République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département Informatique

Mémoire de Licence

Filière : Informatique

Spécialité : Ingénierie des Systèmes d'Information et des Logiciels

---

Thème :

Système de recommandation hybride basé sur l'apprentissage profond

---

Sujet Proposé et encadré par :
Dr BERKANI Lamia

Soutenu le : **14/07/2019**

Présenté par :
KERBOUA Imene Lydia
ZEGHOUD Sofiane

Devant le jury composé de:

Dr BENSAOU N. Président (e)
Dr ZELLAL N. Membre

Binôme n° : 102 / 2019

REMERCIEMENTS

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui nous voudrions témoigner toute notre gratitude.

Nous voudrions tout d'abord adresser toute notre reconnaissance à la directrice de ce mémoire, Madame BERKANI, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter notre réflexion.

Nous tenons à remercier Madame BENSAOU et Monsieur ZELLAL qui nous ont fait l'honneur d'accepter d'examiner notre travail.

Nous désirons aussi remercier les professeurs de l'université de l'USTHB, qui nous ont fourni les outils nécessaires à la réussite de nos études universitaires.

Nous tenons à remercier également Monsieur M. HADJ AMEUR, qui a su répondre à nos questions et apporta son soutien lorsque nous l'avions demandé.

Nous voudrions exprimer notre reconnaissance envers les amis et camarades qui nous ont apporté leur soutien moral et intellectuel tout au long de cette démarche.

Enfin, nous tenons à témoigner toute notre gratitude à nos chers parents pour leur confiance et leur soutien inestimable durant cette longue année.

TABLE DES MATIÈRES

INTRODUCTION

Le développement des technologies de l'information de la communication et de l'internet de manière générale incluant les médias sociaux, les différents sites et portails, a largement facilité le partage et l'échange de données entre les gens. Cependant, ce développement, a engendré également plusieurs autres problèmes dont la difficulté d'accéder aux ressources vu le volume important de données ajouté chaque jour dans ces environnements.

Dans cette optique, les systèmes de recommandation s'avèrent être une solution incontournable pour la résolution de ce problème. Plusieurs algorithmes ont été proposés pour le filtrage d'information dont les plus couramment utilisés sont le filtrage à base de contenu (FBC) et le filtrage collaboratif (FC). Des hybridations selon différentes combinaisons permettent de bénéficier des avantages de ces deux algorithmes. Plusieurs travaux s'intéressent aujourd'hui à l'utilisation de ces techniques avec un même objectif comment améliorer les performances des algorithmes de recommandation pour arriver à prédire de la manière la plus efficace possible des items à des utilisateurs.

L'objet de ce projet de fin d'études de licence est l'utilisation d'une des techniques de l'intelligence artificielle qui est l'apprentissage profond (*Deep learning*) avec les techniques de filtrage d'information. Ayant prouvé son efficacité dans de nombreux domaines tel que le traitement du langage naturel, le "*Deep Learning"* est depuis quelques années devenu l'outils qu'utilisent beaucoup d'entreprises (comme Netflix et Amazon) ou laboratoires de recherche pour développer des systèmes de recommandation. C'est pour cela que nous nous intéressons à la même problématique pour l'implémentation d'un modèle de *deep learning* pour la recommandation d'items.

S'inspirant de l'approche proposée par He et al. [5] de filtrage collaboratif neuronal (NCF : Neural CF) basée sur les deux modèles : (1) la Factorisation Matricielle Généralisée, (Generalized Matrix Factorization - GMF) et le Perceptron Multicouches (Multi Layer Perceptron – MLP), nous avons proposé notre propre approche appelée NHybF (Neural Hybrid Filtering), qui combine le FC et le FBC selon la même architecture basée sur les modèles GMF et HybMLP.

Après cette introduction, ce mémoire sera structuré en trois chapitres comme suit :

Le chapitre 1 – Etat de l'art : présente les concepts de base liés à la recommandation et à l'apprentissage profond, ainsi que des travaux liés à l'utilisation de l'apprentissage profond dans la recommandation.

***Le chapitre 2- Conception* :** propose une approche de recommandation hybride combinant le FC et FBC selon une architecture neuronale multicouches qui utilise le modèle linéaire GMF et le modèle non linéaire HybMLP.

Le chapitre 3 – Implémentation et expérimentation : présente notre système de recommandation ainsi que les tests effectués pour évaluer notre approche en utilisant deux bases de tests : la base connue MovieLens 1m et la base Yelp.

Finalement, la conclusion générale résume notre contribution et présente quelques perspectives futures.

CHAPITRE 1 :

ETAT DE L'ART

1 - Introduction

L'implémentation d'un système de filtrage d'information, de nos jours, est devenue indispensable aux sites web et applications vu le nombre considérable de données y étant contenues, notamment ceux d'e-commerce comme Amazon[1] qui doit sa renommée à son système de recommandation, sans lequel les utilisateurs se perdraient dans les milliers d'articles proposés par le site et ne trouveraient jamais ce qui les intéresserait sans perdre de temps. D'autres applications comme Netflix[2] et Youtube[3] font également appel au filtrage de données dans le but d'offrir de meilleures expériences à leurs usagers en répondant à leurs besoins.

Cette nécessité de filtrage d'informations a fait de la recommandation un important domaine de recherche. Différentes techniques de filtrage ont été élaborées dont le filtrage collaboratif, qui est l'une des techniques les plus productives dans le domaine de la recommandation. Il fut associé récemment aux réseaux de neurones, qui ont prouvé leur efficacité dans d'autres domaines tels que le traitement du langage naturel et la reconnaissance vocale.

Ce chapitre est agencé de la manière suivante : dans la *section 2* nous présenterons des généralités sur les systèmes de recommandation et les différentes techniques utilisées pour le filtrage d'information. Dans la *section 3*, suivra une présentation de l'apprentissage automatique et quelques travaux sur des techniques de filtrage collaboratif basé sur les réseaux de neurones.

2 - Systèmes de recommandation

2.1 - Définition

Les systèmes de recommandation peuvent être considérés comme étant des algorithmes permettant le filtrage ou la sélection des données afin d'offrir une meilleure expérience aux utilisateurs interagissant avec l'application, et ceci en leur suggérant des éléments susceptibles de leur plaire. Ils réduisent considérablement le temps de recherche aux usagers de l'application et permettent une meilleure exploitation de celle-ci. Pour cela, il existe plusieurs méthodes de filtrage de données dont les plus utilisées : le filtrage cognitif, collaboratif et hybride.

[1] https://www.amazon.com/
[2] https://www.netflix.com/
[3] https://www.youtube.com/

2.2 - Approches de recommandation

2.2.1 - Filtrage à base de contenu

Le filtrage cognitif ou bien filtrage basé sur le contenu s'appuie sur la comparaison de profils d'utilisateurs à ceux des items pour leur recommander des éléments se rapprochant de leurs goûts. Un profil est un ensemble de thèmes décrivant les centres d'intérêt et les préférences d'un utilisateur.

Cette méthode de filtrage rencontre quelques difficultés dont :
- La restriction du champ de recommandation, autrement dit l'utilisateur ne verra que ce que qui est indiqué sur son profil (effet entonnoir).
- Le problème de thématique, c'est à dire que des items similaires mais avec une thématique différente n'apparaîtront jamais.
- Problème à la première interaction, l'utilisateur a du mal à spécifier ce qui l'intéresse.

Elle présente pour autant des avantages dont les suivants :
- L'indépendance des utilisateurs, aucun utilisateur ne dépend de l'avis d'un autre.
- Le problème de démarrage à froid n'est pas considérable.
- Le système fonctionne parfaitement avec une communauté faible d'utilisateurs.

2.2.2 - Filtrage collaboratif

Le filtrage collaboratif - FC - a pour principe de recommander à un utilisateur des items en se basant sur ce que d'autres utilisateurs en pensent sans avoir recours à l'analyse du contenu des items, contrairement au filtrage cognitif qui lui, assure une certaine indépendance entre les utilisateurs en se basant sur le contenu de l'item.

Nous pouvons distinguer deux grandes catégories d'algorithmes utilisés pour le filtrage collaboratif :
- Filtrage collaboratif basé mémoire.
- Filtrage collaboratif basé modèle.

1) Filtrage collaboratif basé mémoire

Ce type de filtrage utilise la globalité de la base de données des évaluations afin de faire des prédictions, la complexité des traitements est très élevée lorsque le nombre d'utilisateurs et d'items est important. Deux approches peuvent être perçues :

- ***Le filtrage basé sur l'utilisateur (user-based CF) :***

Dans ce cas, nous cherchons à prédire un item *i* à l'utilisateur *u* en se basant sur l'évaluation de l'item *i* par des utilisateurs ayant les mêmes goûts que l'usager *u*. Et donc, comme première étape viendra la formation de groupes d'usagers ayant les mêmes goûts et ainsi la prédiction des évaluations sur les items.

- ***Le filtrage basé sur l'item (item-based CF) :***

A la différence du filtrage basé-utilisateur, nous cherchons ici à recommander des items similaires à ceux que l'utilisateur *u* a déjà évalués. Il faudra donc pour commencer créer des groupes d'items similaires et par la suite prédire leurs évaluations.

Les algorithmes du filtrage collaboratif basé mémoire passent par deux étapes majeures, la première qui est l'établissement de communautés (voisinages), ceci en calculant la similarité entre les utilisateurs ou les items selon différentes méthodes telles que la similarité vectorielle (*cosine similarity*) ou la corrélation de Pearson. La seconde qui est le calcul de prédiction en se basant sur les communautés établies précédemment, l'une des méthodes les plus utilisées est celle de la somme pondérée (pour plus de détails voir l'annexe A1).

2) Filtrage collaboratif basé modèle

La structure et conception des modèles permettent au système de reconnaître de complexes caractéristiques en se basant sur les données d'entraînement [1], puis faire des prédictions sur les données de test ou les données réelles tout en se basant sur les modèles appris.

Plusieurs techniques sont utilisées dont nous citons [1]: le clustering, les réseaux de Bayes, les algorithmes de régression, les modèles de facteurs latents comme *Matrix Factorization* (MF) avec ses différents modèles dont les suivants [2] : *Singular Value Decomposition* (SVD), SVD++ et *Probabilistic Matrix Factorization* (PMF). Une autre approche récente a fait son entrée dans le domaine de la recommandation, il s'agit de l'apprentissage profond et des réseaux de neurones [6].

- ***Matrix Factorization :***

Depuis la compétition qu'avait lancée Netflix dont le prix était à 1 million de dollars "*Netflix Prize*" en 2006, *Matrix Factorization* devient l'une des méthodes les plus utilisées pour la conception de modèles de filtrage de données.

Le but de cette méthode est principalement de réduire la matrice d'évaluation, mais aussi créer des vecteurs regroupant les caractéristiques de chaque utilisateur (item). On procède en découpant la matrice d'évaluation en deux ou plusieurs sous-matrices de sorte à ce que leur multiplication élément par élément donne la matrice de départ avec les cases qui étaient initialement vides remplies, ce qui donne les prédictions.

La figure suivante illustre la décomposition de la matrice :

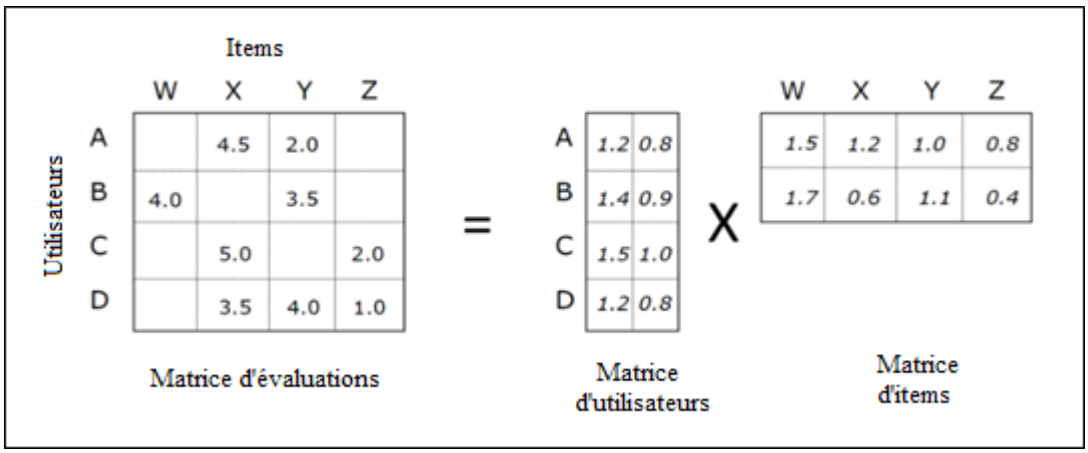


Figure 1.1 : Décomposition de la matrice d'évaluation en deux sous-matrices.

Mathématiquement parlant, la décomposition peut être traduite par la formule qui suit :

$$R_{(m,n)} = U_{(m,k)} \times I_{(k,n)} \quad (1.1)$$

Où:

$R_{(m,n)}$: Matrice des évaluations avec *m* utilisateurs et *n* items.

$U_{(m,k)}$: Matrice des utilisateurs.

$I_{(k,n)}$: Matrice des items.

3) Avantages et inconvénients du filtrage collaboratif

Les techniques de filtrage collaboratif présentent des avantages et des inconvénients dont les suivants [2] :

Tableau 1.1 : Avantages et inconvénients du FC.

Techniques de filtrage collaboratif	Avantages	Inconvénients
FC basé mémoire	- Facile à implémenter. - Ajout facile de nouvelles données.	- Problème de démarrage à froid. - Problème de matrice creuse. - Espace limité, non-idéal pour traitement de données massives.
FC basé modèle	- Améliore les prédictions. - Règle les problèmes liés aux matrices (présence de creux et espace limité).	- Structure coûteuse. - Problème de démarrage à froid. - Nécessite beaucoup de données pour l'apprentissage et une large communauté d'utilisateurs.

2.2.3 - Filtrage hybride

Il s'agit de combiner le filtrage collaboratif et le filtrage à base de contenu afin de remédier aux problèmes posés par les deux méthodes, c'est-à-dire d'élaborer un système de recommandation varié, capable de démarrer à froid et qui n'a pas d'effet entonnoir. Plusieurs hybridations sont utilisées dont la pondération et la cascade, etc. [3]

3 - Filtrage collaboratif et réseaux de neurones

3.1 - Apprentissage automatique (*Machine Learning*)

L'apprentissage automatique *(en anglais: Machine Learning)* est un champ d'étude de l'intelligence artificielle qui cherche à rendre les machines capables d'apprendre sans être explicitement programmées, cette définition a été donnée par Arthur Samuel pionnier de l'intelligence artificielle, en 1959. Tom Mitchell a apporté une nouvelle définition qui est la suivante: "Un programme apprend d'une expérience «*E*» suivant une certaine classe de

tâches «*T*» et une mesure de performance «*P*», si sa performance à la tâche «*T*», mesurée par «*P*», augmente avec l'expérience «*E*»" [7].

Par exemple, nous avons un programme qui filtre les emails en spam et non-spam, ce programme devra apprendre à faire cette classification automatiquement en se basant sur ce que l'utilisateur a déjà classé. La tâche «*T*» est la classification des emails en spam et non-spam. L'expérience «*E*» n'est que la classification effectuée par l'utilisateur, le programme apprend en quelque sorte à mimer l'utilisateur. La mesure de performance «*P*» pourrait être le nombre d'emails classés correctement par rapport à tous les emails [13].

Les problèmes d'apprentissage automatique sont en général soit des problèmes d'apprentissage supervisé (*Supervised Learning*) ou des problèmes d'apprentissage non-supervisé (*Unsupervised Learning*). La différence entre les deux est qu'en apprentissage supervisé, les résultats attendus sont connus mais en apprentissage non-supervisé ils ne le sont pas.

L'une des méthodes utilisées pour l'apprentissage supervisé est l'implémentation de réseaux de neurones. Nous nous intéresserons dans ce travail à l'intégration des réseaux de neurones (*RN*) aux systèmes de recommandation. Nous donnons ci-dessous une description de cet algorithme avant de voir son intégration dans les systèmes de recommandation.

3.2 - Réseau de neurones et apprentissage profond (Deep Learning)

3.2.1 - Réseau de neurones

Le fonctionnement d'un neurone artificiel est inspiré de celui d'un neurone biologique, autrement dit, il exerce les mêmes fonctions qu'un neurone biologique et donc reçoit des signaux, les traite et donne un résultat : il calcule la somme des produits des entrées par les poids et la fait passer par une fonction d'activation qui donne le résultat.

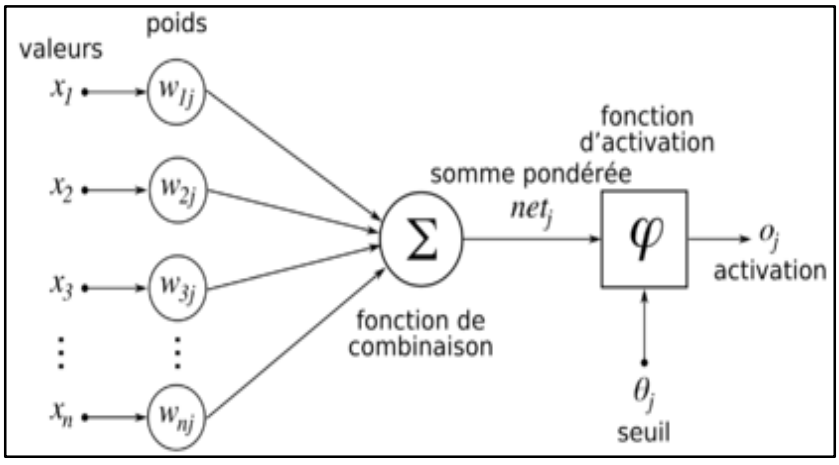


Figure 1.2 : Structure d'un neurone artificiel [15].

La prédiction o_j est donnée par la formule suivante :

$$P\ (o_j = 1 \mid x) = \ \sigma\left(\sum_{i=1}^{n} w_{ij} x_i + \ \theta_j\ \right) \qquad (1.2)$$

Où :

- Fonction d'activation : $\sigma(x) = \frac{1}{1+ e^{x}}$ pour avoir une valeur comprise entre 0 et 1. Plusieurs autres fonctions existent telles que *tanh* qui rend une valeur entre -1 et 1.
- Le seuil θ_j détermine si le nœud sera activé ou pas selon le résultat de la fonction de coût.

Un RN est une succession de couches, où chaque couche contient un nombre N de neurones (voir figure 1.3). Les neurones de chaque couche sont connectés avec ceux de la couche précédente.

Les RNs utilisés suivent généralement l'architecture suivante :

- Une couche d'entrée *(Input Layer)*.
- Couches cachées *(Hidden Layer)* : ce sont des couches intermédiaires qui se chargent de traiter les données des couches précédentes. À noter que l'on appelle un RN à multiples couches cachées un réseau de neurones profond (*Deep Neural Network)* et est utilisé dans l'apprentissage profond (*Deep Learning*).
- Une couche de sortie *(Output Layer)*.

Le nombre de nœuds dans chaque couche peut être différent (voir figure 1.3).

