Tutoriel systèmes de recommandation

Francois Doan-Pope & Roukayatou Ousmane

1 Avril 2020

Table des matières

1 Présentation du sujet

1.1 Introduction aux systèmes de recommandation

Les systèmes de recommandations sont définis comme étant des systèmes qui aident un usager à trouver un ou des items d'intérêt lorsque l'information nécessaire afin de faire un choix est difficilement accessible (Michael Ekstrand et Joe Konstan, s. d.). Un exemple moderne de ce genre de systèmes sont les suggestions de Netflix, qui offrent des recommandations personnalisées à l'utilisateur. Il peut être excessivement difficile de trouver un film qui nous intéresse dû à la grosseur du catalogue. Ainsi, Netflix offre des suggestions afin de faciliter notre choix. Il serait faux de croire que les systèmes de recommandations sont une nouvelle invention, car historiquement, ils pouvaient se manifester sous de critiques de pièces de théatre, par exemple, qui nous suggèrent ce qui mérite d'être vu ou non. De ce fait, c'est plutôt certaine approches qui sont modernes. Les objectifs de ces systèmes peuvent être de vendre un produit (dans le cas d'Amazon, par exemple), de bâtir une communauté (Reddit), ou tout simplement mieux répondre aux besoins des usagers (comme dans le cas de journaux).

1.2 Concepts de base

L'objectif de cette section est de brièvement décrire la taxonomie, les entrées des systèmes (*inputs*), les sorties (*outputs*) et quelques considérations pratiques. De manière générale, les entrées d'un système de recommandation sont des données nous permettant de quantifier les préférences d'un utilisateur. Celles-ci peuvent être des variables démographiques, mais nous allons considérer de plus près les évaluations implicites et explicites.

Les retours d'informations (*feedback*) **explicites** sont disponibles lorsque nous demandons directement à l'usager leurs préférences, sous forme de note sur cinq, par exemple. Malgré que ce type données semblent très pertinentes, il faut exercer une certaine caution. En effet, celles-ci peuvent comporter des biais, notamment dû aux changements de préférences dans le temps. Il n'est généralement pas pertinent d'offrir des recommandations basées sur les avis d'un usager il y a trois ans, par exemple. Les systèmes basés sur les échelles (notes étoiles, notamment) ont aussi leurs lots de problèmes. Pour le lecteur curieux, nous suggérons un article décrivant les raisons qui ont poussé Netflix a remplacer le système de notes par un pouce vers le haut ou le bas, intitulé *The exec who replaced Netflix's 5-star rating system with 'thumbs up, thumbs down' explains why* du *Business Insider* (McAlone 2017). Une des raisons soulevée est le phénomène de "syndrome de critique de film", une situation dans laquelle les usagers donnait une mauvaise note à un film, malgré que celui-ci consomme ce genre de film. Ceci ne permettrait donc pas de recommander les "guilty pleasures" aux usagers.

Les retours d'informations *implicites* sont des interactions des usagers avec le système qui nous permettent de mieux comprendre les préférences de ceux-ci. Des exemples d'interactions sont des clics, si un usager a changé de chanson avant la fin, des achats, etc. Il est évident que les entreprises ont maintenant accès à une quantité importante de ce genre de données, rendant l'analyse intéressante. Certaines caractéristiques des retours d'informations implicites sont à souligner. Premièrement, basé sur des interactions, il est généralement plus facile de déterminer ce que l'usager aime, versus ce qu'il n'aime pas. Cette asymétrie n'est pas présente dans le contexte de retours d'informations explicites. Dans le contexte de "*feedback*" implicite, si l'on ignore les données manquantes, nous prenons le risque de se baser uniquement sur les retours d'informations positifs; incomplètes pour dresser un portrait adéquat. La seconde caractéristique est la nature excessivement bruitée des données implicites. La troisième est que la valeur numérique dans une matrice de retours d'informations explicites représente la préférence, tandis que dans le cas implicite, cela représente la confiance (le temps durant lequel un usager a visionné un film, le nombre de clics,...). Ces informations n'indiquent pas directement les préférences des utilisateurs (Hu, Koren, et Volinsky 2008).

Une note importante doit être fait au niveau de la normalisation des données dans le cas de notes explicites des utilisateurs. Il est important de prendre en compte le biais de chaque utilisateur afin que les évaluations attribuées soient représentatives, car certains utilisateurs peuvent être particulièrement critiques (ou inversement, trop généreux). Il est donc préférable de procéder à une standardisation lorsque nous avons des évaluations basées sur une échelle.

Au niveau des sorties (*outputs*) des systèmes de recommandation, celles-ci peuvent être sous forme de prédictions ou de recommandations. Une prédiction quantifie la probabilité qu'un usager aime un article, ou directement une estimation de la note. La seconde sortie sont les recommandations sous forme de liste.

