart qu'il y'a entre les prédictions du modèle et les données réelles. Pour évaluer un modèle, on utilise des métriques d'évaluation. Pour le KNN et la MF, on évalue la prédiction des notes et les métriques appropriées sont la RMSE (Root Mean Square Error) et la MAE(Mean Absolute Error). La première correspond à la racine carrée de la moyenne des erreurs de prédiction de notes au carré et la seconde fait référence à la moyenne des valeurs absolues des erreurs de prédiction. L'erreur de prédiction ici est la différence entre la note réelle que l'utilisateur  $i$  a attribué à l'item  $j$  et celle prédite par le modèle de système de recommandation. Ces deux métriques sont définies par les formules 12 et 13 :

$$RMSE = \sqrt{\frac{\sum_{i,j \in r_{test}} (r_{ij} - \hat{r}_{ij})^2}{|r_{test}|}} \quad (12)$$

$$MAE = \frac{\sum_{i,j \in r_{test}} |r_{ij} - \hat{r}_{ij}|}{|r_{test}|} \quad (13)$$

avec  $r_{test}$  l'ensemble des données de test.

Pour les graphes on évalue la recommandation top-n et les métriques appropriées sont la *Mean Average Precision (MAP)*, la *Normalized Discounted Cumulative Gain (NDCG)* et le *Hit Ratio (HR)*.

**Mean Average Precision (MAP)** : En utilisant cette mesure, nous effectuons des calculs de précision en prenant en considération la position des items pertinents parmi les N recommandés. Le calcul de la *MAP* est donné par l'équation 14

$$MAP@N = \frac{\sum_{u \in U} AP_N(u)}{|U|} \quad (14)$$

$$AP_N(u) = \frac{\sum_{k=1}^N \frac{hit_k(u)}{k} * h(k)}{hit_N(u)} \quad (15)$$

avec  $AP_N(u)$  désignant la précision moyenne des recommandations top-N proposées à l'utilisateur  $u$  et  $hit_k(u)$  le nombre de bonnes recommandations pour  $k$  items recommandés à l'utilisateur  $u$ .  $U$  est l'ensemble des utilisateurs du jeu de données et  $h(k) = 1$  si l'item à la position  $k$  est une bonne recommandation et 0 sinon.

La *Discounted Cumulative Gain (DCG)* est définie par l'équation 16.

$$DCG@N(u) = \sum_{i=1}^N \frac{Gain(i)}{\log_2(i+1)} \quad (16)$$

avec  $DCG@N(u)$  représentant la *DCG* en considérant les N premiers items recommandés à l'utilisateur  $u$ .  $Gain(i) = 1$  si l'item à la position  $i$  est une bonne recommandation et 0 sinon.

la DCG, soit augmente avec N, soit reste la même. Cela signifie que plus N est grand, plus élevé sera la métrique. Une manière de rendre les comparaisons plus équitables entre différentes valeurs de DCG pour différentes valeurs de  $N$  (top-N) consiste à normaliser le score DCG en le divisant par le DCG maximum possible à chaque valeur de  $N$ . D'où la métrique *NDCG*.

**Normalized Discounted Cumulative Gain (NDCG)** : *NDCG* est calculé en divisant le gain cumulatif (*DCG*) de la liste d'items recommandés par le *DCG* de la liste de recommandation idéale. Cette liste de recommandation idéale est la liste où les items pertinents sont classés dans l'ordre le plus optimal, c'est à dire en premières positions. La *NDCG* varie de 0 à 1, les valeurs plus élevées indiquant une meilleure performance. Elle est donnée par l'équation 17 :

$$NDCG@N(u) = \frac{DCG@N(u)}{IDCG@N(u)} \quad (17)$$

Avec  $IDCG@N(u)$  étant la *DCG* idéale en considérant une recommandation de N items à l'utilisateur  $u$ .

Pour tous les utilisateurs du jeu de données la *NDCG* est donnée par l'équation 18 :

$$NDCG@N = \frac{\sum_{u \in U} NDCG@N(u)}{|U|} \quad (18)$$

**Hit ratio (HR)** : Le *Hit ratio* est la proportion d'utilisateurs à qui le système de recommandation a fait au moins une bonne recommandation. L'équation 19 présente le calcul de cette métrique.

$$HitRatio@N = \frac{\sum_{u \in U} (hit_N(u) > 0)}{|U|} \quad (19)$$

## Références

- [1] Shumeet Baluja, Rohan Seth, Dharshi Sivakumar, Yushi Jing, Jay Yagnik, Shankar Kumar, Deepak Ravichandran, and Mohamed Aly. Video suggestion and discovery for youtube : taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web*, pages 895–904, 2008.
- [2] Jian-Ping Mei, Han Yu, Zhiqi Shen, and Chunyan Miao. A social influence based trust model for recommender systems. *Intelligent Data Analysis*, 21(2) :263–277, 2017.