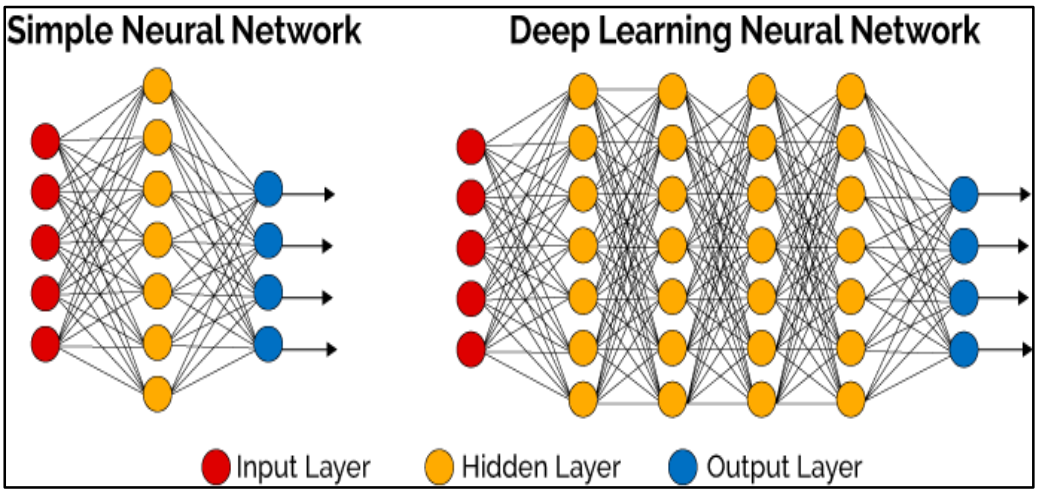


Figure 1.3 : Architecture d'un RN simple (à gauche) et celle d'un RN profond (à droite) [4].

Il existe plusieurs types de réseaux de neurones dont nous citons :

- **Perceptron multi-couches (*Multi Layer Perceptron - MLP* -) :** c'est un réseau de neurones artificiels (perceptron), c'est-à-dire qu'il est constitué d'une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie.
- **Réseau de neurones convolutifs (*Convolutional Neural Network - CNN* -):** les *CNN* sont une variante des réseaux de neurones à propagation directe (*feedforward*) avec la particularité d'avoir deux couches qui leurs sont spécifiques, il s'agit des couches de convolution qui servent à extraire les caractéristiques locales, et les couches de *pooling* pour réduire les données en sélectionnant les caractéristiques représentatifs

(par exemple les caractéristiques ayant le plus haut score via les fonctions d'activation - *max pooling*) de la couche précédente qui est généralement une couche de convolution.

3.2.2 - Apprentissage profond (Deep Learning)

Le *deep learning* est un sous-domaine de l'apprentissage automatique qui utilise des réseaux de neurones profonds, la profondeur du réseau est définie par le nombre de couches utilisées. La puissance de cette technique demeure dans le fait qu'elle puisse traiter un très grand nombre de données, ce qui est nécessaire de nos jours vu l'ampleur qu'a pris le web. Le fait d'avoir plusieurs couches de traitement rend l'apprentissage plus précis et permet l'approximation de fonctions assez complexes. Néanmoins, un nombre de couches trop élevé pourrait provoquer un sur-apprentissage (*overfit*).

3.3 - Apprentissage profond et recommandation

Ayant prouvé son efficacité dans plusieurs autres domaines tels que le traitement du langage naturel, les *chatbots* etc., le *deep learning* a fait son entrée dans la recommandation récemment. Il est utilisé pour plusieurs raisons dont les suivantes [6] :

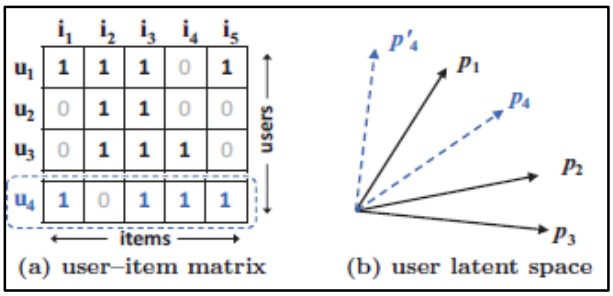
- **Transformation non-linéaire :** à travers les différentes fonctions d'activation telles que "*ReLU*" et "*Sigmoid*" etc. Ce type de transformation permet l'augmentation ou la diminution de la relation linéaire entre deux variables et donc leur corrélation. Les modèles de factorisation matricielle sont des modèles linéaire, qui font des combinaisons linéaires entre les facteurs latents des items et des utilisateurs, cette approche ne permet pas la capture de formes d'interactions utilisateur-item complexes.
- **Plusieurs couches et modèles complexes :** l'apprentissage profond, contrairement à l'apprentissage automatique, permet la création de modèles complexes avec plusieurs couches de traitement (cachées) de différents types tels que les couches de convolution ou les LSTM[4].
- **Flexibilité :** le développement de modèles de *deep learning* est devenu très simple grâce à l'existence de plus d'un framework (*Keras*, *Tensorflow*...etc).

3.4 - Travaux liés à l'utilisation des RNs avec le FC

3.4.1 - Filtrage collaboratif basé sur les réseaux de neurones

Une étude récente d'He et al. (2017) [5] où sont relevés quelques problèmes du modèle de MF (figure 1.4), propose de créer un perceptron multicouches (MLP) qui permet l'approximation de la fonction d'interaction sans faire de produit entre les facteurs latents. Par la suite, le modèle MF a été associé au MLP pour avoir de meilleurs résultats en termes de précision de l'algorithme. De cette concaténation des deux modèles relèvera un troisième modèle de factorisation matricielle neuronale (NeuMF).

[4] **LSTM :** Acronyme de Long Short Term Memory, utilisé dans les réseaux de neurones récurrents.



(a) Matrice d'interaction utilisateur-item * (b) Espace latent de l'utilisateur

Figure 1.4 : Limitation de MF en utilisant le feedback implicite [5].

Le vecteur p_4 devrait être plus proche du vecteur p_3 que de p_2.

L'architecture du modèle est résumée dans la figure ci-dessous :

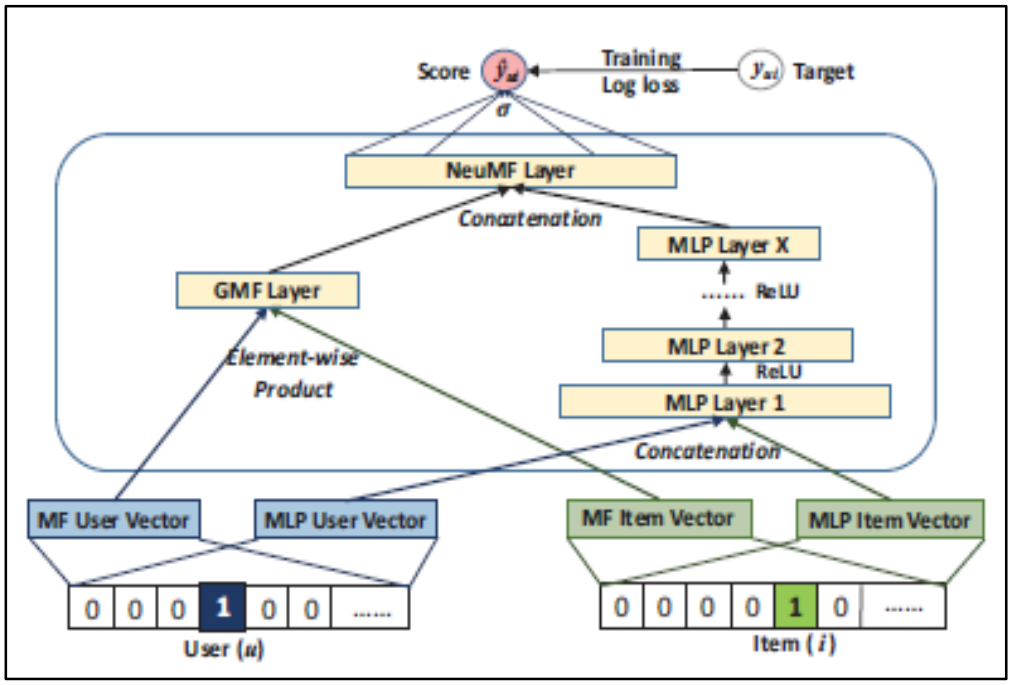


Figure 1.5 : Modèle FC Neuronal (*Neural Collaborative Filtering* -NCF-) [5].

En entrée sont donnés les identifiants des utilisateurs et des items car ce travail se concentre sur le FC classique. La couche suivante constitue les vecteurs des facteurs latents des utilisateurs et items pour chaque modèle, sous forme de couche d'inclusion (*embedding layer*)[5]. Puis deux parties, l'une où est repris le modèle général de factorisation matricielle, une deuxième qui apprend la fonction d'interaction. Enfin, les résultats de ces deux méthodes sont regroupées afin d'améliorer la prédiction. En sortie, une approximation de l'interaction 1 ou 0 qui peut être interprétée comme l'intérêt que portera l'utilisateur à l'item.

Pour conclure, les résultats obtenus sont nettement supérieurs à ceux des modèles déjà existants (eALS [8], BPR [10], ItemKNN [9], ItemPop [11]) sur les différents jeux de

[5] **Couche d'inclusion (*embedding layer*) :** plus de détails dans le chapitre 2, section 2.

données (MovieLens-1m[6], Pinterest[7]). Le modèle a été évalué avec les métriques Hit Ratio (HR)[8] et Normalized Discounted Cumulative Gain (NDCG)[7]. Il a aussi été conclu de cette étude que plusieurs couches cachées dans les modèles permettent un meilleur apprentissage de l'interaction utilisateur-items.

3.4.2 - Filtrage collaboratif basé sur les réseaux de neurones convolutifs

Nous présentons l'approche de recommandation par FC avec RN convolutifs, les auteurs **He et al** [12] ont considéré qu'il était plus intéressant et sémantiquement plausible d'utiliser un produit dyadique pour représenter les corrélations entre les *embeddings* des utilisateurs et ceux des items, le but étant de mettre en évidence le lien entre ces données.

L'approche proposée a pour première étape la construction d'une matrice d'interaction de dimension $K \times K$ (K étant la taille de l'embedding i.e. le nombre de facteurs latents) à partir du produit dyadique (tensoriel) des facteurs latents des utilisateurs et items. En seconde étape, la matrice résultante est passée par un réseau de neurones convolutifs (voir figure 1.6) qui se charge de récupérer les caractéristiques des interactions utilisateur-item. Le nombre de couches dépend du nombre de facteurs K (voir figure 1.7). Enfin, le modèle retourne 1 ou 0 pour déterminer si l'utilisateur porte un intérêt à l'item ou non comme dans le travail précédent.

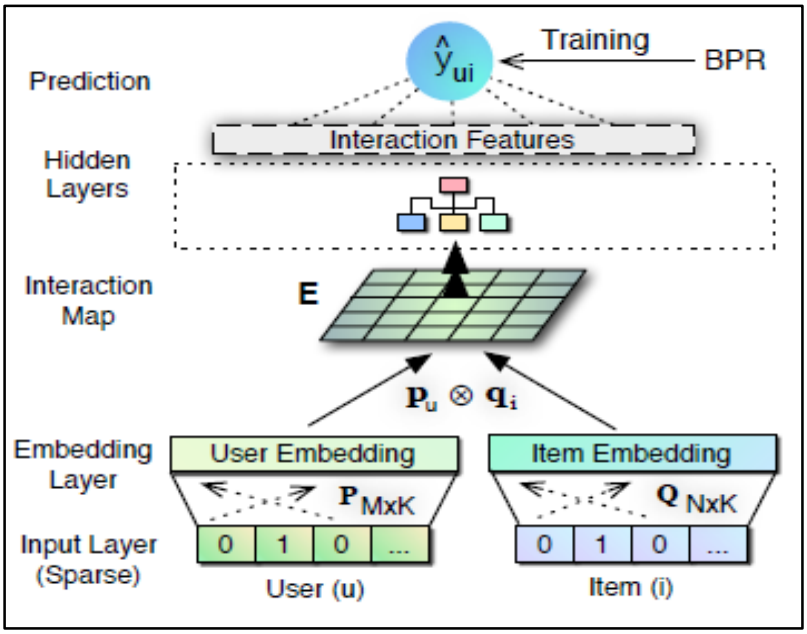


Figure 1.6 : Architecture NCF basé sur le produit tensoriel [12].

La figure (1.6) illustre un exemple d'architecture de ConvNCF avec 6 couches de convolution et une taille d'*embedding* de 64.

[6] http://grouplens.org/datasets/movielens/1m/
[7] https://sites.google.com/site/xueatalphabeta/academic-projects
[8] **HR** et **NDCG :** plus de détails dans le chapitre 3, section 3.

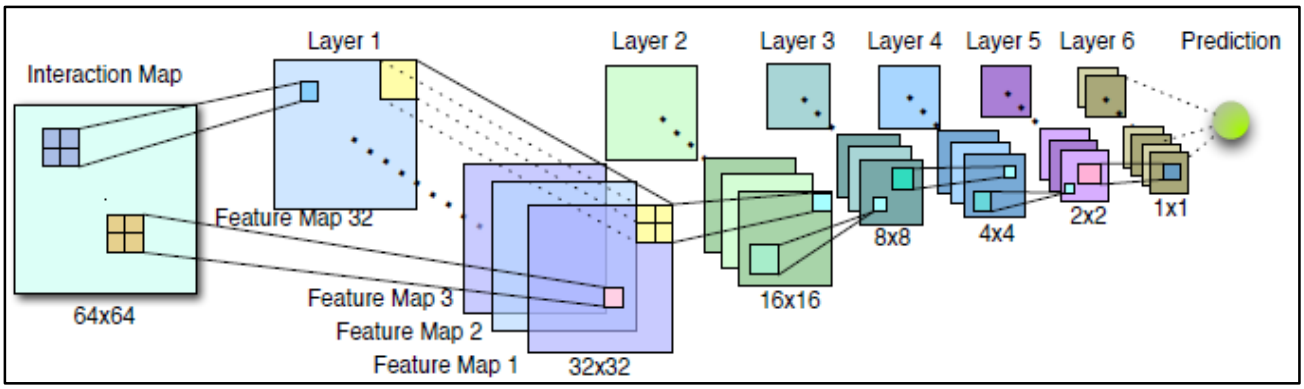


Figure 1.7 : Exemple d'architecture de ConvNCF [12].

En conclusion, le modèle proposé surpasse les méthodes avec lesquelles il a été comparé (NCF [5], MF) en termes de précision de la prédiction.

4 - Conclusion

Nous pouvons conclure de ce premier chapitre qu'il existe plusieurs méthodes de recommandation, chacune ayant ses avantages et inconvénients. Nous avons pu voir aussi que l'apprentissage profond basé sur les réseaux de neurones apporte de meilleurs résultats comparé aux algorithmes d'apprentissages traditionnels.

Nous présentons dans le chapitre suivant notre propre approche de recommandation basée sur le FC et l'apprentissage profond, en s'inspirant des travaux présenté dans ce chapitre.

CHAPITRE 2 :

CONCEPTION

1 - Introduction

Après avoir défini les différentes approches de filtrage et avoir présenté quelques travaux sur le filtrage collaboratif et les réseaux de neurones dans le premier chapitre, nous passons dans cette partie à l'implémentation d'une approche de recommandation basée sur les réseaux de neurones profonds.

Ce chapitre est composé de deux sections, la première où nous présenterons les détails de l'approche proposée. Dans la deuxième section nous expliquerons les détails de l'entraînement de celle-ci ainsi que les modifications apportées au *dataset*.

2 - Approche proposée

2.1 - Motivation

Dans l'approche proposée par **He et al [5]** où seuls les identifiants des utilisateurs et items sont utilisés afin de créer les facteurs latents représentant leurs caractéristiques, et donc un problème de démarrage à froid se pose. En effet, le modèle arrive à prédire l'interaction d'un utilisateur *u* avec item *i* seulement s'il connaît ces derniers et s'est déjà entraîné à prédire l'interaction de ce même utilisateur *u* avec d'autres items ainsi que l'interaction d'autres utilisateurs avec l'item *i*. Le modèle rencontre donc un problème si nous essayons de prédire l'interaction d'un utilisateur et/ou d'un item jamais rencontré(s) au préalable.

Il y a possibilité de minimiser le problème, voire de le corriger. Et cela en exploitant les données des utilisateurs (telles que l'âge, le sexe, l'occupation, etc.) ou items (genres, année de sortie, etc.) déjà recueillies qui ne seront pas apprises mais données en entrée.

Ces données permettraient aussi d'améliorer le modèle proposé par **He et al [5]** en lui intégrant une partie de filtrage cognitif et donc créer une hybridation. Le nouveau modèle apprendra donc par lui-même comment exploiter ces nouvelles données qui lui seront plus ou moins pertinentes et pourra ainsi mieux prédire l'interaction utilisateur-item.

2.2 - Système de recommandation hybride basé RN - NHybF (Neural Hybrid Filtering)

2.2.1 - Description générale du modèle et son architecture

Dans cette partie, nous proposons une hybridation du FC avec le filtrage cognitif. En entrée sont données des informations sur l'utilisateur et les caractéristiques des items. Le type d'informations données à un réseau de neurones peut être catégorique comme l'état civil de la personne ou continu comme l'âge ou le poids de quelqu'un. Le réseau devra retourner une prédiction de l'interaction, c'est-à-dire une valeur comprise entre 1 et 0 qui indique l'intérêt que porte l'utilisateur à l'item.

Le schéma suivant décrit l'intégration de ces données au modèle :

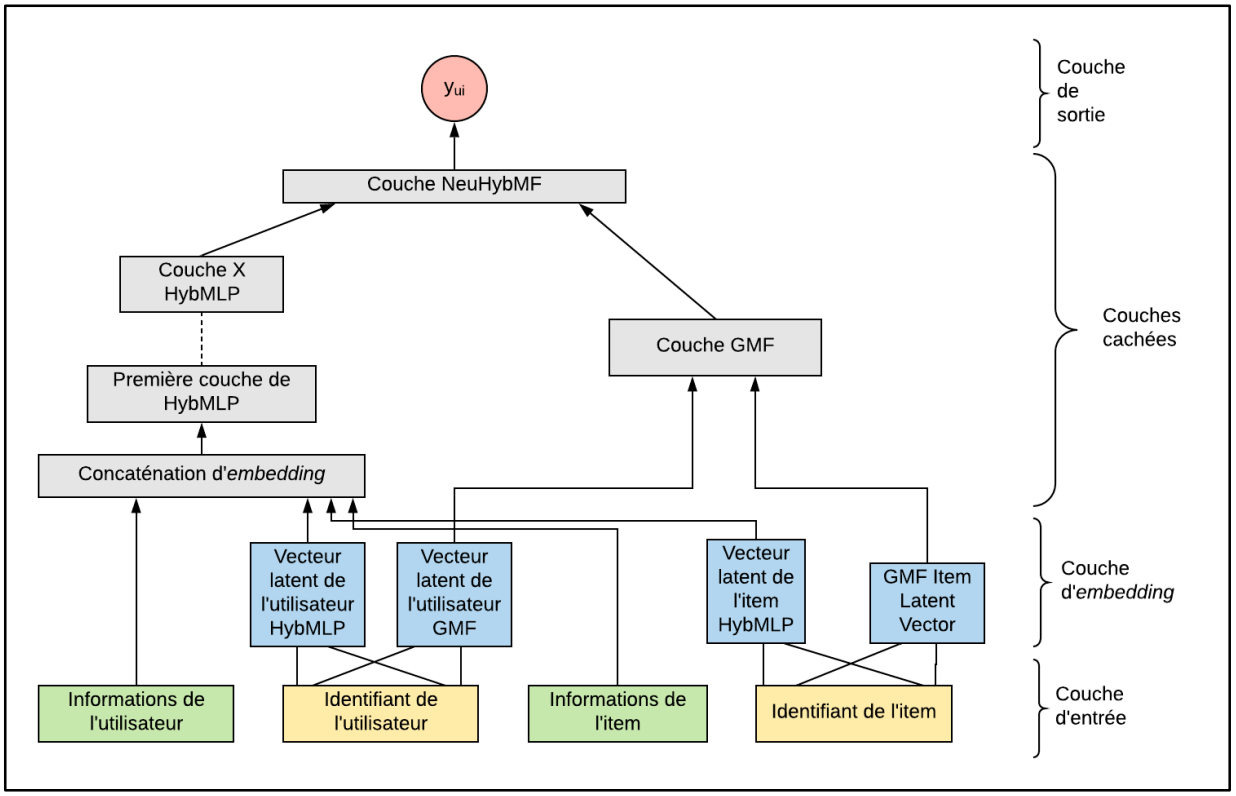


Figure 2.1 : Architecture du modèle proposé.