1.3 Les types de systèmes de recommandation

1.3.1 L'approche non personnalisée

Cette approche est généralement basée sur la popularité d'un item, ou l'avis d'un expert. On peut donc penser à des critiques de films, le guide Michelin, les listes de meilleurs vendeurs, etc. Malgré que ces techniques sont plutôt rudimentaires, il ne faut pas pour autant les délaissés, car ceux-ci peuvent être assez efficaces. Une situation dans laquelle cette technique excellence est dans le cas de nouveaux usagers, terme qu'on appelle "cold-start". Comme nous n'avons presque aucune information à propos de ces utilisateurs, l'approche non personnalisé est une manière d'offrir des recommandations. L'un des désavantages évident de cette approche est

qu'on ne prend pas en compte les préférences individuelles de l'usager.

1.3.2 Le filtrage par contenu (Content-based filtering)

Comme le nom l'indique, le "content-based filtering" utilise les attributs du contenu. Ces attributs peuvent être la description du produit, la maison de production d'un film, les étiquettes (*tags*) associés par les utilisateurs, etc. Pour utiliser cette méthode, nous avons besoin de données hautement structurés. L'avantage est que nous n'avons pas besoin de beaucoup d'usagers dans le système. Par ailleurs, il est simple d'expliquer pourquoi un item en particulier est recommandé. Un des plus importants désavantage est que l'utilisateur ne sera probablement jamais présenté avec du contenu avec lequel il n'est pas familier. Ainsi, le système de recommandation n'offrira pas de suggestions "découverte" (Michael Ekstrand et Joe Konstan, s. d.).

1.3.3 Le filtrage collaboratif (collaborative filtering)

Cette méthode est basé sur l'idée que nous pouvons baser nos recommandations sur des individus ayant des préférences similaires, sans forcément avoir de connaissances sur les items. On peut penser aux recommandations d'*Amazon*, lorsqu'on nous présente "les usagers ayant acheté ce produit ont également acheté cet autre produit". Cette approche est basée sur les utilisateurs, mais nous pourrions également procéder du point de vue item.

L'approche par filtrage collaboratif est divisé en deux catégories: l'approche par modèle et l'approche mémoire. Dans l'approche mémoire, on tente de mesurer la similitude entre items (*item-based*) ou utilisateurs (*user-based*), et on utilise les voisins pour la recommandation. L'approche modèle tente de paramétriser la matrice d'usagers et d'items afin de produire des prédictions (Michael Ekstrand et Joe Konstan, s. d.).

L'avantage principal de la méthode par filtrage collaboratif est que nous sommes en mesure de recommander des items sans avoir une compréhension directe des facteurs qui influencent les préférences des usagers. On peut ainsi utiliser plusieurs entrées, dont les évaluations explicites et implicites mentionnées plus tôt. Considérant que la méthode est basée sur les préférences d'usagers, un problème important est la situation de "*cold-start*". Il est impossible de recommander à un usager sans avoir ses préférences, ou de recommander un nouvel item pour lequel nous n'avons pas de données d'interactions.

1.3.4 Les systèmes de recommandations hybrides

Les systèmes de recommandations modernes sont généralement des systèmes hybrides, c'est-à-dire une combinaison de plusieurs algorithmes. Le concept derrière cette approche est que la combinaison de plusieurs modèles permetent de capitaliser sur les forces de chacun et de pallier leurs faiblesses (pensez méthodes d'ensemble). Le gagnant de la fameuse compétition de Netflix a utilisé une combinaison de plus de 100 approches. Il est important de noter que les algorithmes utilisés dans les systèmes de recommandations hybrides n'ont pas à être différent afin d'être plus performant qu'une approche dite "pure" (Michael Ekstrand et Joe Konstan, s. d.).

2 Revue de littérature

Les systèmes de recommandation traitent le problème de surcharge d'informations que les utilisateurs rencontrent normalement en leur fournissant des recommandations de contenu et de service personnalisées et exclusives (Isinkaye, Folajimi, et Ojokoh 2015). Depuis leurs l'avènement plusieurs classifications ont été proposées dans la littérature. Ils sont généralement classifiées en 3 catégories : l'approche basée sur le contenu, le filtrage collaboratif et l'approche hybride qui est une combinaison des deux précédentes (Adomavicius et Tuzhilin 2005).

Les systèmes de filtrage basés sur le contenu recommandent à l'utilisateur en fonction de ses préférences passées. En revanche, les systèmes de filtrage collaboratif basent leur recommandation pour un utilisateur donné, sur les préférences passées des autres utilisateurs qui partagent ses goûts (Bendakir et Aïmeur, s. d.).

Selon Adomavicius et Alexander Tuzhilin, les systèmes de filtrage collaboratif peuvent être classifiés en deux groupes: les approches basées sur la mémoire et celles basées sur les modèles (Adomavicius et Tuzhilin 2005).