Les deux modèles (GMF et HybMLP) peuvent être construits et entraînés séparément comme il sera expliqué plus bas.

2.2.2 - Description des couches du modèle

1 – Couche d'entrée (*Input Layer*) :

Dans cette approche, nous prenons en entrée les identifiants des utilisateurs et des items, en plus des informations et caractéristiques recueillies relatives à chacun d'entre eux. Ces informations seront par la suite ajoutées aux facteurs appris par le modèle donc concaténées à l'*Embedding*.

2 – Couches d'inclusion (*Embedding Layer*) :

L'*embedding* est la représentation de variables discrètes en un vecteur de nombres continus. En termes de réseaux de neurones, il s'agit de vecteurs de représentation de variables discrètes appris.

Nous prenons comme exemple le *Word Embedding* qui s'est avéré très efficace dans l'interprétation du langage naturel. Les mots sont préalablement codés en *One-Hot Encoding*[9],

[9] **One Hot Encoding :** Codage vectoriel dans un vecteur à N dimensions où l'on prévoit un vocabulaire de N mots, le *i*-ème mot sera représenté par un vecteur d'entiers dont la seule valeur non-nulle sera un 1 situé à la position *i*. Le vecteur sera donc très long et surtout très creux.

le but est ainsi de concevoir un algorithme capable de former des vecteurs moins creux et qui surtout posséderaient une relation logique entre eux, par exemple, les vecteurs des mots "Journal", "Magazine" ne seront pas éloignés les uns des autres si représentés dans un espace vectoriel (voir figure 2.2), à l'inverse des mots comme "Journal" et "Randonnée".

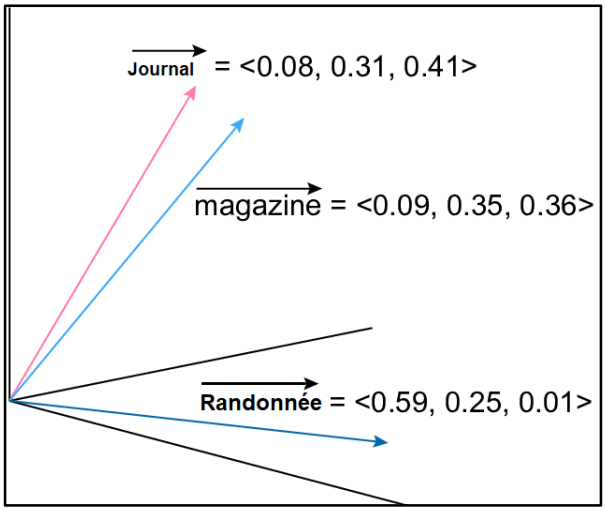


Figure 2.2 : Représentation vectorielle des mots avec la valeur du vecteur de chaque mot.

À l'exemple du Word Embedding, nous cherchons à représenter les items ainsi que les utilisateurs sous forme vectorielle. Ces nombres représentent les caractéristiques de l'utilisateur (ou l'item) apprises automatiquement, l'algorithme reconnaitra ainsi les utilisateurs ayant un taux élevé de similarité, de même pour les items.

La figure ci-dessous (figure 2.3) explique la création des vecteurs latents pour un élément donné, l'index de l'élément est codé en *One-Hot vector* puis multiplié par les poids du RN pour donner le vecteur de représentation (*embedding*).

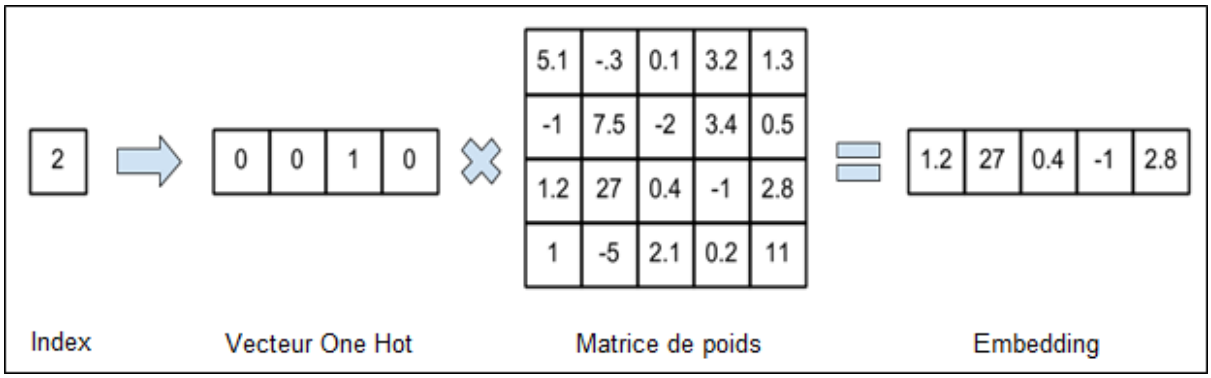


Figure 2.3 : Exemple de création d'*embedding*.

Il est préférable de permettre aux deux modèles (GMF et HybMLP) d'apprendre les caractéristiques des utilisateurs et items séparément pour rendre le modèle plus flexible, sachant que le nombre optimal de facteurs pour HybMLP et GMF n'est pas forcément le même.

Il faut noter que les facteurs latents de ces modèles sont appris à partir des identifiants des utilisateurs et items seulement.

3 – Partie Factorisation Matricielle Généralisée (*Generalized Matrix Factorization* - GMF) : *Modèle linéaire*

Cette partie représente l'opération faite par la factorisation matricielle pour calculer les prédictions, ce modèle est composé des couches suivantes (voir figure 2.4) :

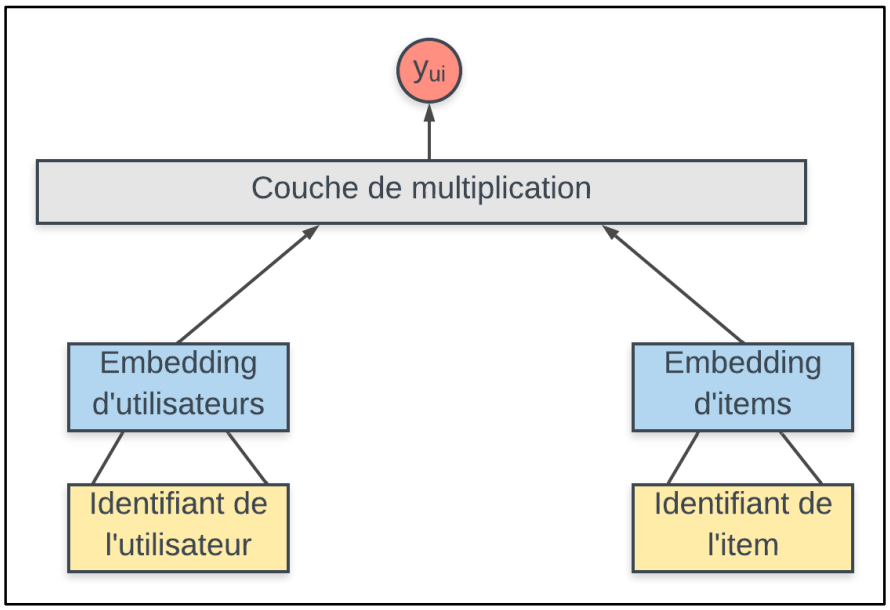


Figure 2.4 : Architecture du modèle GMF.

Description textuelle des couches du modèle :

- *Embedding* d'utilisateurs GMF : Vecteurs de facteurs latents des utilisateurs.
- *Embedding* d'items GMF : Vecteurs de facteurs latents des items.
- Couche de multiplication : Dans cette couche il s'agit de faire la multiplication élément par élément de l'*embedding* (facteurs) des utilisateurs et celui des items, suivant cette formule [5] :

$$\phi^{GMF} = p_u^G \odot q_i^G \qquad (2.1)$$

Où :

p_u^G : Vecteur des facteurs latents de l'utilisateur *u* de GMF.

q_i^G : Vecteur des facteurs latents de l'item *i* de GMF.

4 – Partie Perceptron Multicouches Hybride (*Hybrid Multi Layer Perceptron* - HybMLP): *Modèle non-linéaire*

Cette partie se charge de l'apprentissage de la fonction d'interaction donnée par la formule suivante [5] :

$$y_{ui} = \begin{cases} 1 & s'il\ y\ a\ une\ interaction\ entre\ l'utilisateur\ u\ et\ l'item\ i \\ 0 & sinon \end{cases} \qquad (2.2)$$

Pour cela le modèle HybMLP suit l'architecture suivante (voir figure 2.5) :

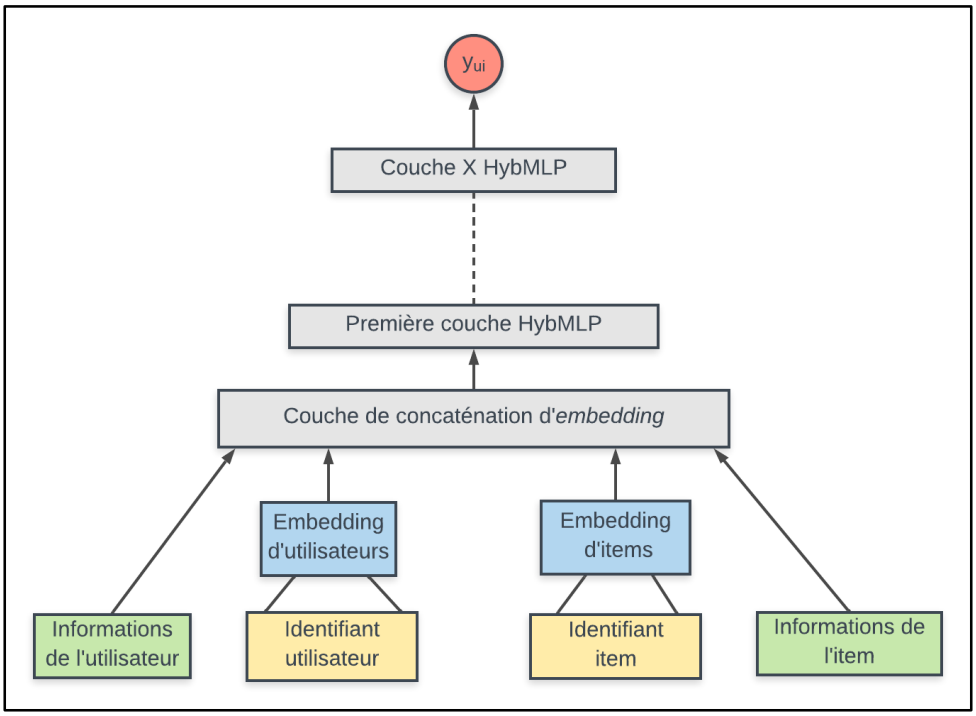


Figure 2.5 : Architecture du modèle HybMLP.

Description textuelle des couches du modèle :

- *Embedding* d'utilisateurs HybMLP : Vecteurs de facteurs latents des utilisateurs.
- *Embedding* d'items HybMLP : Vecteurs de facteurs latents des items.
- Couche de concaténation : Dans cette couche a lieu la concaténation des *embeddings* avec les caractéristiques non-apprises données en entrée. Cette couche constitue la première couche cachée du modèle HybMLP. La concaténation ne suffit pas pour apprendre l'interaction entre les caractéristiques des utilisateurs et des items donc ces données sont passées par d'autres couches cachées.
- Autre(s) couche(s) cachée(s) (*HybMLP Hidden Layer(s)*) : Ces couches sont ajoutées pour permettre l'apprentissage des interactions comme expliqué précédemment. La fonction d'activation utilisée dans chaque couche est *ReLU* (*Rectified Linear Unit* : $ReLU(x) = max(0, x)$) car elle permet de réduire les risques de sur-apprentissage et la saturation des neurones. La formule du modèle est donnée ci-dessous [5] :

$$\phi^{MLP} = a_L(W_L^T(a_{L-1}(\dots a_2(W_2^T \begin{bmatrix} p_u^M \\ f_u \\ q_i^M \\ f_i \end{bmatrix} + b_2)\dots)) + b_L) \qquad (2.3)$$

Où :

a_L : Fonction d'activation de la couche L.
b_L : Bias de la couche L (fonctionne comme le seuil).
W_L^T : Poids de la couche L.

p_u^M : Vecteur des facteurs latents de l'utilisateur *u* de HybMLP.
f_u : Vecteur d'informations de l'utilisateur *u*.
q_i^M : Vecteur des facteurs latents de l'item *i* de HybMLP.
f_i : Vecteur d'informations de l'item *i*.

Il se peut qu'il n'y ait pas de couche de traitement en plus de celle de la concaténation, alors les caractéristiques seront directement passées à la couche de sortie. On appellera cette version HybMLP-0 (réseau de neurones simple) qui sera testée dans le chapitre 3.

Pour décider du nombre de nœuds optimal de chaque couche nous en fixons un pour la dernière que nous appellerons *predictive factors* [5], la couche précédente aura le double de ce nombre de nœuds et ainsi de suite jusqu'à la première. Il est conseillé de répartir le nombre de nœuds par couche dans un ordre descendant, c'est-à-dire que la première couche cachée aura le plus grand nombre de nœuds ce dernier diminuera jusqu'à la dernière couche.

La figure ci-dessous (figure 2.6) illustre les détails de chaque couche, c'est-à-dire le nombre de nœuds par couche du modèle HybMLP. Par exemple, si nous prenons PF = 16 et le nombre de couches X=3, nous aurons un modèle comme suit : [64 → 32 → 16].

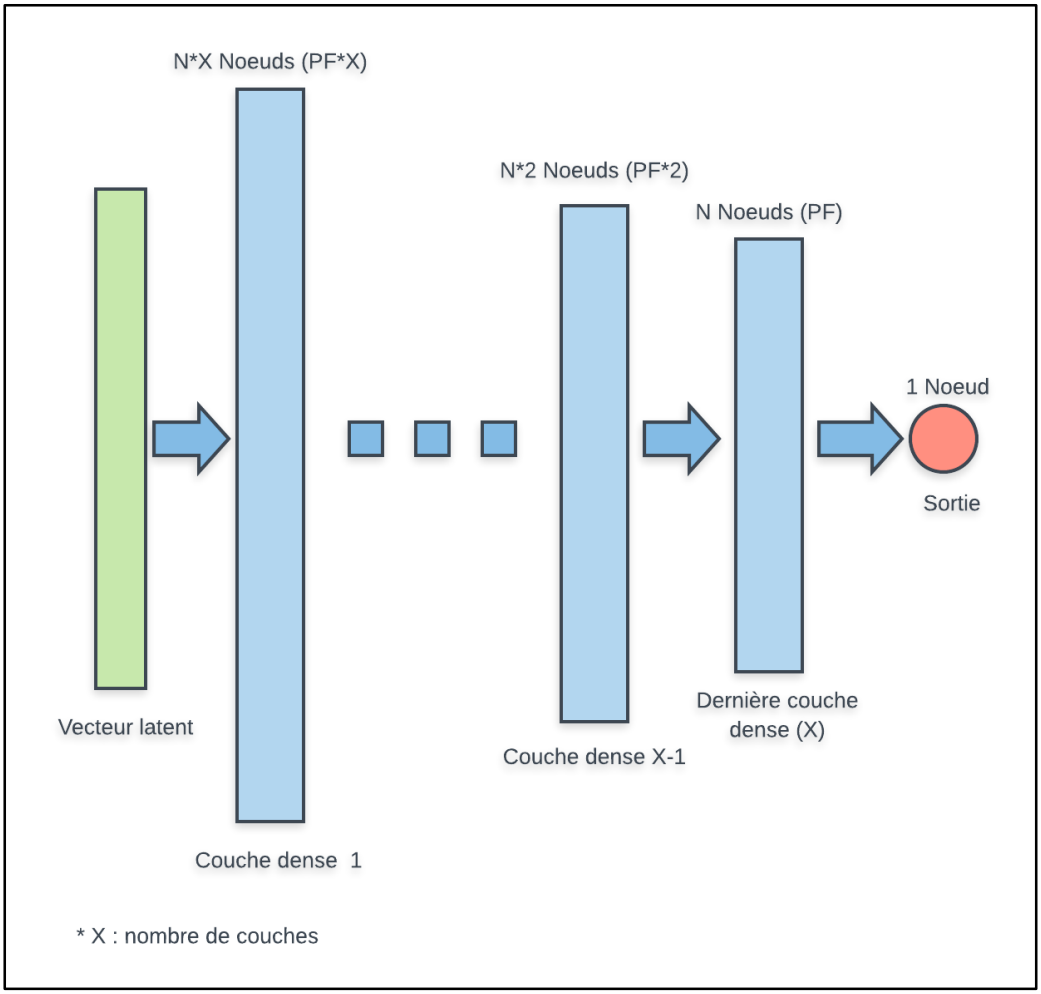


Figure 2.6 : Structure du modèle selon le nombre de nœuds par couche.

5 – Partie Factorisation Matricielle Neuronale Hybride (*Neural Hybrid Matrix Factorization* - NeuHybMF) :

Dans cette partie nous concaténons les dernières couches des deux modèles précédents (voir figure 2.1).

Description textuelle de la couche :

- Couche de concaténation : Les résultats des deux modèles précédents sont combinés pour permettre d'avoir de meilleurs résultats.
  La dernière couche de HybMLP est concaténée à celle de GMF.

L'entraînement de ce modèle peut se faire de deux manières : la première avec le pré-entraînement des modèles le composant (GMF et HybMLP) et la deuxième sans le pré-entraînement de ces modèles.

6 - Couche de sortie (*Output Layer*) :

Cette couche prend en entrée les vecteurs des dernières couches de HybMLP et GMF concaténés précédemment et les fait passer par une fonction d'activation. Le résultat devant être compris entre 0 et 1, la fonction d'activation utilisée est la fonction *Sigmoid* : $\sigma(x) = 1/(1+e^{x})$. La formule de la fonction d'activation de la couche de sortie est la suivante [5] :

$$\hat{y}_{ui} = \sigma(h^{T} \begin{bmatrix} \phi^{GMF} \\ \phi^{HybMLP} \end{bmatrix}) \qquad (2.4)$$

Où :

h^{T} : Vecteur des prédictions des deux modèles GMF et HybMLP.
$\hat{y}_{ui}$: Prédiction du modèle NeuHybMF.

3 - Entraînement

3.1 - Gestion du jeu de données (*Dataset*)

3.1.1 - Exploitation des données

Les données extraites à partir des jeux de données doivent être traitées et transformées avant de passer par le modèle. Nous prenons dans ce qui suit la base MovieLens[10] pour expliquer ce traitement. La base de données met à disposition les genres des films (données relatives aux items) et aussi l'âge, l'occupation et le sexe de la personne (données relatives aux utilisateurs). Nous transformons les données en nombres qu'on pourra exploiter. Par exemple, comme un film peut faire partie de 18 genres différents, nous représentons son genre par un vecteur booléen de 18 colonnes, chacune contenant une valeur qui indique l'appartenance du film à un genre précis.

[10] https://grouplens.org/datasets/movielens/1m/

3.1.2 - Transformation des données explicites en données implicites

1) Définition des données implicites et explicites

Une importante partie du projet consiste en la transformation des données explicites en données implicites. Les données explicites représentent le degré de préférence de l'utilisateur, par exemple une note donnée à un item, un 'j'aime' ou 'je n'aime pas' qui indiquent si oui ou non l'utilisateur a apprécié l'item…etc. Tandis que les données implicites sont des données floues qui représentent plus le comportement d'un utilisateur vis-à-vis d'un item que son ressenti, cela pourrait être un clique, un achat ou bien un film visionné, il n'y a donc pas moyen de dire si l'utilisateur a apprécié l'item ou pas.

2) La raison de l'utilisation des données implicites

Dans la vie courante, les utilisateurs ne donnent que très rarement leurs avis sur les items avec lesquels ils ont interagi. La récolte de données explicites devient donc assez difficile, il est ainsi préférable de concevoir un modèle qui répond aux besoins des utilisateurs de la vie courante plutôt qu'aux utilisateurs "idéaux" qui partagent souvent leurs expériences. D'autant plus, nous aurons à notre disposition bien plus de données à exploiter.

Ainsi, si nous considérons toutes les interactions utilisateur-item du *dataset* comme étant des instances positives, nous aurons non-seulement plus de données à exploiter, mais aussi une idée du style de films auxquels l'utilisateur s'intéresse, étant donné qu'il s'y est assez intéressé pour visionner le film.

3) Construction du dataset

Nous ne pouvons pas nous permettre de laisser le model s'entraîner que sur des instances positives, ce dernier n'apprendra qu'à donner des résultats positifs. Il nous faut donc récolter des instances négatives. Ayant considéré toutes les interactions comme étant des instances positives, nous allons dans la même philosophie considérer l'absence d'interaction entre un utilisateur et un item comme étant une instance négative. Le choix nous revient quant au nombre d'instances négatives que nous voulons ajouter et ainsi l'équilibre entre les instances positives et négatives. Le nombre d'instances négatives devient ainsi un paramètre variable et influe sur les résultats finaux.

A noter que comme pour "instance positive", le terme "instance négative" ne représente pas forcément une instance où l'item *i* n'est pas pertinent à l'utilisateur *u*. Il est possible que l'utilisateur n'ait juste pas interagi avec l'item pour quelconque raison bien qu'il pourrait l'intéresser. Il est juste plus probable qu'un utilisateur s'intéresse à un item avec lequel il a interagi plutôt qu'à un autre.

4) Répartition du dataset entre l'entraînement et le test

Étant donné que nous procéderons dans le dernier chapitre à deux types d'évaluation différents qui ont une relation avec l'entraînement du modèle et ainsi une relation avec la répartition du *dataset.* Les détails de cette dernière sont donnés dans ce qui suit :

- **Premier type d'évaluation [5] :**

- ***Les données d'entraînement :***

Nous prenons le *dataset* de données implicites qui ne contiennent jusqu'à maintenant que des instances positives et y ajoutons les instances négatives. Et cela en sélectionnant *X fois* un utilisateur *u* ainsi qu'un item *i* au hasard ne gardant que les cas où l'on ne trouve pas d'interaction entre ces deux derniers.

- ***Les données de test :***

Pour chaque utilisateur, nous choisissons au hasard un item avec lequel il a interagi (instance positive) et y ajoutons par la suite 99 items avec lesquels il n'a pas interagi (instances négatives) et cela pour simuler les conditions réelles dans lesquelles fonctionnent les systèmes de recommandation. En effet, nous considérons que seulement 1% des données sont pertinentes pour l'utilisateur et essayons à l'aide de métriques adaptées de voir si le model a considéré l'item le plus susceptible d'être pertinent à l'utilisateur plus que ceux qui ne le sont pas.

A noter que les couples (utilisateur, item) présents dans les données de test ne le sont pas dans les données d'entraînement.

- **Deuxième type d'évaluation (utilisateurs inconnus) :**

- ***Les données d'entraînement :***

Nous sélectionnons au hasard 80% des utilisateurs ainsi que tous les items avec lesquels ils ont interagi puis nous y ajoutons les instances négatives de ces mêmes utilisateurs.

- ***Les données de test :***

Nous prenons 20% des utilisateurs restants puis y ajoutons les instances de la même manière que la première approche d'évaluation. Ainsi, le modèle ne connaîtra pas les utilisateurs lors de l'évaluation, nous pourrons ainsi simuler le cas de la recommandation d'items à de nouveaux utilisateurs et voir si le problème de démarrage à froid a été corrigé.

3.2 - Entraînement de NHybF

3.2.1 - Fonction de coût (*loss function*)

Les réseaux de neurones sont entraînés en utilisant un processus d'optimisation qui requiert une fonction de coût à minimiser. Cette fonction calcule la différence (l'erreur) entre la prédiction faite par le modèle et la valeur réelle qu'il était censé prédire.

Dans cette approche tout comme dans celle de **He et al [5]** nous avons trois modèles (GMF, HybMLP, NeuHybMF) à entraîner et dans chacun d'entre eux la fonction utilisée est *binary cross-entropy* ou bien *log loss* car il s'agit d'un problème de Régression Logistique (*Logistic Regression*) où nous faisons une classification binaire, cette fonction est la plus adaptée à ce type de problèmes. La formule de la fonction est la suivante [5] :

$$L = -\sum_{(u,i)\in Y\cup Y^-} y_{ui} log\ \hat{y}_{ui} + (1 - y_{ui})\ log(1 - \hat{y}_{ui}) \quad (2.5)$$

Où :

y_{ui} : Valeur de l'interaction entre l'utilisateur *u* et l'item *i*.
$\hat{y}_{ui}$: Prédiction de l'interaction entre l'utilisateur *u* et l'item *i*.
Y : Ensemble d'instances positives.
Y^- : Ensemble d'instances négatives.

3.2.2 - Optimisation

Des algorithmes d'optimisation sont aussi utilisés pour mettre à jour les valeurs des poids du modèle. Nous utilisons la même approche présentée dans le travail d'**He et al [5]**, ce qui nous amène à l'utilisation d'*Adam* (*Adaptive Moment Estimation)* pour l'entraînement de GMF et HybMLP. Et *SGD* (*Stochastic Gradient Descent*) pour l'entraînement de NeuHybMF.

La différence entre Adam et SGD est que le premier adapte le taux d'apprentissage à chaque paramètre, par contre le deuxième utilise un seul taux d'apprentissage pour mettre à jour les paramètres. Les deux méthodes ajustent les poids pour un groupe de donnée (*batch*) traité en même temps. Ce qui fait que nous utilisons SGD pour entraîner NeuHybMF[11] est que nous avons déjà les paramètres pré-entraînés de GMF et HybMLP.

4 - Conclusion

Après avoir expliqué les détails du système de recommandation hybride basé sur les RNs. Nous présenterons dans le prochain chapitre les détails d'implémentation et d'évaluations, ainsi que la comparaison avec d'autres méthodes déjà existantes dans la littérature.

[11] **NeuHybMF :** le nom du modèle étant un peu long, nous utiliserons le terme NHybF pour le décrire lors des évaluations du chapitre 3.

CHAPITRE 3 :

IMPLEMENTATION ET EXPERIMENTATION

1 - Introduction

Dans ce chapitre, nous présentons les détails liés à l'implémentation et évaluation de notre système de recommandation hybride NHybF (*Neural Hybrid Filtering*) combinant le FC et le filtrage cognitif et incluant les deux modèles GMF et HybMLP.

Ce chapitre est composé de trois sections, la première où nous présentons les outils utilisés pour le développement des différents algorithmes. Une deuxième où nous détaillerons les métriques utilisées pour l'évaluation des modèles. Enfin, les résultats et comparaisons entre les modèles seront présentés selon deux types d'évaluation comme mentionné dans le chapitre 2 (évaluation avec des utilisateurs connus et avec nouveaux utilisateurs dans le test).

2 – Notre système de recommandation

2.1 - Environnement de développement

Pour le développement des approches présentées précédemment, nous avons utilisé le langage de programmation *Python* version 3.6. Le framework de *deep learning* utilisé pour développer les modèles est *Keras*[12] version 2.2.4 avec *Tensorflow*[13] (version 1.13.1) comme *backend*. En ce qui concerne la lecture et transformation de données, nous avons utilisé la bibliothèque *Pandas*[14] version 0.24.2. Pour l'affichage des graphes nous avons utilisé *Matplotlib*[15] version 3.1.0. Les détails d'utilisation de ces bibliothèques et de leurs méthodes sont expliqués en annexe (A.2).

L'utilisation d'un GPU (*Graphic Processing Unit*) est préférable pour entraîner les modèles compte tenu de sa rapidité comparée au CPU. Pour cela nous avons utilisé un outil gratuit que Google a mis à disposition de tout programmeur ne disposant pas de carte graphique puissante sur son ordinateur. Il s'agit de *Colaboratory*[16], un notebook où l'on passe un code et l'exécution se fait avec les processeurs de Google, puis les résultats nous sont retournés sur la console du notebook. Seulement les versions 2.7 et 3.6 de *Python* sont supportées, il y a possibilité de choisir le type d'accélérateur à utiliser (aucun - i.e. CPU -, GPU - Tesla K80 -

[12] https://keras.io/
[13] https://www.tensorflow.org/
[14] https://pandas.pydata.org/
[15] https://matplotlib.org/3.1.0/
[16] https://colab.research.google.com/

ou TPU[17]). Il est possible de faire le lien entre le notebook et Google Drive[18] ou GitHub[19] pour enregistrer ou importer des fichiers de données.

2.2 - Description des modules de notre système de recommandation

Pour permettre une manipulation facile à entraîner notre système de recommandation, nous avons mis en place une application développée en *Java*, utilisant *JavaFX* pour la création de la GUI.
Un environnement virtuel python est nécessaire. Il devra contenir toutes les bibliothèques mentionnées dans la section précédente, c'est-à-dire *Pandas*, *NumPy*, *Matplotlib*, *Keras* et *Tensorflow*. La figure ci-dessous (figure 3.1) représente la fenêtre principale de l'application incluant tous les modèles développés tout au long de ce travail.

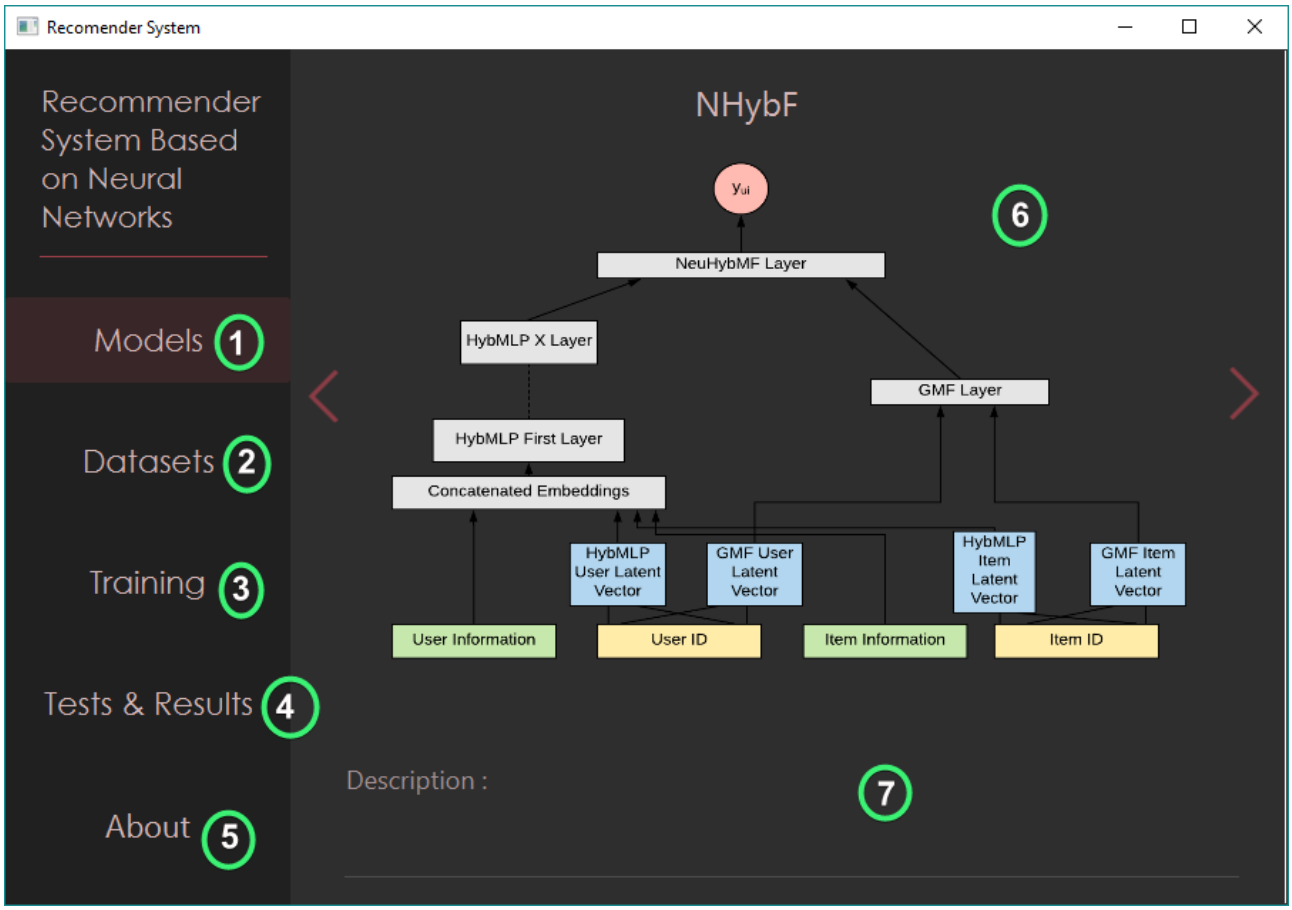


Figure 3.1 : Implémentation de notre système de recommandation – Fenêtre principale.

Description des fonctionnalités décrites dans la figure 3.1 :
1 : Page où nous avons présenté les modèles de NHybF (GMF, HybMLP et NeuHybMF).

[17] **TPU :** Tensor Processing Unit, circuit intégré spécifique pour une application (ASIC) développé par Google en 2016 utilisé dans les RNs et le Machine Learning.
[18] https://www.google.com/drive/
[19] https://github.com/

2 : Nous permettons dans cette section à l'utilisateur de choisir un des trois *datasets* décrits dans la section 4 de ce chapitre (MovieLens 100k, MovieLens 1m et Yelp) et en créer un *trainset* et un *testset* avec des données implicites.
3 : Paramètres d'entraînement des modèles. Il est permis à l'utilisateur de personnaliser ces réglages et de choisir la structure à entraîner.
4 : Affichage des tests et résultats du modèle entraîné.
5 : Informations à propos du développement du système de recommandation.
6 : Le schéma d'un des trois modèles est affiché.
7 : Une brève description du modèle est affichée.

Afin de procéder au test de notre système de recommandation, il faut tout d'abord générer les fichiers *train* et *test* (voir figure 3.2). Trois *dataset* sont disponibles, il faut donc choisir le nombre d'instances négatives par rapport aux positives et choisir le chemin pour enregistrer les fichiers.

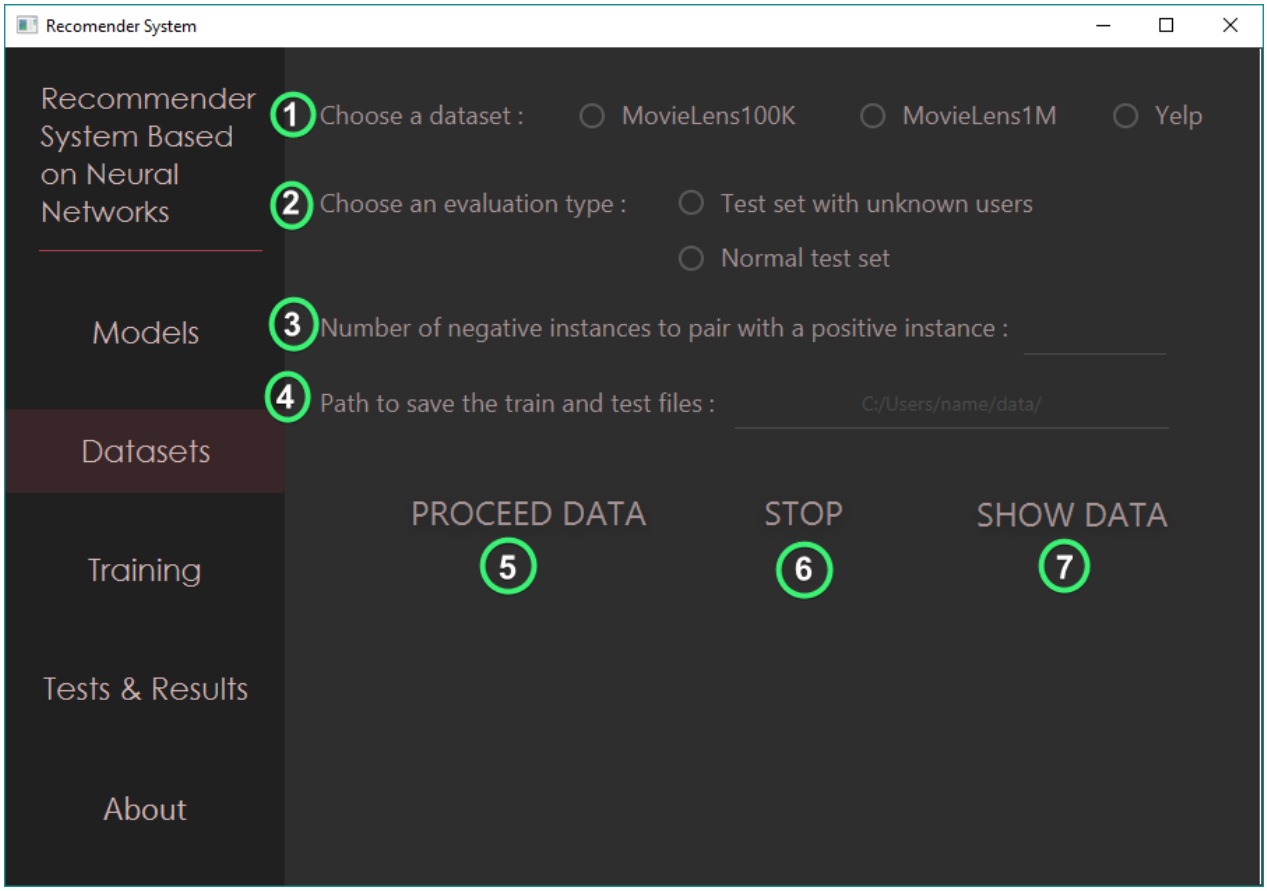


Figure 3.2 : Génération des jeux d'entraînement et de tests.

Description des fonctionnalités décrites dans la figure 3.2 :
1 : Sélectionner le jeu de données.
2 : Sélectionner le type d'évaluation qui détermine le découpage du *dataset* qui détermine la manière dont sera découpé ce dernier.
3 : Indiquer le nombre d'instances négatives par instance positive.

4 : Indiquer le chemin du dossier dans lequel les fichiers *train* et *test* seront sauvegardés.
5 : Lancer le traitement du *dataset* sélectionné.
6 : Arrêter le processus.
7 : Afficher les fichiers *train* et *test*.

Après avoir généré les fichiers d'entraînement et de test, vous pouvez procéder à l'entraînement des modèles. Dans la première page, vous devrez sélectionner un des modèles de NHybF à entraîner avec tous les paramètres qui lui sont relatifs. Dans la seconde, les modèles NCF et MLP peuvent être entraînés. Si aucun paramètre n'est défini, l'entraînement sera lancé avec des valeurs données par défaut. Vous pouvez aussi activer l'*early stopping*[20].