Les tâches de filtrage collaboratif présentent de nombreux défis: Les algorithmes doivent avoir la capacité de traiter des données très clairsemées, d'évoluer avec le nombre croissant d'utilisateurs et d'articles, de faire des recommandations satisfaisantes dans un court laps de temps (Su et Khoshgoftaar 2009).

L'article "A review on Matrix Factorization Techniques in Recommender Systems" par Mehta et Rana présente la factorisation de matrices comme étant une solution très utilisée pour les systèmes de recommandations (Mehta et Rana 2017). Notamment, elle permet de contourner le fait que les systèmes de recommandations ne sont généralement pas extensibles (scalable) et ne fonctionnent pas de manière optimale avec des données à haute dimensions. Cependant,

la factorisation matricielle, décrits dans la prochaine section, est une exception, car elle vise à réduire la dimensionnalité. L'une des premières techniques de factorisation matricielle appliquée aux systèmes de recommandation fût le SVD (singular value decomposition) en 2000. Cependant, une imputation doit être faite avant d'utiliser le SVD, ce qui réduit la précision du modèle, peut mener à un sur-apprentissage et est plutôt dispendieux au niveau computationnel (Mehta et Rana 2017). Dans le cadre de la compétition Netflix, Simon Funk propose une alternative : utiliser la descente du gradient afin de minimiser l'erreur basée sur les valeurs connues.

3 Description des méthodes

3.1 Le filtrage collaboratif basé utilisateur

Le "*user-based collaborative filtering*" est une approche mémoire. L'intuition est que les utilisateurs avec des préférences similaires vont accorder des notes similaires. Ainsi, nous pouvons prédire les notes manquantes (ou les items à recommander) en tentant de trouver les usagers similaires et en agréger leurs notes. Deux méthodes populaires permettant de calculer la similitude entre usager sont la similarité cosinus et le coefficient de corrélation Pearson (Hahsler 2015). Arpès le calcul de similitude, on peut utiliser l'approche des k plus proches voisins, par exemple. L'inconvénient de cette méthode est que nous avons besoin de l'entièreté de la base de données en mémoire et que le calcul des distances peut être très coûteux.

3.2 Le filtrage collaboratif basé items

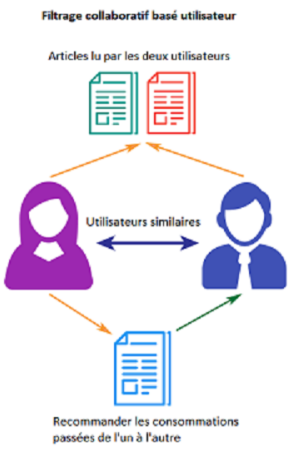


FIGURE 1 – UBCF

Le "*item-based collaborative filtering*" est basé sur l'idée que les utilisateurs vont préférer des articles similaires à ceux qu'ils aiment. Cette approche basée mémoire tente d'identifier les relations entre items basé sur la matrice de notes. L'idée est similaire au filtrage collaboratif basé utilisateurs; on peut mesurer la similitude avec la similarité cosine pour ensuite trouver les k items les plus proches (Hahsler 2015). Il est important de noter que nous ne conservons que les items similaires pour chaque article. Ainsi, ce modèle est plus efficace car la matrice de similitude est nettement plus petite, permettant ainsi d'être déployé à grande échelle. Cependant, les résultats sont généralement légèrement moins intéressants que pour la méthode user-based CF.

3.3 Factorisation matricielle

Nous avons brièvement introduit la factorisation matricielle dans la revue de littérature. Compte tenu de la grande quantité de variantes disponibles, l'explication de la méthode sera d'un point de vue global. Pour mieux comprendre les subtilités des différentes méthodes, je vous suggère l'article de Mehta et Rana mentionné à la section 2.

3.3.1 Singular Value Decomposition (SVD)

Si l'on considère qu'une matrice d'évaluations item-usager est une représentation des préférences des usagers, alors ne serait-il pas intéressant de réduire la dimensionnalité afin d'obtenir des préférences plus générales? L'idée derrière la "*Singular Value Decomposition (SVD)*" est de réduire la dimensionnalité de la matrice originale en trois morceaux. L'algèbre linéaire nous garanti que si nous avons une matrice, nous pouvons la factoriser en trois matrices. Rappelons-nous que par factorisation, on peut penser à des chiffres, par exemple 12, qui pourrait être factorisé en 3 et 4. Si l'on multiplie 3 et 4, nous obtenons notre chiffre original; 12. Évidemment, factoriser de cette façon notre matrice ne serait pas intéressant. Nous voulons plutôt réduire la dimensionalité des matrices afin d'obtenir la meilleure approximation possible de la matrice originale, ce qu'on appelle "*rank K approximation*" (Michael Ekstrand et Joe Konstan, s. d.). Ceci nous permet notamment d'éliminer le bruit. Les prédictions sont donc basées de la matrice reconstruite. L'équation de SVD est la suivante:

$$R{=}P\Sigma Q^T$$

R est une matrice m x n d'évaluations

P est une matrice m x k usager-dimensions

Q est une matrice n x k item-dimensions

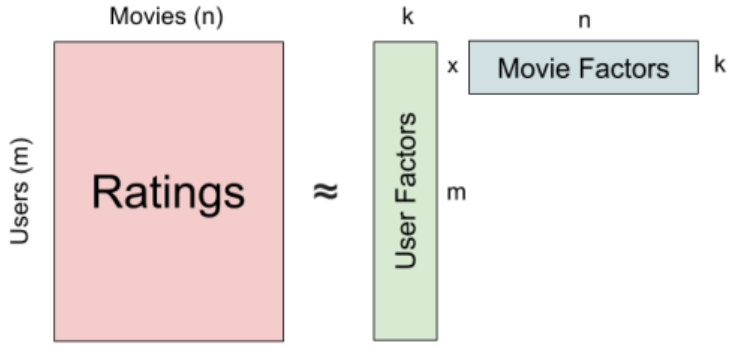
Σ est une matrice diagonale

Un des inconvénients majeurs de la méthode SVD est que la matrice originale doit être complète. Ceci est problématique, car si nous avions une matrice d'évaluations complètes, nous n'aurions pas besoin d'effectuer des recommandations. Ainsi, on doit procéder à une imputation avant d'effectuer une SVD, ce qui engendre son lot de problèmes.

3.3.2 FunkSVD

La méthode proposée par Simon Funk est basé sur l'intuition derrière SVD (bien qu'elle ne constitue pas une décomposition de matrice au niveau mathématique). L'objectif est de

décomposer la matrice originale en **deux** matrices représentant le plus fidèlement possible l'original. Pour ce faire, on peut trouver le "*best rank K approximation*" basé sur l'erreur quadratique moyenne au carré pour les observations que nous avons, permettant ainsi d'ignorer les observations manquantes (Michael Ekstrand et Joe Konstan, s. d.). La méthode de descente du gradient nous permet de trouver les deux matrices qui minimisent l'erreur. Pour ce faire, les valeurs des deux matrices de facteurs sont initialisées de manière aléatoire et itérativement, on calcule l'erreur basée sur les valeurs pour lesquelles nous avons une observation dans la matrice originale.



Dans la figure ci-dessus, k est un hyperparamètre représentant les dimensions latentes (ou les "*features*" cachées). Une dimension ou variable latente est une dimension qui n'est pas directement observable, mais qu'on peut déduire des données. Ainsi, on espère pouvoir utiliser ces dimensions latentes afin de modéliser les préférences des utilisateurs. Un choix de k élevé augmente nécessairement la personnalisation des recommandations, mais mène inévitablement à du surapprentissage. Un k de 1 représenterait l'item le plus populaire. À noter qu'afin d'éviter le surapprentissage, il est recommandé d'utiliser la régularisation (Michael Ekstrand et Joe Konstan, s. d.).

3.3.3 Alternating Least Squares (ALS)

La descente du gradient mentionné à la section précédente est l'une des méthodes d'optimisation utilisées pour la factorisation de matrices. Malgré qu'elle fonctionne généralement bien, lorsque nous sommes confrontés à des situations dans lesquelles nous avons une énorme quantité de données (téraoctets ou pétaoctets), il est préférable d'utiliser la méthode "alternating least square" (ALS), car celle-ci est plus adapté au calcul en parallèle. En effet, avec la méthode ALS, les vecteurs de dimensions sont ajustés un à la fois. Une application particulièrement intéressante de ALS est lorsque nous avons des retours d'informations implicites, contrairement à des notes explicites. Les quantités de données issues des interactions des usagers avec le système rendent la méthode de descente du gradient (SGD) très peu

pratique pour le développement de systèmes de recommandations. De plus, dans le cas du "*One-Class CF (OC-CF)*", c'est-à-dire lorsque nous avons uniquement des informations binaires d'interactions, la méthode *SGD* nécessite que l'on remplace les valeurs manquantes par 0, autrement le modèle prédira toujours 1. La matrice résultante est donc énorme, rendant ainsi le calcul impracticable. À noter que la performance est généralement moins bonne qu'avec SGD, selon l'étude de Christophe Aberger (Aberger, s. d.).

Nous avons mentionné que les retours d'informations implicites peuvent être sous forme binaire. Toutefois, nous pouvons également avoir une quantité, tel que le nombre de fois qu'un utilisateur a écouté une chanson. Hu et al. proposent une méthode afin de mesurer les préférences des usagers basé sur les retours d'informations implicites. Nous avons deux mesures: la préférence P et la confiance C.