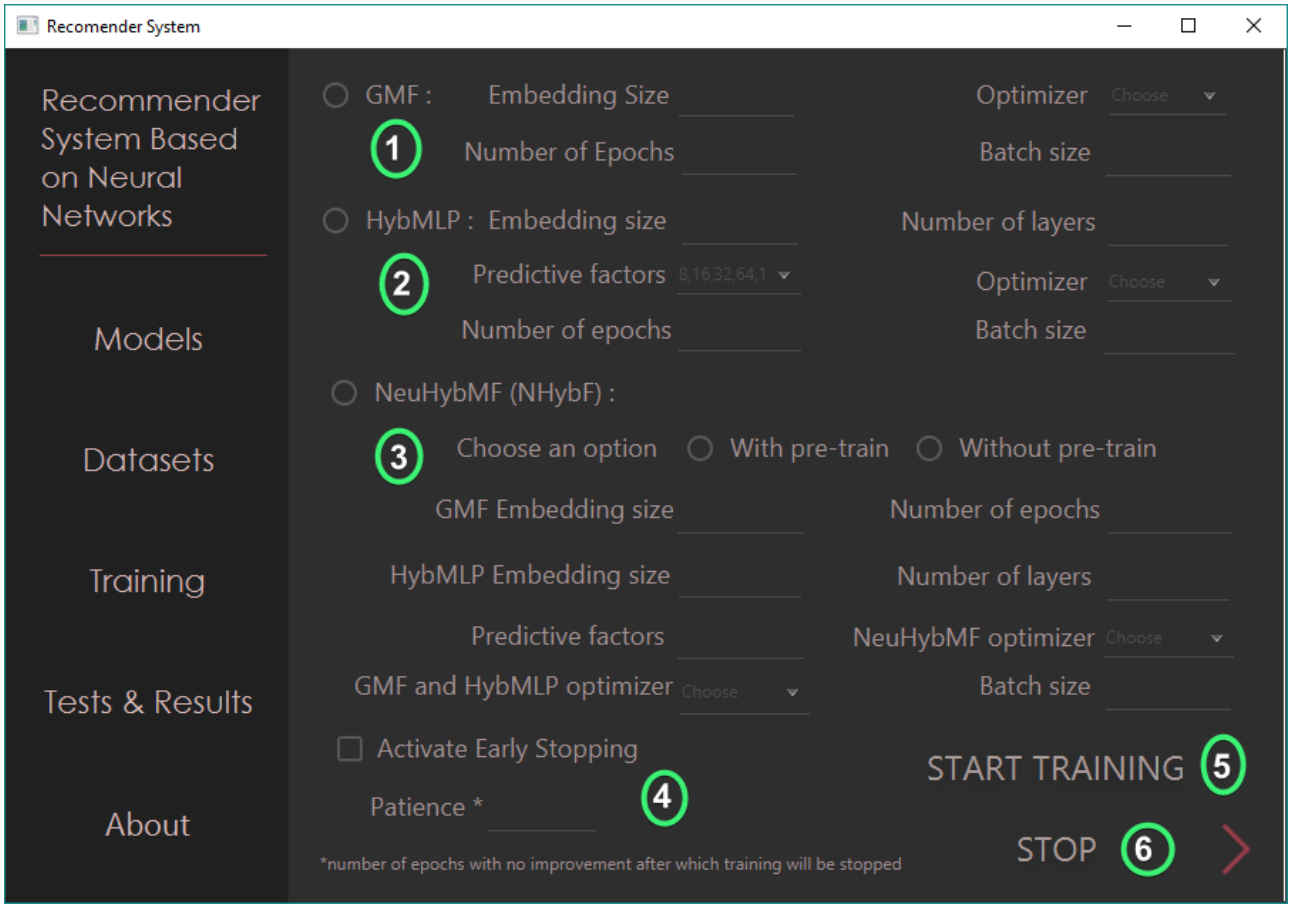


Figure 3.3 : Lancement de l'entraînement des modèles.

Description des fonctionnalités décrites dans la figure 3.3 :
1 : Sélectionner le modèle GMF et remplir les informations le concernant pour pouvoir lancer son entraînement.
2 : Sélectionner le modèle HybMLP et remplir les informations le concernant pour pouvoir lancer son entraînement.
3 : Sélectionner le modèle NHybF et remplir les informations le concernant pour pouvoir lancer son entraînement.

[20] **Early Stopping :** se charge d'arrêter l'entraînement après un certain nombre d'*epoch* si l'entraînement ne montre plus aucune évolution.

4 : Indiquer si l'*early stopping* sera activé ou pas, si c'est le cas alors mentionner le nombre d'*epoch* après lequel peut s'arrêter l'entraînement.
5 : Lancer le traitement l'entraînement.
6 : Arrêter le processus.

Les fichiers contenant les poids, structures et résultats des métriques des modèles durant l'entraînement seront enregistrés dans le dossier spécifié précédemment.
Enfin, la dernière page « Tests & Results » permet d'afficher les résultats que nous avons obtenus avec notre entraînement et les résultats de l'entraînement lancé à partir de l'application (lancé à partir de la GUI).

3 - Métriques d'évaluation

Comme mentionné précédemment (chapitre 2), nous auront deux types d'évaluations, une classique où nous avons connaissance des utilisateurs existants et une autre où ce n'est pas le cas, afin de tester si notre approche corrige plus ou moins le démarrage à froid du système de recommandation.

Ainsi expliqué dans la partie de la gestion du *dataset*, nous avons fait en sorte de simuler les conditions réelles des bases de données (peu d'items pertinents pour l'utilisateur) où nous considérons que seulement un item sur cent pourrait être pertinent pour l'utilisateur. Deux métriques utilisées dans l'article d'**He et al [5]** que nous allons présenter s'avèrent les plus appropriées à notre cas.

Remarque :
Avant d'expliquer le fonctionnement des métriques, il est important d'expliquer un point sur le fonctionnement de la fonction d'activation de la dernière couche, *Sigmoid.*

Comme mentionnée précédemment (formule 2.2), la fonction retourne un nombre entre 0 et 1, ce qui est parfait vu que notre problème est un problème de classification binaire qui est de prédire l'interaction d'un utilisateur avec un item. Ce nombre retourné peut être considéré comme étant la probabilité de cette interaction, par exemple, si le chiffre retourné est 0.99 la probabilité de l'interaction serait considérée comme forte alors qu'à l'inverse, si c'était 0.03 cette dernière serait considérée comme faible.

Ainsi, nous pouvons, pour chaque utilisateur, trier les items selon la probabilité d'interaction prédite dans l'ordre décroissant et ainsi avoir les items les plus recommandés en tête de liste. Là, nous pourrons voir si le model a considéré l'item le plus susceptible d'être pertinent à l'utilisateur (celui avec lequel il y a eu interaction) plus que ceux qui ne le sont pas (les 99 autres) ainsi qu'à quel degré le model a considéré l'item.

3.1 - Hit Ratio (HR@k)

Cette métrique permet de compter le nombre de *"Hit"*. Ainsi, pour chaque utilisateur, il s'agit de prendre les k items qui lui sont les plus recommandés par le modèle et chercher si l'item pertinent figure dans la liste. Si c'est le cas, le score est de 1 et 0 sinon. Puis, il s'agit de répéter le calcul pour tous les utilisateurs et calculer la moyenne.

3.2 - Normalized Discounted Cumulative Gain (NDCG@k)

L'objectif est que le modèle soit le plus précis possible dans la prédiction de la probabilité de l'interaction entre l'utilisateur et l'item. Une manière d'évaluer cette prédiction est de prendre en considération la position de l'item pertinent par rapport aux autres dans la liste des items recommandés, c'est ce que fait la métrique *NDCG*, sa formule est la suivante :

$$NDCG@k = \sum_{i}^{k} \frac{Pertinence(item_i)}{log_2(i+1)} \quad (3.1)$$

Où:

k : Taille de la liste d'items recommandés.
i : $Position\ de\ l'item \in [1, k]$.
$Pertinence(x) \in \{0, 1\}$.

Comme pour le score HR, nous sélectionnons les k items les plus pertinents prédits pour chaque utilisateur et calculons leur score NDCG. Etant donné que la pertinence d'un item est booléenne et qu'un seul item est pertinent pour chaque utilisateur nous pouvons en conclure que $max\ (NDCG) = 1$ quand l'item pertinent est en première position et $min(NDCG) = 0$ quand il ne figure pas dans les *k* items recommandés ($NDCG \in [0, 1]$).

Nous calculons les scores $NDCG@k$ pour les items recommandés à chaque utilisateur puis calculons leur moyenne que nous considérons comme étant le score $NDCG@k$ du modèle.

4 - Evaluation des modèles

Afin de procéder à l'évaluation du modèle et sa comparaison aux méthodes de l'état de l'art, nous utilisons trois jeux de données accessibles au grand public :

- **MovieLens 100K[21] (ML-100K) :** contient 100 000 évaluations, récoltées à partir d'interactions de 943 utilisateurs avec 1682 films. En plus des détails concernant ces utilisateurs et items, tels que l'âge, le sexe et l'occupation de la personne. Le genre, le titre et la date de sortie des films ainsi que d'autres informations.
- **MovieLens 1M[18] (ML-1M) :** contient 1 000 209 évaluations, récoltées à partir d'interactions de 6040 utilisateurs avec approximativement 3900 films. Le même type d'informations qui concernent les utilisateurs et items dans la base ML-100K sont mis à disposition.
- **Yelp[22] :** nous prenons un échantillon de la base collecté et traité par Belkacem et Ouafi [15]. Cet échantillon contient 110894 évaluations effectuées par 5436 utilisateurs sur 4733 restaurants. Des informations sur les utilisateurs et les restaurants sont aussi mis à disposition. Un prétraitement a été fait sur cette base pour transformer les données, comme expliqué dans le chapitre 2, en vecteurs exploitables.

[21] https://grouplens.org/datasets/movielens/
[22] https://www.yelp.com/dataset/

Le tableau suivant résume les informations relatives aux jeux de données utilisés :

Tableau 3.1 : Statistiques relatives aux jeux de données.

Dataset	**# d'utilisateurs**	**# d'items**	**# d'évaluations**	**Densité**
ML-100K	943	1682	100 000	6.3%
ML-1M	6040	3952	1 000 209	4.19%
Yelp	5436	4733	110 894	0.43%

La densité est calculée à partir de la formule suivante :

$$D = \frac{nombre\ d'évaluations\ contenues\ dans\ le\ dataset}{nombre\ d'utilisateurs \times nombre\ d'items} \quad (3.2)$$

Nous pouvons remarquer du calcul de la densité des *datasets* que les matrices d'évaluations sont très creuses.

Notre objectif par rapport à l'évaluation est de discuter les deux aspects suivants :

1 - L'efficacité du *deep learning* dans la prédiction de l'interaction.

2 - Amélioration par rapport aux méthodes de l'état de l'art en ce qui concerne le démarrage à froid.

4. 1 - Evaluation avec utilisateurs et items connus

4.1.1 – Evaluation selon différents critères pour déterminer les meilleurs paramètres des modèles

Pour évaluer le modèle proposé, il est important d'étudier l'évolution des modèles le composant ayant subi un enrichissement, c'est-à-dire : HybMLP et NeuHybMF (qu'on notera NHybF dans ce qui suit).

Les tableaux ci-dessous contiennent les valeurs des métriques HR et NDCG du modèle donné. Nous évaluons le TOP 10 de chaque utilisateur après un certain nombre d'*epochs*[23] puis calculons la moyenne des HR et NDCG de tous les utilisateurs par itération, et assignons la valeur de la meilleure *epoch* au modèle. À noter que pour cette évaluation les jeux d'entraînement des différents *datasets* sont équilibrés (le nombre d'instances négatives est égal à celui des instances positives).

[23] **Epoch :** une itération sur tout le jeu de données.

1) Evaluation de HybMLP

Tableau 3.2 : Evaluation du modèle HybMLP sur ML-100K avec différentes tailles d'*embedding* différents nombre de couches cachées – *Predictive factors* = 16.

MovieLens 100K												
N. C.[24]	**HybMLP-0**		**HybMLP-1**		**HybMLP-2**		**HybMLP-3**		**HybMLP-4**		**HybMLP-5**	
T. E.[25]	**HR @10**	**NDC G@10**	**HR@ 10**	**NDC G@10**	**HR@ 10**	**NDC G@10**	**HR@ 10**	**NDC G@10**	**HR@ 10**	**NDC G@10**	**HR@ 10**	**NDC G@10**
8	0.57	0.329	0.553	0.313	0.559	0.32	0.547	0.311	**0.571**	**0.321**	0.569	0.326
16	0.57	0.327	0.559	0.314	**0.573**	**0.323**	0.568	0.328	0.552	0.318	0.571	0.326
32	0.57	0.322	0.553	0.313	0.553	0.317	0.566	0.32	0.564	0.322	**0.567**	**0.322**
64	0.57	0.329	0.552	0.313	0.552	0.318	0.553	0.328	0.564	0.325	**0.569**	**0.32**
128	0.57	0.329	**0.596**	**0.349**	0.554	0.322	0.531	0.302	0.55	0.314	0.559	0.316

Du tableau il est clair qu'il n'y a pas amélioration, voir une diminution de la précision et des valeurs des métriques avec l'augmentation du nombre de couches et de facteurs.

Compte-tenu du sur-apprentissage qui a lieu avec le jeu de données ML-100K et l'incapacité d'arriver à de bons résultats, nous procédons à la même évaluation mais avec le second jeu de données plus volumineux (ML-1m).

Tableau 3.3 : Evaluation du modèle HybMLP sur ML-1m après 10 epochs avec différents nombres de facteurs (taille de l'*embedding*) – *Predictive factors* = 16.

MovieLens 1m												
N. C.	**HybMLP-0**		**HybMLP-1**		**HybMLP-2**		**HybMLP-3**		**HybMLP-4**		**HybMLP-5**	
T. E.	**HR @10**	**NDC G@10**	**HR @10**	**NDCG @10**	**HR @10**	**NDC G@10**	**HR @10**	**NDC G@10**	**HR @10**	**NDC G@10**	**HR @10**	**NDC G@10**
8	0.563	0.334	0.717	0.445	0.733	0.462	0.748	0.477	0.755	0.482	**0.758**	**0.489**
16	0.566	0.334	0.719	0.448	0.747	0.482	0.766	0.499	0.766	0.5	**0.773**	**0.51**
32	0.568	0.334	0.741	0.472	0.756	0.486	0.768	0.498	0.781	0.512	**0.789**	**0.523**
64	0.566	0.334	0.751	0.481	0.762	0.492	0.773	0.506	0.78	0.514	**0.788**	**0.521**
128	0.569	0.335	0.752	0.484	0.766	0.581	0.773	0.508	0.779	0.515	**0.782**	**0.522**

[24] **N. C.** : Nombre de couches.
[25] **T. E.** : Taille de l'*embedding*.

Après comparaison des résultats de l'entraînement et tests avec la base de données ML-100K, les résultats obtenus avec ML-1m confirment que les modèles de *deep learning* ont de meilleures performances lorsque le jeu de données d'entraînement est large (donc plus de données d'apprentissage).

Les figures ci-dessous (figure 3.2 et 3.3) illustrent l'évolution de la prédiction selon le nombre de couche des deux modèles MLP et HybMLP (pour plus de détails sur les valeurs des métriques voir les tableaux A3.1 et A3.8 en annexe).

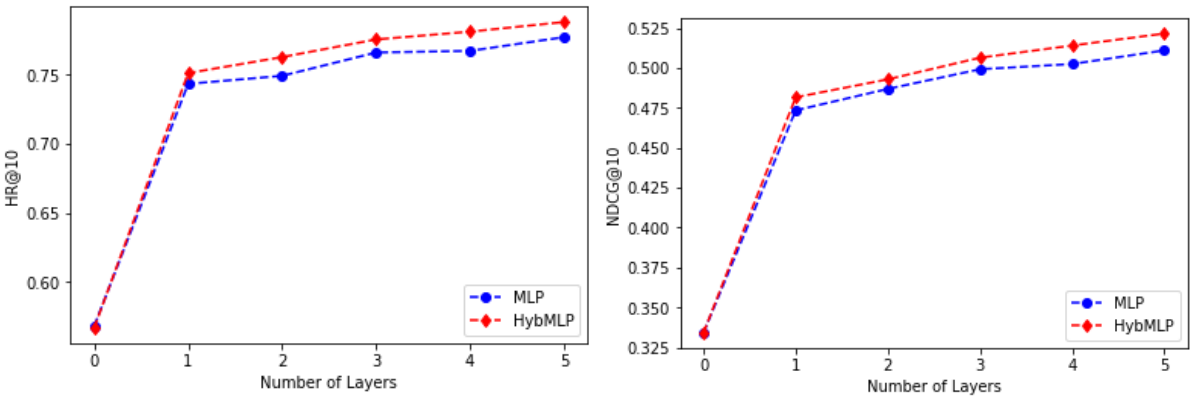


Figure 3.4 : Evolution de HybMLP selon le nombre de couches – ML-1m.

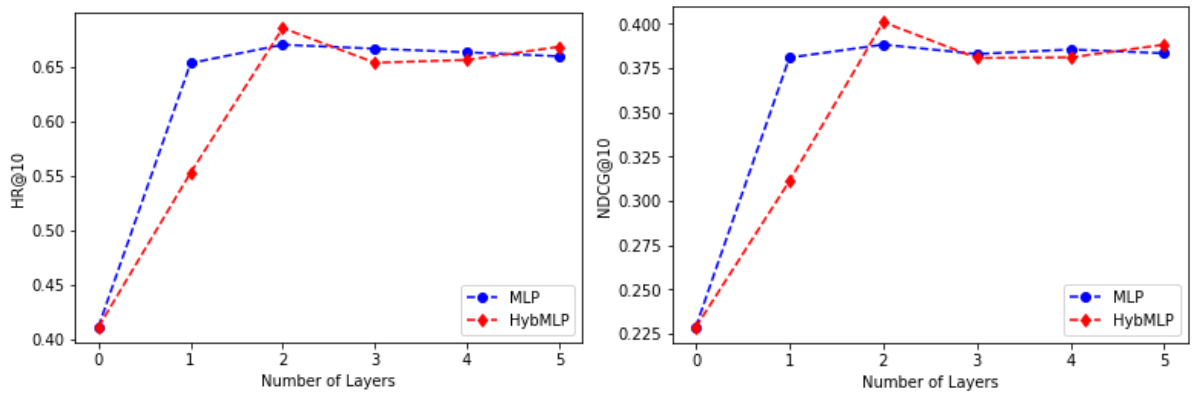


Figure 3.5 : Evolution de HybMLP selon le nombre de couches - Yelp.

Les résultats obtenus avec 5 couches cachées justifient l'utilisation d'un réseau de neurones profond au lieu d'un réseau simple (HybMLP 0) : nous avons obtenu une précision HR = 0.41 et HR=0.57 respectivement pour Yelp et ML-1m lorsqu'il n'y avait pas de couche cachées pour les traitements. Mais lorsque ce nombre augmente les prédictions et valeurs des métriques s'améliorent indéniablement. Il faudrait quand même prendre en considération le sur-apprentissage qui risque de fausser les résultats si le nombre de couches est trop élevé pour les données d'entraînement.

Les figures ci-dessous (figure 3.6 et 3.7) illustrent les valeurs des métriques HR et NDCG lorsque la taille de l'*embedding* est variée (pour plus de détails sur les valeurs des métriques voir les tableaux A3.2 et A3.9 en annexe).

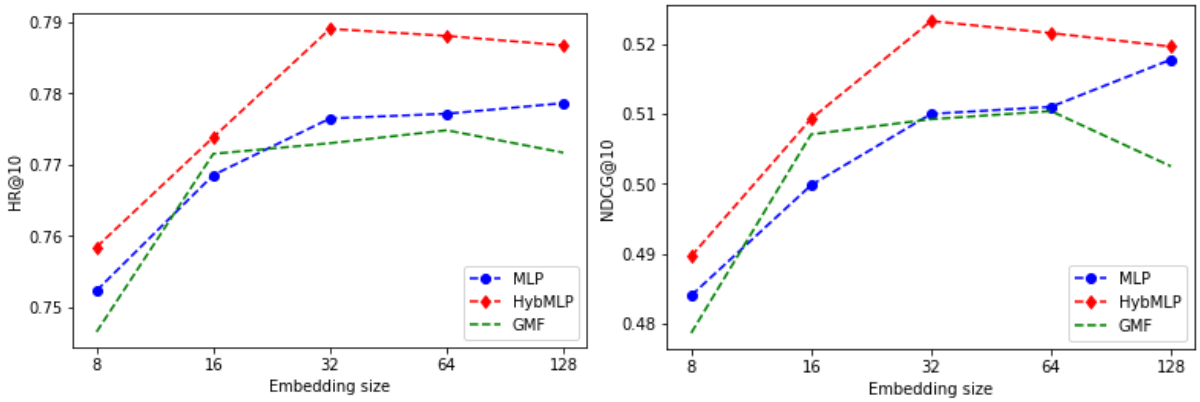


Figure 3.6 : Evaluation selon la taille des *embeddings* de différents modèles - ML-1m.