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} < 0 \end{cases}$$

La valeur binaire de préférence p indique simplement si nous avons un retour d'information dans les données. La valeur r est le retour d'information.

$$c_{ui} = 1 + \alpha r_{ui}$$

La mesure de confiance c est calculé en fonction de l'ampleur de r. Ainsi, un usager ayant écouté plus souvent une chanson aura une plus grande mesure de confiance, indiquant sa préférence pour celle-ci. Même si nous n'avons aucun retour d'information, c sera 1, n'excluant pas la possiblité que l'utilisateur aimerait ce contenu. Le choix du paramètre α doit être évalué selon le problème. Les auteurs proposent ensuite d'utiliser la méthode ALS, dû à l'importante taille de la matrice résultante [hu_collaborative_2008].

Il est important de noter qu'il s'agit d'une des nombreuses approches possibles. Nous allons voir lors des exemples en R une autre solution pour quantifier les préférences en ajustant les valeurs manquantes; c'est l'approche ALS pondérée (Pan et al. 2008).

3.4 Les approches hybride

Cette technique se base sur plusieurs méthodes de recommandation à la fois. Son objectif est de contourner les inconvénients des approches vu plus haut et de proposer des recommandations plus fines . Ces modèles peuvent se baser sur n'importe quel type de données disponibles.

Selon Burke (2002) nous avons sept différents types de méthodes d'hybridation (Burke 2002). Le tableau ci-dessous présente ces méthodes.

Methode	Description
Pondérée	Les résultats pondérés de plusieurs techniques de recommandation sont combinés pour produire une nouvelle recommandation.
Permutation	Le système permute entre les différentes techniques de recommandation selon le résultat de la recommandation.
Mixte	Les recommandations de plusieurs techniques sont présentées en même temps.
Combinaison	Différentes techniques de recommandation sont combinées en un unique algorithme de recommandation.
Augmentation	Le résultat (output) d'une technique de recommandation est utilisé comme données en entrée (input) pour l'autre technique.
En cascade	En cascade & Un système de recommandation raffine les résultats fournis par un autre système.
Méta-Niveau	Le modèle appris par une technique de recommandation est utilisé comme données en entrée (input) pour l'autre technique.

3.5 Autres approches

La factorisation machine est une généralisation des méthodes de factorisation. Elle permet de prendre en compte les informations de contextes (démographiques, géographiques, ... etc.)

Les méthodes *Deep Learning* peuvent également étre appliquées au filtrage collaboratif. Elles permettent de passer d'un traitement statique des algorithmes classiques à un traitement plus dynamique. Les réseaux neuronaux profonds (MLP, Auto-encodeur, CNNs, RNNs, etc.) sont capables de modéliser les interactions non linéaires entre les utilisateurs et les items, ce qui permet d'avoir des modèles d'interaction complexe pour déterminer précisément les préférences des usagers.

4 Évaluation des systèmes de recommandations

L'évaluation des systèmes de recommandations n'est pas une tâche tout à fait claire. Le choix de la bonne mesure à optimiser est un obstacle majeur. La pertinence d'une métrique dépend des caractéristiques des données et du type de tâches que le système de recommandations doit effectuer. Nous proposons ici deux catégories de mesures générales.

— **Mesures de précision statistique (MAE, RMSE)**: Ces mesures sont utilisées dans le contexte de retour d'information explicite ou l'objectif est de prédire une note.

On tente donc de mesurer la différence entre les notes prédites et les notes réelles des usagers. Les mesures couramment utilisées sont: l'erreur absolue moyenne (MAE), l'erreur quadratique moyenne (RMSE). Plus elles sont faibles, meilleure est la précision du modèle.

— **Précision et rappel**: Ce sont les deux mesures d'évaluation les plus populaires des systèmes de recommandation, car on tente d'évaluer la qualité des recommandations, ce qui est généralement plus près de la réalitée d'affaires. La précision est le nombre des recommandations correctes sur le total des recommandations possibles. On cherche à quantifier la quantité d'items recommandés qui sont pertinents. Le rappel est la capacité du modèle à trouver tous les éléments pertinents et à les recommander à l'utilisateur. On cherche donc à quantifier le nombre d'items susceptible d'intéresser l'utilisateur identifiés parmi tous les items. Ces méthodes d'évaluation considèrent la procédure de prédiction comme une opération binaire qui distingue les bons items des de ceux qui ne le sont pas. Leurs formules sont:

$$Précision = \frac{\text{Nombre de bonnes recommandations}}{\text{Nombre total de recommandations}}$$

$$Rappel = \frac{\text{Nombre de bonnes recommandations}}{\text{Nombre total de bonnes recommandations possibles}}$$

5 Revue des ressources R

Nous avons ci-dessous la liste des principaux packages R pour développer des systèmes de recommandations. Malheureusement, il n'existe pas de librairie complète pour les systèmes de recommandations.