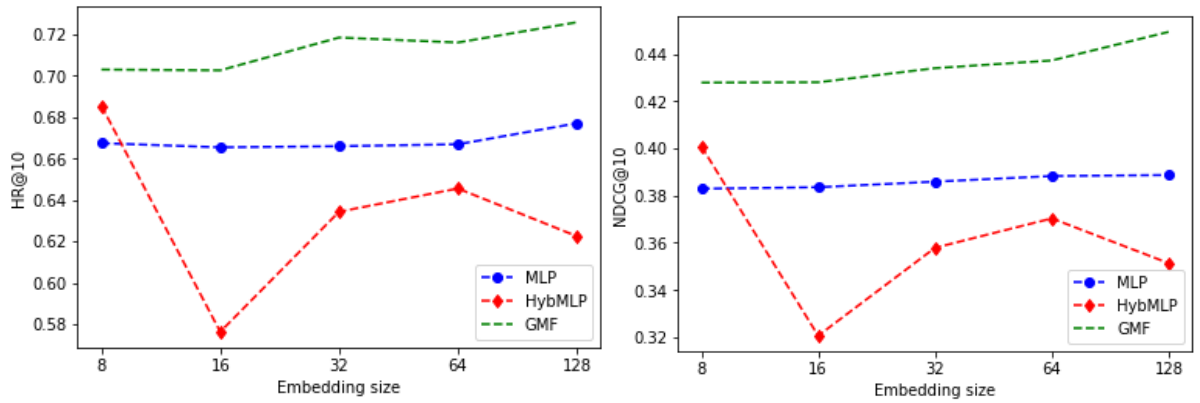


Figure 3.7 : Evaluation selon la taille des *embeddings* de différents modèles - Yelp.

Nous avons procédé à cette évaluation pour trouver la taille optimale de l'*embedding* (le nombre de facteurs latents optimal) pour chaque modèle et chaque *dataset*. Le nombre de facteurs latents le plus optimal pour ML-1m est de 32 facteurs. Par contre, pour Yelp la taille la plus optimale est de 8 facteurs, car nous avons beaucoup plus de données relatives aux items et utilisateurs à traiter en comparaison avec MovieLens.

2) Evaluation de NHybF

Nous avons ici deux manières d'entraîner NHybF comme cité dans l'article d'**He et al** [5]. La première consiste en l'entraînement du modèle au complet c'est-à-dire GMF et HybMLP en même temps, et la seconde propose d'entraîner HybMLP et GMF séparément, puis entraîner la dernière couche de concaténation après avoir récupéré les poids des deux modèles.

a) Sans pré-entraînement :

Tableau 3.4 : NHybF sans pré-entraînement de GMF et HybMLP après 10 epochs. ML-1m.

MovieLens 1m												
N. C.	**HybMLP-0**		**HybMLP-1**		**HybMLP-2**		**HybMLP-3**		**HybMLP-4**		**HybMLP-5**	
T. E.	**HR@ 10**	**NDCG @10**	**HR@ 10**	**NDC G@10**	**HR @10**	**NDC G@10**	**HR@ 10**	**NDC G@10**	**HR @10**	**NDCG @10**	**HR @10**	**NDC G@10**
8	0.557	0.331	0.459	0.256	0.495	0.284	0.41	0.223	0.457	0.263	0.383	0.211
16	0.554	0.329	0.505	0.292	0.474	0.281	0.523	0.311	0.437	0.257	0.359	0.198
32	0.556	0.331	0.547	0.323	0.547	0.324	0.515	0.305	0.527	0.312	0.424	0.233
64	0.559	0.331	0.537	0.312	0.555	0.328	0.541	0.322	0.542	0.322	0.512	0.305
128	0.558	0.33	0.547	0.325	0.56	0.331	0.552	0.331	0.55	0.328	0.528	0.315

Le modèle n'arrive pas à faire de bonnes prédictions, car il s'agit d'une combinaison de deux autres modèles qui apprennent en même temps à prédire la même chose.

b) Avec pré-entraînement :

Tableau 3.5 : NHybF avec pré-entraînement de GMF et HybMLP après 10 epochs - ML-1m.

MovieLens 1m												
N. C.	**HybMLP-0**		**HybMLP-1**		**HybMLP-2**		**HybMLP-3**		**HybMLP-4**		**HybMLP-5**	
T. E.	**HR@ 10**	**NDC G@10**	**HR@1 0**	**NDCG @10**	**HR@ 10**	**NDC G@10**	**HR@ 10**	**NDC G@10**	**HR@ 10**	**NDC G@10**	**HR @10**	**NDC G@10**
8	0.744	0.478	0.75	0.488	0.757	0.491	0.759	0.496	0.762	0.498	**0.765**	**0.503**
16	0.773	0.507	0.774	0.511	0.777	0.519	0.787	0.525	0.785	0.524	**0.789**	**0.527**
32	0.768	0.509	0.772	0.513	0.78	0.518	0.79	0.527	0.791	0.528	**0.796**	**0.534**
64	0.776	0.512	0.78	0.518	0.782	0.524	0.791	0.529	0.794	0.531	**0.801**	**0.536**
128	0.772	0.5	0.78	0.521	0.79	0.527	0.792	0.529	0.792	0.53	**0.791**	**0.535**

Les résultats avec le pré-entraînement sont nettement supérieurs par rapport aux précédents, ce qui prouve son efficacité dans l'entraînement de NHybF, et c'est ce qui a été utilisé pour les tests qui suivent.

La figure (figure 3.8) ci-dessous montre les prédictions selon le nombre de nœuds dans la dernière couche cachée (*predictive factors*) de HybMLP (pour plus de détails sur les valeurs des métriques voir le tableau A3.3 en annexe).

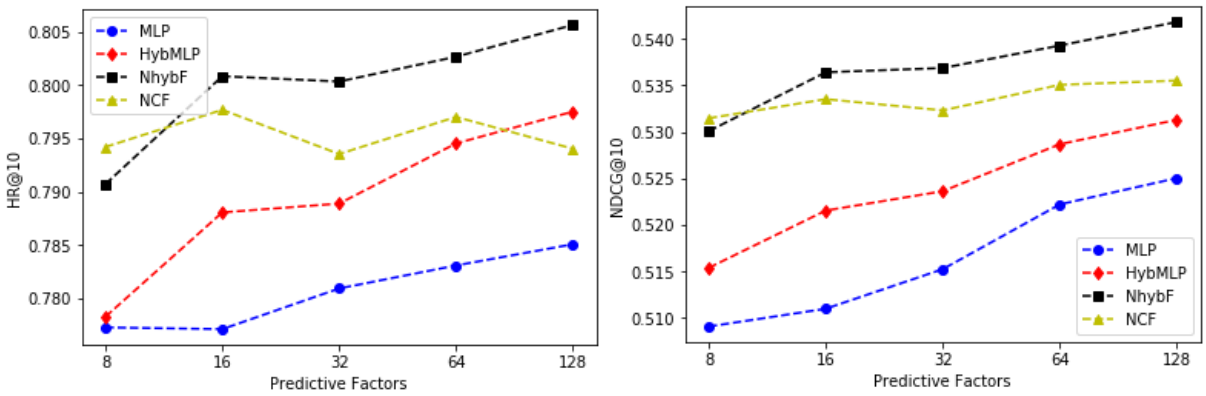


Figure 3.8 : Performance d'HR@10 et NDCG@10 selon le nombre de *predictive factors* - ML-1m.

Le nombre de nœuds des couches d'un modèle est important à prendre en considération, si ce nombre est petit les résultats ne risquent pas d'être très fameux car les données ne seront pas assez traitées (transformées en passant par des fonctions d'activation des nœuds d'autres couches cachées). Par contre, un nombre élevé de nœuds par couche peut mener à un sur-apprentissage (selon la taille du jeu de données).

4.1.2 – Brouillage du *dataset*

Afin de voir l'efficacité du modèle nous avons procédé au brouillage des données d'apprentissage, c'est à dire augmenter le nombre d'instances négatives par rapport aux positives. Et cela dans le but de trouver le nombre d'instances négatives adéquats pour permettre au modèle de donner une meilleure précision.

Les tests suivants ont été effectués sur les meilleurs modèles NHybF et HybMLP obtenus à partir de l'entraînement sur le *dataset* équilibré (voir tableau 3.6).

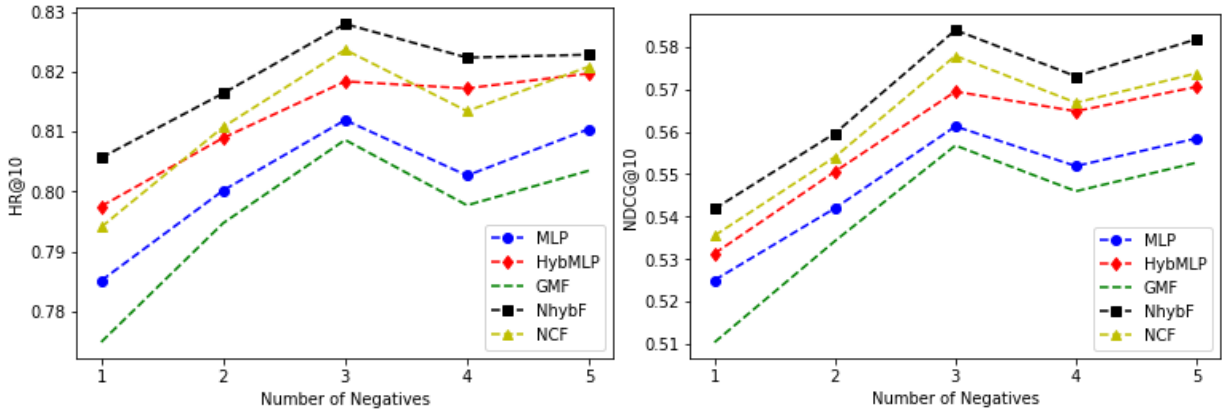


Figure 3.9 : Performance des modèles selon le nombre d'instances négatives par instance positive - ML-1m.

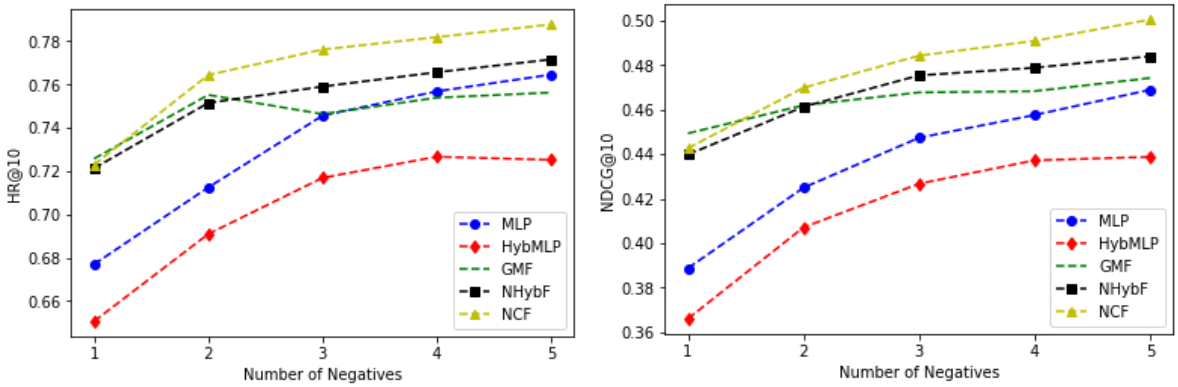


Figure 3.10 : Performance des modèles selon le nombre d'instances négatives par instance positive - Yelp.

Des figures ci-dessus (figures 3.9 et 3.10) nous pouvons déduire que notre modèle fonctionne mieux lorsque nous avons un *dataset* composé de trois instances négatives par instance positive pour ML-1M, et avec cinq instances négatives pour Yelp (pour plus de détails sur les valeurs des métriques voir les tableaux A3.4 et A3.10 en annexe).

4.1.3 - Evaluation du Top K de la recommandation d'items

Les figures 3.11, 3.12, 3.13 et 3.14 ci-dessous illustrent l'évaluation des Top-K listes d'items recommandés, avec K allant de 1 à 10 (pour plus de détails sur les valeurs des métriques voir les tableaux A3.5 et A3.11 en annexe).

Ces évaluations ont été effectuées sur la meilleure structure de chaque modèle comme expliqué dans le tableau suivant :

Tableau 3.6 : Structures des meilleurs modèles.

Dataset	Paramètres	T. E.	*Predictive factors*	Nombre de couches	Nombre d'instances négatives
ML-1m	**GMF**	128	/	/	3
	MLP	64	128	5	3
	HybMLP	64	128	5	3
Yelp	**GMF**	128	/	/	5
	MLP	64	64	5	5
	HybMLP	8	32	5	5

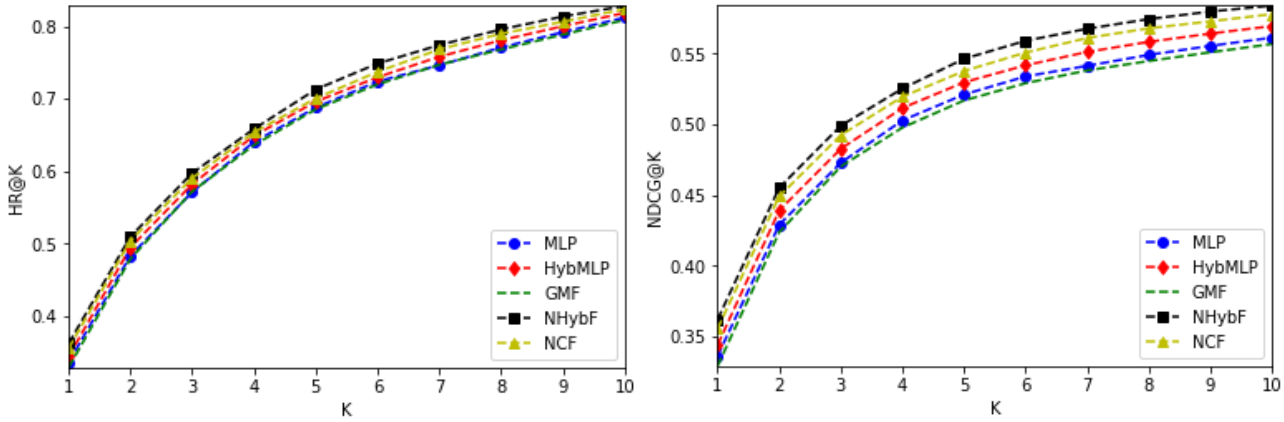


Figure 3.11 : Evaluation du Top-K de recommandation d'items avec K variant de 1 à 10 - ML-1m.

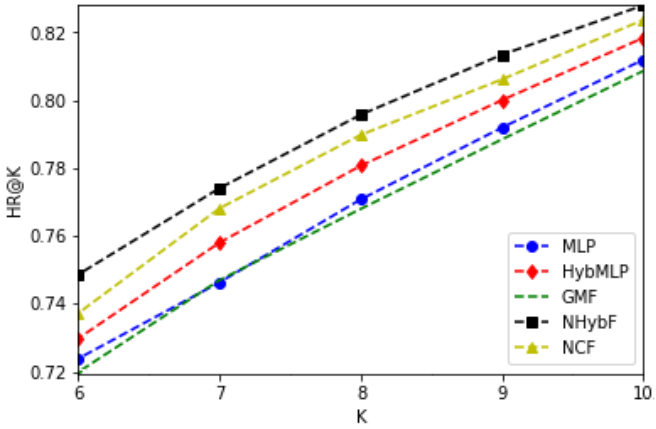


Figure 3.12 : Evaluation du Top-K de recommandation d'items avec K variant de 6 à 10 - ML-1m.

Les modèles arrivent à surpasser les approches de l'état de l'art dans le cas de la base ML-1m. Contrairement à Yelp où le filtrage collaboratif à lui seul arrive à mieux prédire les items pertinents.

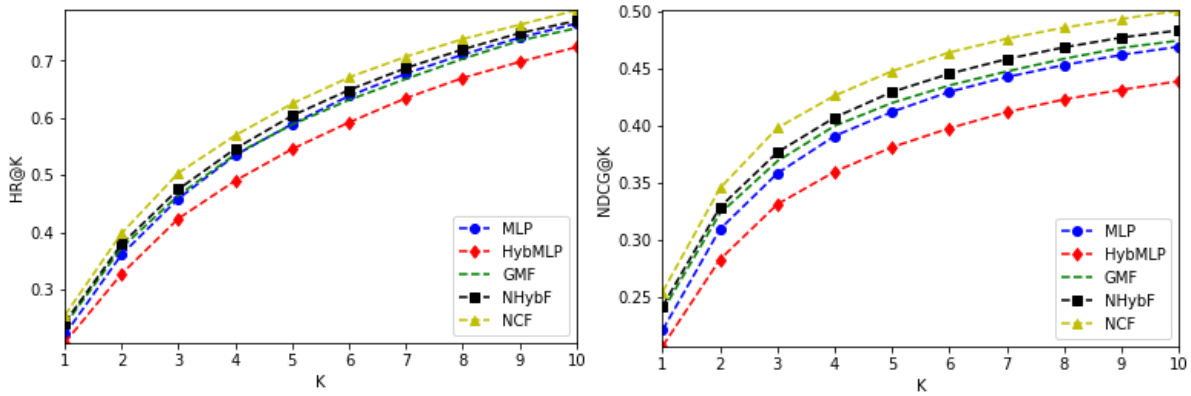


Figure 3.13 : Evaluation du Top-K de recommandation d'items avec K variant de 1 à 10 - Yelp.

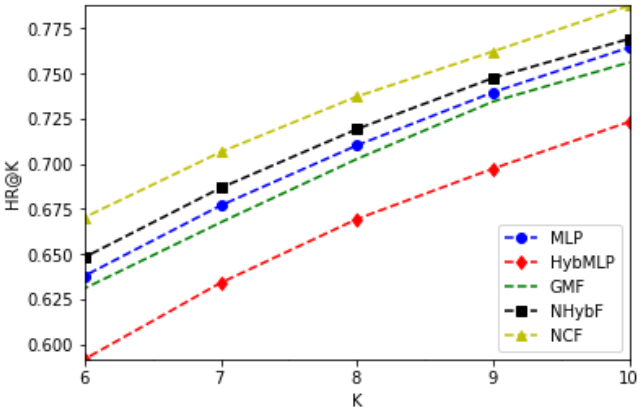


Figure 3.14 : Evaluation du Top-K de recommandation d'items avec K variant de 6 à 10 – Yelp.

4. 2 - Evaluation avec des utilisateurs inconnus

Nous prenons les meilleures structures des modèles entraînés précédemment et les entraînons de nouveau sur le *dataset* modifié où les utilisateurs des données d'entraînement ne sont pas les mêmes que ceux des données de test. L'approche proposée par **He et al [5]** devra donc se résoudre à faire un filtrage collaboratif basé items vu que ce sont ces derniers que le modèle sera en mesure de reconnaître.