5.1 rrecsys

Traite les ensembles de données de recommandations standard en entrée et génère des prévisions d'évaluations et des listes d'items recommandés. Plusieurs mesures de performance sont disponibles (MAE, RMSE, Precision, Recall, ...) La liste d'algorithmes disponible est la suivante:

— Moyenne globale/par item/par usager/plus populaire
— KNN basé items
— FunkSVD
— Bayesian Personalized Ranking (BPR)

— Weighted ALS

5.2 recommenderLab

Il s'agit de la librairie de systèmes de recommandations la plus populaire et la plus complète sur R. Elle permet de facilement développer et tester plusieurs algorithmes de recommandations. Notamment, on y a implanté des outils de séparation des données, de validation croisée, de prédiction d'évaluations et de listes, ainsi que diverses mesures d'erreurs. Voici une liste non-exhaustive d'algorithmes disponibles:

— User-based collaborative filtering (UBCF)
— Item-based collaborative filtering (IBCF)
— FunkSVD
— ALS
— Recommandations hybrides

À noter que l'implantation de FunkSVD est basé sur l'implantation de la libraire **rrecsys***. La dernière mise à jour de cette librairie sur GitHub fût il y a 7 mois. Ce sera avec cette librairie que nous effectuerons la majorité de nos analyses.

5.3 slimrec

Il s'agit tout simplement d'une librairie qui a implémenté la méthode SLIM. La méthode est basée sur une combinaison linéaire des évaluations (implicites ou explicites) des usagers. Les coefficients sont ajustés avec la régression *elastic net*. La librairie comprend une fonction permettant d'optimiser alpha par validation croisée. *SLIMreC* est basé sur la librairie est basée sur **glmnet**. La librairie ne sera pas utilisée dans ce tutoriel, mais dans la vignette de SLIMrec, vous pouvez trouver des exemples très simples.

5.4 Rsparse

Les auteurs de la librairie ont implantés des algorithmes tels que *ALS*, *Factorization machines*, *Eslatic Net regression* et *SVD*. *Rsparse* est utile car les calculs sont nettement plus rapide que les autres librairies. Toutefois, ce n'est pas une librairie exclusivement pour les systèmes de recommandations. Ainsi, d'un point de vue d'introduction au sujet, elle n'est pas notre premier choix. Cependant, si vous désirez utiliser un modèle et que vous avez plus d'éxperience, *Rsparse* est définitivement la librairie la plus intéressante.

5.5 recosystem

C'est une couverture R de LIBFM (une bibliothèque C++ open source pour les systèmes de recommandation utilisant la factorisation matricielle parallèle). Le calcul parallèle multicoeur hautes performances est pris en charge dans ce package.

6 Exemples d'analyses de données

Pour les analyses à venir, veuillez vous assurer que les librairies suivantes sont installées:

- rrecsys
- recommenderlab
- RColorBrewer
- rsparse

ATTENTION: Certains des exemples peuvent être long. Une approximation du temps d'exécution est fourni à titre indicatif. Vous pouvez également observer que lorsque nous effectuons une prédiction, souvent nous spécifions le nom de la librairie (recommenderlab::predict, par exemple), car sinon cela crée des conflits et génère une erreur (plusieurs librairies ont la même fonction...) . Aussi, veuillez vous assurer d'avoir le bon chemin pour l'import des données last.fm.

6.1 Évaluations explicites

6.1.1 Description des données et de l'outil

Pour les exemples en R, nous allons utiliser la base de données MovieLens 100k, inclus dans la librairie recommenderlab. Il s'agit d'un jeu de données classique dans la littérature. On y retrouve 99 392 évaluations de 943 usagers sur 1 664 films. Les évaluations sont basées sur une échelle de 1 à 5, et chaque usagers a évalué un minimum de 20 films. Ainsi, nous n'avons pas à se soucier du problème de "cold-start".