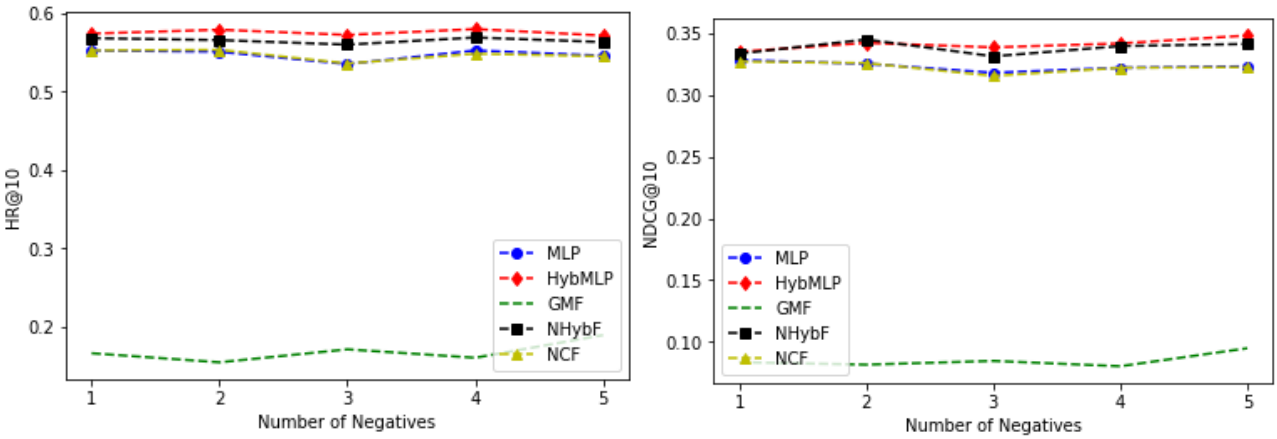


Figure 3.15 : Performance des modèles selon le nombre d'instances négatives par instance positive (avec GMF) - ML-1m.

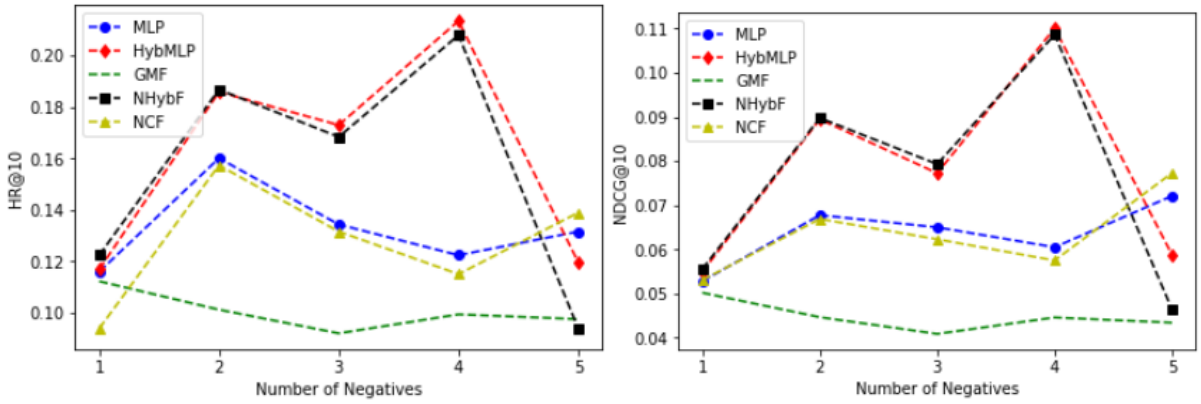


Figure 3.16 : Performance des modèles selon le nombre d'instances négatives par instance positive (avec GMF) - Yelp.

Pour plus de détails sur les valeurs des métriques représentées sur les figures précédentes voir respectivement les tableaux A3.6 et A3.12 en annexe.

Le modèle GMF affiche des résultats inférieurs à ceux des autres modèles (ce qui nous a même restreint à l'exclure du graphe afin de pouvoir mieux étudier les autres modèles) puisqu'il requiert les vecteurs latents des utilisateurs ainsi que ceux des items pour fonctionner, à l'inverse de MLP qui devrait apprendre à faire un filtrage collaboratif basé items.

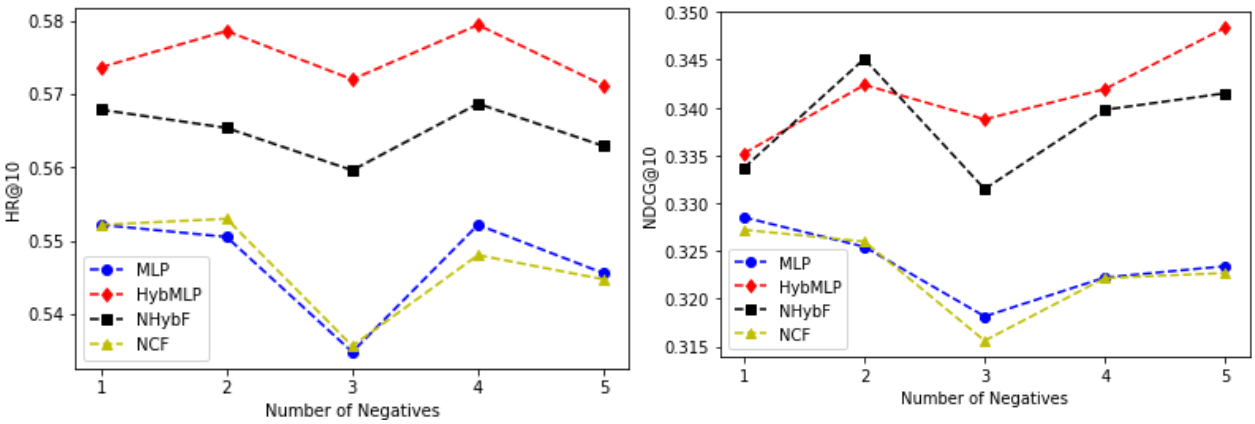


Figure 3.17 : Performance des modèles selon le nombre d'instances négatives par instance positive (sans GMF) - ML-1m.

Nous pouvons remarquer que HybMLP à lui seul a une meilleure performance que NHybF et ceci car GMF n'arrive pas à faire les prédictions comme expliqué précédemment (pour plus de détails sur les valeurs des métriques représentées sur les figures précédentes voir respectivement les tableaux A3.7 et A3.13 en annexe).

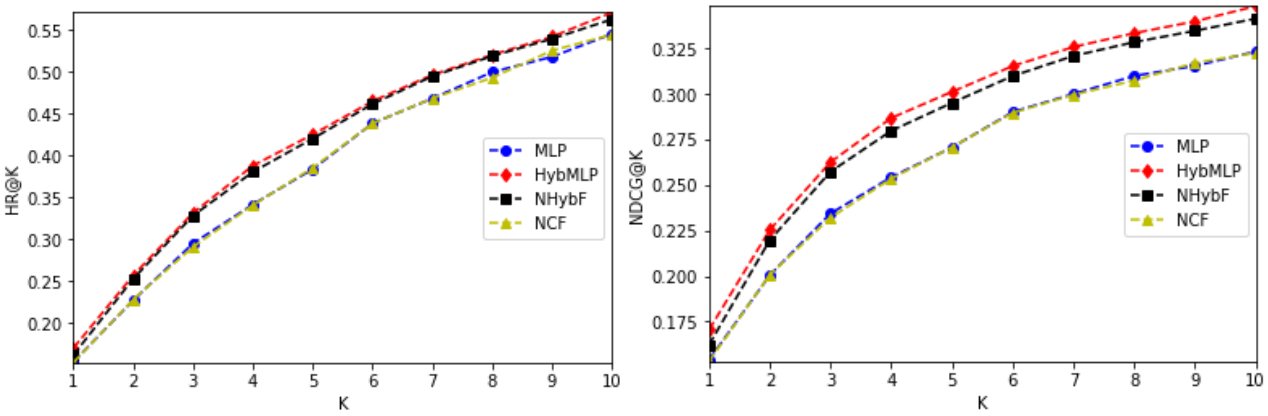


Figure 3.18 : Evaluation du Top-K de recommandation d'items avec K variant de 1 à 10 (sans GMF) – ML-1m.

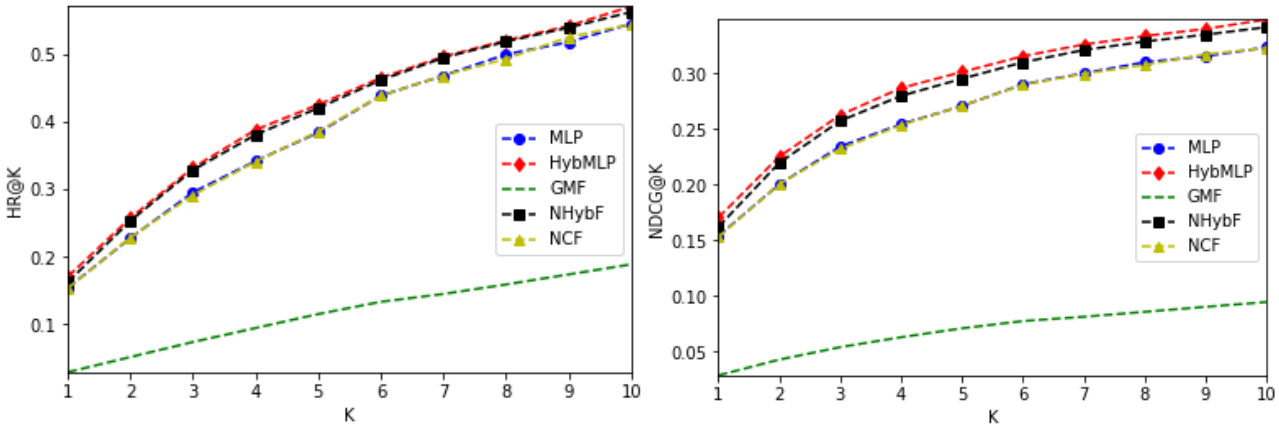


Figure 3.19 : Evaluation du Top-K de recommandation d'items avec K variant de 1 à 10 (avec GMF) - ML-1m.

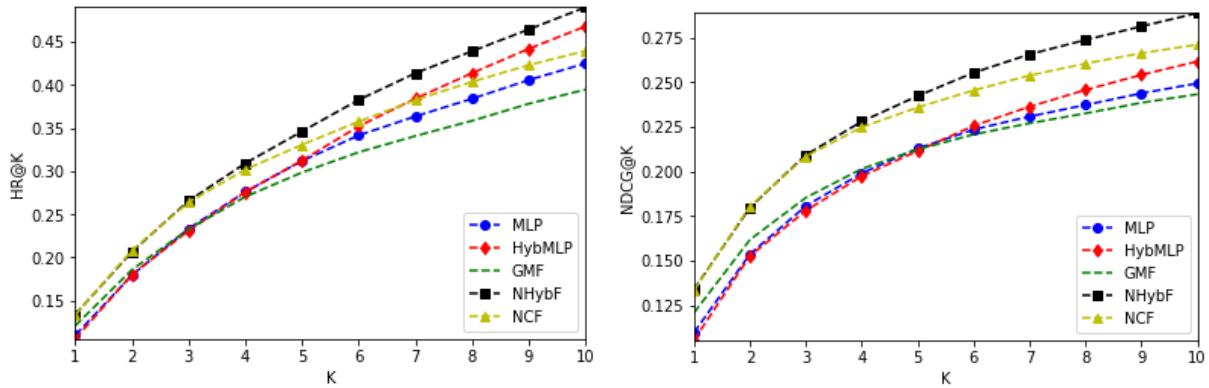


Figure 3.20 : Evaluation du Top-K de recommandation d'items avec K. variant de 1 à 10 (avec GMF) – Yelp.

4.3 – Discussion générale

L'objectif principal du modèle proposé étant de permettre au système de recommandation de rencontrer moins de difficultés face au démarrage à froid, il a été démontré que même sans ces difficultés le modèle surpasse les méthodes de l'état de l'art.

Le *deep learning* a prouvé son efficacité en donnant de meilleures prédictions et précisions. Plus le nombre de données à traiter augmente plus il est nécessaire d'avoir un réseau profond comme prouvé par les évaluations faites dans les figures 3.4, 3.5 et 3.8.

Le but était aussi de trouver un certain équilibre entre les instances négatives et positives de l'entraînement, car comme expliqué dans le chapitre 2 il s'agit d'un problème de classification, et ce type d'algorithmes est entraîné sur des jeux de données qui contiennent autant d'informations sur une classe qu'une autre. Dans notre cas, il a été démontré par expérimentation que pour le premier type d'évaluation le nombre d'instances négatives est égal à trois par instance positive pour ML-1m et de cinq pour Yelp.

5 – Conclusion

Dans ce chapitre, nous avons prouvé l'efficacité de notre modèle dans les deux cas d'évaluation.

Lors de la première approche d'évaluation, nos approches ont donné de meilleurs résultats que les approches étudiées pour ce qui est des tests effectués sur la base de films, Movielens. En revanche, ces approches étudiées (filtrage collaboratif) semblent afficher de meilleurs scores que nos approches (filtrage hybride) quand testées sur la base des restaurants, Yelp.

Nous pouvons ainsi en conclure que le filtrage collaboratif est plus intéressant pour la recommandation de restaurants que le filtrage hybride et l'inverse en ce qui concerne la recommandation de films. Ce qui semble logique vu que, lorsqu'on s'intéresse à un film, on doit cet intérêt en grande partie au contenu de ce dernier, alors que, si vous demandiez à une tierce personne ce qui l'a fait intéresser à un quelconque restaurant, elle dira probablement que c'était la recommandation d'une connaissance, ou alors ne donnera pas de réponse claire, « ça m'a juste intéressé » et c'est là où le filtrage collaboratif devient très intéressant, le model a donc appris de lui-même les points qu'il considérera lors de la prédiction de l'interaction. Le filtrage hybride permet cela également en raison de sa partie filtrage collaboratif, mais, comme vu avec la base Yelp, cela pourrait détériorer la précision si le problème est un problème de filtrage collaboratif. À noter que HybMLP a réussi à avoir des résultats presque aussi bons que ceux de MLP et ceci en ayant une taille de vecteur latents (e*mbedding size*) bien inférieure, 8 pour HybMLP contre 64 pour MLP, ce qui montre que le model a malgré tout tenu compte des données ajoutées qui ne lui ont pas été inutiles.

En ce qui concerne la deuxième évaluation où les modèles n'avaient pas connaissance des utilisateurs des données de test et donc où les risques du démarrage à froid étaient plus importants, nos approches ont affiché les meilleures résultats sur les tests effectués sur Yelp et l'écart qui sépare nos approches à celles existantes s'est même agrandi sur les tests effectués sur Movielens. Ce qui nous confirme le modèle a su tenir compte des données des utilisateurs et items que nous lui avons ajouté et donc a pu faire l'hybridation du FC et du FBC et a ainsi atténué les effets du problème de démarrage à froid bien connu des systèmes de recommandation basés sur le FC.

CONCLUSION

Nous nous sommes intéressés dans ce travail à l'application de l'apprentissage profond dans les systèmes de recommandation. Suite à l'étude de l'état de l'art, nous nous sommes inspirés particulièrement du travail d'**He et al [5]** qui a proposé l'approche de filtrage collaboratif (FC) basée sur une architecture avec les deux modèles: (1) la Factorisation Matricielle Généralisée, (*Generalized Matrix Factorization* - GMF) et le Perceptron Multicouches (*Multi Layer Perceptron* – MLP), nous avons proposé notre propre approche appelée NHybF (*Neural Hybrid Filtering*), qui combine le FC et le filtrage à base de contenu FBC selon une architecture basée sur GMF et HybMLP.

Afin d'évaluer notre approche, nous avons effectué plusieurs tests en utilisant les deux bases MovieLens 100k et MoviLens 1M, ainsi que la base Yelp. Les résultats ont montré une amélioration des performances de l'algorithme hybride NHybF comparé à l'algorithme de FC Neuronal sous la même architecture avec les métriques HR@K et NDCG@K. Dans notre évaluation nous avons tenu compte également des données implicites et des avantages et inconvénients de leur utilisation pour le développement des modèles.

Cependant, nous considérons que le modèle n'a pas appris de données assez complexes, comme les commentaires que pourrait laisser un utilisateur sur un item etc. Ce qui nous amène à chercher à exploiter d'autres types de données, pas forcément relatives à l'utilisateur ou l'item uniquement, mais plutôt relative à l'interaction elle-même. Par exemple, des textes contenant l'avis que l'utilisateur porte sur un item donné.

Comme perspectives futures à ce travail, nous envisageons tout d'abord d'évaluer notre approche en utilisant d'autres bases de données. Il serait aussi intéressant d'évaluer notre système dans une situation réelle pour récolter le feedback des utilisateurs sur les recommandations effectuées. Finalement, nous souhaitons utiliser d'autres architectures de *deep learning* comme les réseaux de neurones convolutifs (*Convolutional Neural Network* – CNN).

REFERENCES

[1] X. Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques (2009). Published by Hindawi Publishing Corporation, Advances in Artificial Intelligence, Volume 2009, Article ID 421425, 19 pages.

[2] D. Bokde, S. Girase, D. Mukhopadhyay - Department of Information Technology, Maharashtra Institute of Technology, Pune, india. Matrix factorization model in collaborative filtering algorithms (2015). 1877-0509 © 2015 The Authors. Published by Elsevier B.V.

[3] G. Kembellec, G. Chartron, I. Saleh. Les moteurs et systèmes de recommandation. Collection systèmes d'information, web et informatique ubiquitaire. London, ISTE éditions, cop. 2014. Pages 3 4.

[4]https://towardsdatascience.com/mnist-vs-mnist-how-i-was-able-to-speed-up-my-deep-learning-11c0787e6935, site consulté en Février 2019.

[5] X. He and al. Neural Collaborative Filtering. International World Wide Web Conference Committee (IW3C2), published under Creative Commons CC BY 4.0 License. WWW 2017, April 3–7, 2017, Perth, Australia. ACM 978-1-4503-4913-0/17/04. (2017).

[6] S. Zhang, L. Yao, A. Sun, Nanyang, Yi Tay. Deep Learning based Recommender System: A Survey and New Perspectives. ACM Computing Surveys, Vol.1, No.1, Article 1, July 2018.

[7] Machine Learning by Tom M. Mitchell. Published October 1st 1997 by McGraw-Hill. ISBN 0071154671 (ISBN13: 9780071154673).

[8] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In SIGIR, pages 549–558, 2016.

[9] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In WWW, pages 285–295, 2001.

[10] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In UAI, pages 452–461, 2009.

[11] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In Advances in Neural Information Processing Systems, volume 20, 2008.

[12] X. He, X. Du, X. Wang, F. Tian, J. Tang, and T. Chua. 2018. Outer Product-based Neural Collaborative Filtering. (2018).

[13] A. Ng, Machine Learning Course from Stanford University (Coursera).

[14] S. Belkacem et M. Ouafi, Recommandation de membres dans les réseaux sociaux – Cas du réseau social Yelp. Mémoire de fin d'études de master, Option Systèmes Informatiques Intelligents (SII), Département Informatique, USTHB, 2015.

[15]https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels#/media/Fichier:ArtificialNeuronModel_francais.png, site consulté en Mars 2019.

ANNEXE

A.1 – Etapes du filtrage collaboratif basé mémoire

Le filtrage collaboratif basé mémoire passe par deux étapes : le calcul de similarité et le calcul de prédiction.

1) Calcul de similarité

- **Similarité vectorielle :**

$$sim(U_1, U_2) = \cos(r_{U_1}, r_{U_2}) = \frac{\sum_{i=1}^{n}(r_{U_1 i} \times r_{U_2 i})}{\sqrt{\sum_{i=1}^{n} r_{U_1 i^2}} \times \sqrt{\sum_{i=1}^{n} r_{U_2 i^2}}}$$

Où :

U_j : Utilisateur j.

r_{U_j} : Vecteur d'évaluation de l'utilisateur j.

$r_{U_j i}$: L'évaluation de l'item i par l'utilisateur j.

Exemple :
Nous avons le tableau suivant :

User\Item	I1	I2	I3	I4	I5	I6	I7
U1	4	0	0	5	1	0	0
U2	5	5	4	0	0	0	0
U3	0	0	0	2	4	5	0
U4	0	3	0	0	0	0	3

On obtient approximativement : $sim(U_1, U_2) = 0.38$ et $(U_1, U_3) = 0.31$. La similarité est proche alors que les utilisateurs U_1 et U_3 ont des goûts très différents, l'un déteste les films que l'autre a apprécié contrairement à U_1 et U_2 qui ont l'air d'apprécier le même film.

L'inconvénient de cette méthode est qu'elle considère les items non évalués comme étant des évaluations négatives.

- **Corrélation de Pearson :**

La corrélation de Pearson a été mise en place pour remédier au problème qu'exposait la similarité vectorielle :

$$Pearson(U_1, U_2) = \frac{\sum_i (r_{U_1 i} - \overline{r_{U_1}}) \times (r_{U_2 i} - \overline{r_{U_2}})}{\sqrt{\sum_i (r_{U_1 i} - \overline{r_{U_1}})^2 \times (r_{U_2 i} - \overline{r_{U_2}})^2}}$$

Où:

U_j : Utilisateur j.

$r_{U_j i}$: L'évaluation de l'item i par l'utilisateur j.

$\overline{r_{U_j}}$: La moyenne des évaluations de l'utilisateur j.

Exemple :

Prenons la matrice d'évaluations suivante :

User\Item	I1	I2	I3	I4	I5	I6	I7
U1	4			5	1		
U2	5	5	4				
U3				2	4	5	
U4		3					3

On calcule la moyenne des évaluations pour chaque utilisateur et on la soustrait de chaque évaluation de l'utilisateur, ce qui donne :

User\Item	I1	I2	I3	I4	I5	I6	I7
U1	2/3			5/3	-7/3		
U2	1/3	1/3	-2/3				
U3				-5/3	1/3	4/3	
U4		0					0

Après le calcul de la similarité on obtient : $sim(U_1, U_2) = 0.09$ et $sim(U_1, U_3) = -0.56$, on en conclut que les utilisateurs U_1 et U_2 sont plus susceptibles d'apprécier les mêmes items.

2) Calcul de prédictions

- **<u>Somme pondérée :</u>**

$$P(u_i, i_k) = \overline{r_{u_i}} + \frac{\sum_{u_j \in U_i} sim(u_i, u_j) \times (r_{u_j, i_k} - \overline{r_{u_j}})}{\sum_{u_j \in U_i} |sim(u_i, u_j)|}$$

Où:

u_i: Utilisateur i.

i_k: Item k.

$sim(u_i, u_j)$: La similarité entre les utilisateurs i et j.

r_{u_j, i_k}: Evaluation de l'item k par l'utilisateur j.

$\overline{r_{u_i}}$: Moyenne des évaluations faites par l'utilisateur i.

A.2 – Développement du système de recommandation avec Python

Dans cette section nous expliquons quelques détails sur le développement du système de recommandation sous Python.

A2.1 – Prétraitement des données

1- ***Lecture des fichiers :*** les fichiers doivent être en format csv avec un séparateur entre les colonnes. La lecture se fait avec la fonction read_csv() de Pandas.
2- ***Transformation des données :*** le tableau suivant (table A2.1) décrit les fonctions utilisées.
3- ***Sauvegarde des fichiers d'entraînement et de test.***

Tableau A2.1 : Fonctions de transformation des données.

Dataset	***Biblioth-èques utilisées***	***Fonctions***	***Description***	***Complexité***	
MovieLens	Numpy, Pandas	*genre_items()*	Transforme les genres des items (films) donnés (action, comedy…) en vecteurs booléens.	$O(y * k * n)$	***n :*** nombre d'items. ***k :*** taille des données relatives à chaque item (k=18 pour ce *dataset*). ***y :*** taille de la chaine des genres d'items.
Yelp	Numpy, Pandas, codecs	*transform_users()*	Transforme les données relatives aux utilisateurs lues à partir d'un fichier json (objet) en vecteurs numériques.	$O(k * n)$	***n :*** nombre d'utilisateurs. ***k :*** taille des données relatives à chaque item (k=17 pour ce *dataset*).
		transform_items()	Transforme les données relatives aux items lues à partir d'un fichier json (objet) en vecteurs numériques.	$O(k * n)$	***n :*** nombre d'items. ***k :*** taille des données relatives à chaque item (k=50 pour ce *dataset*).

A2.2 – Génération du jeu d'entraînement et de test

1- ***Lecture des données :*** lecture des fichiers des jeux d'entraînement et de test.
2- ***Génération des fichiers :*** génère les fichiers d'entraînement et de test avec les fonctions décrites dans le tableau suivant :

Tableau A2.2 : Fonctions de génération des jeux d'entraînement et jeux de données.

Biblioth -èques utilisées	***Fonctions***	***Description***	***Complexité***	
Numpy, Pandas, sklearn	*dataset_split_opt1()*	Partage le *dataset* en *trainset* et *testset* tel que le *trainset* contient 80% du nombre d'utilisateurs. Prend une instance positive pour chaque utilisateur pour le *testset*. Supprime les interactions des utilisateurs choisis pour le test du *trainset*. Ajoute les instances négatives au *testset*. Ajoute aussi des instances négatives au *trainset*.	$O(u * (n + y) + x * (n - u))$	***u :*** nombre d'utilisateurs du *testset*. **n :** nombre d'interactions du *dataset*. ***x :*** nombre de tentatives afin de trouver le nombre d'instances négatives par instance positive souhaité dans le *trainset* (number of negatives). ***y :*** nombre de tentatives pour trouver une instance positive et 99 autres négatives pour le *testset*.
Numpy, Pandas	*dataset_split_opt2()*	Partage le *dataset* en *trainset* et *testset* tel que le *testset* contient une interaction (instance positive) par utilisateur. Supprime les interactions choisies pour le test du *trainset*. Ajoute les instances négatives au *testset*. Ajoute aussi des instances négatives au *trainset*.	$O(u * (n + y) + x * (n - u))$	***u :*** nombre d'utilisateurs du *dataset*. ***n :*** nombre d'interactions du *dataset*. ***x :*** nombre de tentatives afin de trouver le nombre d'instances négatives par instance positive souhaité dans le *trainset* (number of negatives). ***y :*** nombre de tentatives pour trouver une instance positive et 99 autres négatives pour le *testset*.

A2.3 – Construction, entraînement des modèles et évaluation

1- ***Construction des modèles :*** les modèles NHybF, HybMLP et GMF sont construits à partir de l'API Model[26] de *Keras.*
2- ***Entraînement :*** entraînement d'un modèle donné : utilise la fonction *fit()* de *Keras* afin d'effectuer une itération sur tout le jeu de données.
3- ***Evaluation :*** évalue le modèle après chaque itération et sauvegarde le modèle de la meilleure itération ainsi que les scores des métriques de toutes les itérations. Les fonctions d'évaluation sont décrites dans le tableau suivant :

Tableau A2.3 : Fonctions d'évaluation.

Biblioth -èques utilisées	***Fonctions***	***Description***	***Complexité***	
Numpy, math	*evaluate_by_user()*	Recherche l'item pertinent dans la liste des prédictions.	$O(k)$	***k :*** taille de la lise des items recommandés.
Numpy, Pandas	*evaluate_model()*	Prédit l'interaction des couples utilisateur-item présents dans le *testset* Trie le *testset* selon la prédiction du modèle ($\in [0,1]$) dans l'ordre décroissant. Sélectionne pour chaque utilisateur les k items les plus recommandés. Recherche l'item pertinent dans la liste des prédictions (appelle evaluate_by_users()).	$O(n * \log(n) + u * (n + k) + X)$	***n :*** taille des données du test. ***u :*** nombre d'utilisateurs du *testset.* ***k :*** taille de la liste d'items recommandés. ***X :*** complexité de la fonction de prédiction, dépend du modèle choisi ainsi que de sa structure.

[26] **Model (functional API) :** https://keras.io/models/model/

A.3 – Evaluations et tests effectués

1) MovieLens 1m :

- *Evaluation avec des utilisateurs et items connus :*

Tableau A3.1 : Evaluation de HybMLP selon différents nombres de couches.

Nombre de couches		**0**	**1**	**2**	**3**	**4**	**5**
Modèle	***Métrique @10***						
MLP	**HR**	0.568	0.743	0.749	0.766	0.767	**0.777**
	NDCG	0.334	0.473	0.487	0.499	0.502	**0.511**
HybMLP	**HR**	0.567	0.751	0.762	0.775	0.781	**0.788**
	NDCG	0.334	0.481	0.492	0.506	0.514	**0.521**

Tableau A3.2 : Evaluation des modèles selon différentes tailles d'*embedding*.

T. E.	**8**		**16**		**32**		**64**		**128**	
Métrique @10	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**
MLP	0.752	0.483	0.768	0.499	0.776	0.509	0.777	0.51	0.778	0.517
HybMLP	**0.758**	**0.489**	**0.773**	**0.509**	**0.789**	**0.523**	**0.788**	**0.521**	**0.786**	**0.519**
GMF	0.746	0.478	0.771	0.507	0.773	0.509	0.774	0.51	0.771	0.502

Tableau A3.3 : Evaluation selon le nombre de nœuds de la dernière couche (*predictive factors*).

Predictive factors	**HybMLP**		**NHybF**	
Métrique @10	**HR**	**NDCG**	**HR**	**NDCG**
8	0.778	0.515	0.79	0.53
16	0788	0.521	0.8	0.536
32	0.789	0.523	0.8	0.536
64	0.794	0.528	0.802	0.539
128	**0.797**	**0.531**	**0.805**	**0.541**

Tableau A3.4 : Evaluation des différents modèles selon le nombre d'instances négatives par instance positive.

Nombre d'instances négatives	**1**		**2**		**3**		**4**		**5**	
Métrique @10	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**
MLP	0.785	0.525	0.8	0.542	0.812	0.561	0.802	0.552	0.81	0.558
HybMLP	**0.797**	**0.531**	**0.808**	**0.55**	**0.818**	**0.569**	**0.817**	**0.564**	**0.819**	**0.57**
GMF	0.774	0.51	0.794	0.534	0.808	0.556	0.797	0.546	0.803	0.552
NHybF	**0.8**	**0.541**	**0.816**	**0.559**	**0.827**	**0.583**	**0.822**	**0.573**	**0.822**	**0.581**
NCF	0.794	0.535	0.811	0.554	0.823	0.578	0.813	0.567	0.82	0.574

Tableau A3.5 : Evaluation du Top K de recommandation d'items.

K		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Modèle	***Métrique @10***										
MLP	**HR**	0.336	0.483	0.571	0.64	0.687	0.724	0.746	0.77	0.792	0.812
	NDCG	0.336	0.429	0.473	0.503	0.521	0.534	0.541	0.549	0.555	0.561
HybMLP	**HR**	**0.344**	**0.494**	**0.581**	**0.648**	**0.695**	**0.729**	**0.757**	**0.78**	**0.8**	**0.818**
	NDCG	**0.344**	**0.439**	**0.482**	**0.511**	**0.529**	**0.541**	**0.551**	**0.558**	**0.564**	**0.569**
GMF	**HR**	0.329	0.479	0.572	0.636	0.685	0.72	0.747	0.768	0.788	0.809
	NDCG	0.329	0.424	0.47	0.498	0.517	0.529	0.538	0.545	0.551	0.557
NHybF	**HR**	**0.362**	**0.509**	**0.597**	**0.658**	**0.712**	**0.748**	**0.774**	**0.795**	**0.813**	**0.828**
	NDCG	**0.362**	**0.455**	**0.499**	**0.525**	**0.546**	**0.559**	**0.568**	**0.574**	**0.580**	**0.584**
NCF	**HR**	0.355	0.504	0.59	0.653	0.7	0.737	0.768	0.79	0.806	0.824
	NDCG	0.355	0.449	0.492	0.52	0.538	0.551	0.561	0.568	0.573	0.578

- *Evaluation de nouveaux utilisateurs :*

Tableau A3.6 : Evaluation des différents modèles selon le nombre d'instances négatives par instance positive.

Nombre d'instnaces négatives	**1**		**2**		**3**		**4**		**5**	
Métrique @10	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**
MLP	0.552	0.329	0.55	0.325	0.534	0.318	0.552	0.322	0.545	0.323
HybMLP	**0.574**	**0.335**	**0.579**	**0.342**	**0.572**	**0.339**	**0.579**	**0.342**	**0.571**	**0.348**
GMF	0.166	0.083	0.154	0.081	0.171	0.084	0.16	0.08	0.189	0.095
NHybF	**0.569**	**0.334**	**0.565**	**0.345**	**0.56**	**0.331**	**0.569**	**0.34**	**0.563**	**0.341**
NCF	0.552	0.327	0.553	0.326	0.536	0.316	0.548	0.322	0.545	0.323

Tableau A3.7 : Evaluation du Top K de recommandation d'items.

K		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Model	***Metric @10***										
MLP	**HR**	0.154	0.228	0.296	0.342	0.384	0.439	0.469	0.5	0.518	0.546
	NDCG	0.154	0.2	0.234	0.254	0.271	0.29	0.3	0.31	0.315	0.323
HybMLP	**HR**	**0.17**	**0.257**	**0.332**	**0.388**	**0.425**	**0.465**	**0.497**	**0.52**	**0.542**	**0.571**
	NDCG	**0.171**	**0.225**	**0.262**	**0.287**	**0.301**	**0.315**	**0.326**	**0.333**	**0.34**	**0.348**
GMF	**HR**	0.028	0.051	0.074	0.094	0.115	0.133	0.145	0.159	0.174	0.189
	NDCG	0.02	0.04	0.05	0.063	0.071	0.078	0.082	0.086	0.09	0.095
NHybF	**HR**	**0.162**	**0.253**	**0.328**	**0.381**	**0.421**	**0.462**	**0.495**	**0.519**	**0.54**	**0.562**
	NDCG	**0.162**	**0.22**	**0.257**	**0.28**	**0.295**	**0.309**	**0.321**	**0.329**	**0.335**	**0.341**
NCF	**HR**	0.153	0.228	0.291	0.341	0.386	0.439	0.469	0.493	0.526	0.545
	NDCG	0.153	0.2	0.232	0.254	0.271	0.29	0.299	0.307	0.317	0.323

2) Yelp :

- *Evaluation avec utilisateurs et items connus :*

Tableau A3.8 : Evaluation de HybMLP selon différents nombres de couches.

Nombre de couches		**0**	**1**	**2**	**3**	**4**	**5**
Modèle	***Métrique @10***						
MLP	**HR**	0.411	0.653	0.67	**0.666**	0.663	0.659
	NDCG	0.229	0.38	0.388	**0.383**	0.385	0.383
HybMLP	**HR**	0.411	0.552	**0.685**	0.653	0.656	0.65
	NDCG	0.228	0.311	**0.4**	0.381	0.381	0.366

Tableau A3.9 : Evaluation des modèles selon différentes tailles d'*embedding*.

T. E.	**8**		**16**		**32**		**64**		**128**	
Métrique @10	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**
MLP	0.667	0.383	0.665	0.384	0.666	0.386	0.667	0.388	**0.677**	**0.389**
HybMLP	**0.685**	**0.4**	0.576	0.321	0.634	0.358	0.646	0.37	0.622	0.351
GMF	0.703	0.428	0.703	0.428	0.718	0.434	0.716	0.437	**0.726**	**0.449**

Tableau A3.10 : Evaluation des différents modèles selon le nombre d'instances négatives par instance positive.

Nombre d'instances négatives	**1**		**2**		**3**		**4**		**5**	
Métrique @10	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**
MLP	0.677	0.389	0.712	0.425	0.746	0.447	0.757	0.458	**0.764**	**0.469**
HybMLP	0.651	0.366	0.691	0.407	0.717	0.427	0.727	0.437	**0.725**	**0.439**
GMF	0.726	0.449	0.755	0.468	0.746	0.468	0.754	0.468	**0.756**	**0.474**
NHybF	0.721	0.44	0.751	0.461	0.759	0.475	0.766	0.479	**0.772**	**0.484**
NCF	0.723	0.442	0.764	0.47	0.776	0.484	0.782	0.491	**0.788**	**0.5**

Tableau A3.11 : Evaluation du Top K de recommandation d'items.

K		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Modèles	***Métrique @10***										
MLP	**HR**	0.221	0.36	0.458	0.534	0.589	0.637	0.676	0.71	0.739	0.764
	NDCG	0.221	0.309	0.358	0.39	0.411	0.429	0.442	0.452	0.461	0.468
HybMLP	**HR**	0.206	0.327	0.423	0.489	0.545	0.591	0.634	0.669	0.697	0.723
	NDCG	0.206	0.282	0.331	0.359	0.381	0.397	0.411	0.422	0.431	0.438
GMF	**HR**	0.237	0.372	0.463	0.536	0.587	0.63	0.667	0.702	0.734	0.756
	NDCG	0.237	0.323	0.368	0.399	0.419	0.435	0.447	0.458	0.467	0.474
NHybF	**HR**	**0.242**	**0.379**	**0.479**	**0.545**	**0.603**	**0.648**	**0.686**	**0.719**	**0.747**	**0.769**
	NDCG	**0.242**	**0.3285**	**0.376**	**0.406**	**0.429**	**0.445**	**0.458**	**0.468**	**0.476**	**0.483**
NCF	**HR**	**0.254**	**0.398**	**0.503**	**0.569**	**0.624**	**0.67**	**0.706**	**0.737**	**0.762**	**0.787**
	NDCG	**0.254**	**0.345**	**0.397**	**0.426**	**0.447**	**0.463**	**0.475**	**0.485**	**0.493**	**0.5**

- *Evaluation avec de nouveaux utilisateurs :*

Tableau A3.12 : Evaluation des différents modèles selon le nombre d'instances négatives par instance positive.

Nombre d'instances négatives	**1**		**2**		**3**		**4**		**5**	
Métrique @10	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**	**HR**	**NDCG**
MLP	0.116	0.053	0.16	0.068	0.134	0.065	0.122	0.06	0.132	0.072
HybMLP	0.117	0.055	0.186	0.089	0.173	0.077	0.213	0.11	0.12	0.059
GMF	0.112	0.05	0.101	0.045	0.091	0.041	0.099	0.045	0.098	0.043
NHybF	**0.122**	**0.056**	**0.187**	**0.09**	**0.168**	**0.079**	**0.208**	**0.109**	0.094	0.046
NCF	0.093	0.053	0.157	0.067	0.132	0.062	0.115	0.057	**0.139**	**0.077**

Tableau A3.13 : Evaluation du Top K de recommandation d'items.

K		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Modèle	***Métrique @10***										
MLP	**HR**	0.109	0.179	0.233	0.276	0.312	0.342	0.364	0.384	0.405	0.425
	NDCG	0.109	0.154	0.18	0.199	0.213	0.224	0.231	0.237	0.244	0.249
HybMLP	**HR**	**0.105**	**0.18**	**0.231**	**0.276**	**0.313**	**0.352**	**0.384**	**0.413**	**0.442**	**0.468**
	NDCG	**0.106**	**0.153**	**0.178**	**0.197**	**0.212**	**0.226**	**0.236**	**0.246**	**0.254**	**0.262**
GMF	**HR**	0.12	0.185	0.233	0.27	0.298	0.322	0.341	0.349	0.378	0.394
	NDCG	0.121	0.162	0.185	0.202	0.212	0.221	0.227	0.233	0.239	0.243
NHybF	**HR**	**0.134**	**0.207**	**0.265**	**0.309**	**0.346**	**0.383**	**0.414**	**0.434**	**0.464**	**0.49**
	NDCG	**0.134**	**0.18**	**0.209**	**0.228**	**0.242**	**0.255**	**0.266**	**0.274**	**0.281**	**0.289**
NCF	**HR**	0.133	0.208	0.264	0.302	0.33	0.357	0.382	0.383	0.423	0.44
	NDCG	0.133	0.18	0.208	0.225	0.236	0.246	0.254	0.26	0.266	0.271