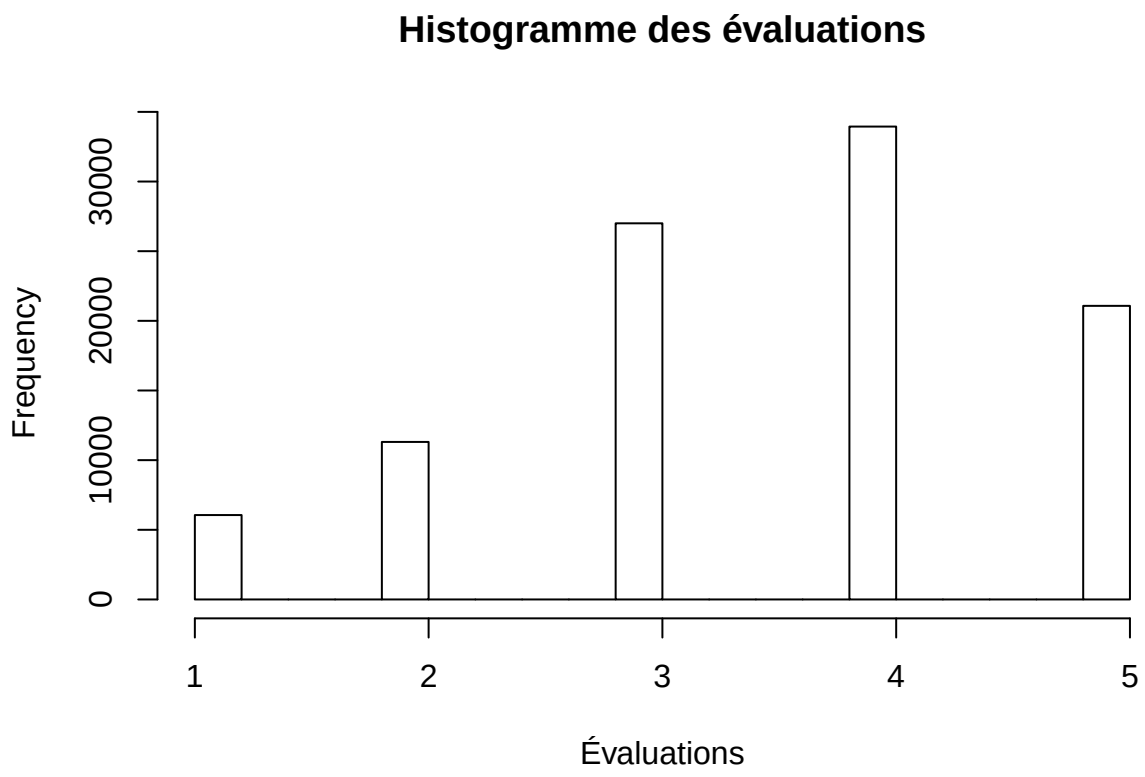
```
# Librairies Requises en ordre d'utilisation

packages <- c("recommenderlab", "RColorBrewer", "rrecsys", "rsparse", "mlapi")
if (length(setdiff(packages, rownames(installed.packages()))) > 0) {
  install.packages(setdiff(packages, rownames(installed.packages())))
}
```

```r
library(recommenderlab)
library(RColorBrewer)
library(rrecsys)
library(mlapi)
library(rsparse)

# Données pour la partie sur recommandation implicites.
# Veuillez vous assurer d'avoir le bon chemin

# Import des données
rawdata <- read.csv("lastfm.csv")

#**********************************
#**ATTENTION**: Certains des exemples peuvent être long.
#Une approximation du temps d'exécution est fourni à titre
#indicatif. Vous pouvez également observer que lorsque
#nous effectuons une prédiction, souvent nous spécifions
#le nom de la librairie (recommenderlab::predict, par exemple),
#car sinon cela crée des conflits et génère une erreur
#(plusieurs librairies ont la même fonction...) . Aussi,
#veuillez vous assurer d'avoir le bon chemin pour l'import
#des données last.fm.
#**********************************
```

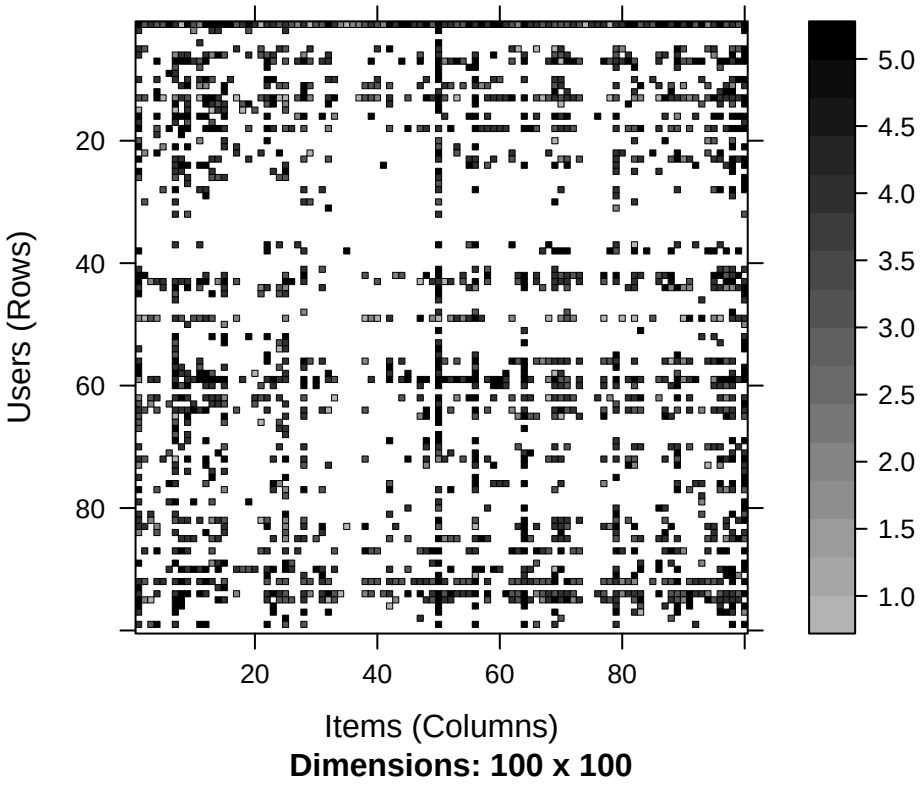
```r
# Import des données
data("MovieLense")

# Exploration des donnees
head <- MovieLense[c(1:5), c(1:3)]
as(head, "matrix")
```

```
##   Toy Story (1995) GoldenEye (1995) Four Rooms (1995)
## 1                5                3                 4
## 2                4               NA                NA
## 3               NA               NA                NA
## 4               NA               NA                NA
## 5                4                3                NA
```

```r
# Histogramme des evaluations
hist(getRatings(MovieLense),
     breaks=15,
     xlab='Évaluations',
     main='Histogramme des évaluations')
```




```r
image(MovieLense[1:100, 1:100])
```



On peut remarquer que les données sont sous forme d'une matrice de 943 usagers x 1664 films, dont chaque valeurs est une note sur 5. La classe realRatingMatrix est unique à la librairie et est optimisé pour stocker de manière efficace les matrices parcimonieuses comportant des évaluations (généralement 1 à 5). Nous pouvons constater la représentation *sparse* dans les figures ci-dessus.

6.1.2 Note sur le format des matrices

Afin d'utiliser recommenderLab, vos matrices doivent être de type realRatingMatrix ou binaryRatingMatrix. Dans notre exemple, nous importons les données sous forme realRating-Matrix. Nous pouvons toutefois convertir une matrice régulière sous forme usager x item, comportant des évaluations parcimonieuse, avec la fonction as(have, "realRatingMatrix).

Le contraire est également possible, comme démontré à la section 6.1.1 pour afficher le début de la matrice.

À noter que le nom "*rating*" dans *realRatingMatrix* ou *binaryRatingMatrix* pourrait indiquer une évaluation explicite de l'utilisateur, mais cela peut également indiquer une évaluation implicite.

6.1.3 La division d'un échantillon

On pourrait simplement diviser l'échantillon comme dans un problème régulier, mais puisque l'objectif est de baser nos prédictions sur des consommations passées, c'est pourquoi *recommenderlab* offre plus de flexibilité.

```
# Séparation du jeu de données en train et test
# grâce à la fonction evaluationScheme
set.seed(12345)
e_scheme <- evaluationScheme(MovieLense,
                             method="split",
                             train=0.7,
                             given=4,
                             goodRating=4)
```

Pour la méthode *evaluationScheme*, trois options s'offrent à nous: split, cross-validation et bootstrap. Dans cet exemple, on sélectionne aléatoirement 70% des usagers pour l'ensemble d'entraînement et quatre évaluations par usagers de l'ensemble test seront fournies au modèle afin de faire une prédiction. C'est ce qu'on appelle le schéma d'évaluation. Les autres évaluations seront celles utilisées pour calculer l'erreur. L'option "goodRating" permet de définir la valeur d'une évaluation qu'on considère positive.

6.1.4 Les modèles

La commande suivante permet d'obtenir les modèles disponibles selon le type de matrice:

```
# Modeles disponibles pour "realRatingMatrix"
names(recommenderRegistry$get_entries(dataType = "realRatingMatrix"))
```

```
##  [1] "ALS_realRatingMatrix"          "ALS_implicit_realRatingMatrix"
##  [3] "IBCF_realRatingMatrix"         "LIBMF_realRatingMatrix"
##  [5] "POPULAR_realRatingMatrix"      "RANDOM_realRatingMatrix"
##  [7] "RERECOMMEND_realRatingMatrix"  "SVD_realRatingMatrix"
##  [9] "SVDF_realRatingMatrix"         "UBCF_realRatingMatrix"
```

Dans ce cas, nous avons la liste des modèles pour les matrices comportant des notes d'utilisateurs.

6.1.4.1 La méthode de recommandation par popularité La première méthode de recommandation est basée sur la popularité, c'est-à-dire le nombre d'usagers ayant accordés

une évaluation à un film. Malgré que cette technique n'est pas sophistiquée, elle permet généralement d'offrir des recommandations très fiables et permet notamment de pallier au problème de "cold-start".

```
# Premier modèle - films populaires
# Temps d'exécution: 0.3361001 secs

model_pop <- Recommender(MovieLense[1:500], method = "POPULAR")
recom_pop <- recommenderlab::predict(model_pop, MovieLense[501:502], n=5)
```

Dans ce premier exemple, nous n'allons pas utiliser l'ensemble d'entraînement et de test défini à la section précédente. L'objectif sera plutôt de recommander 5 films pour 2 usagers. Il est toutefois possible de prédire l'évaluation à partir de la méthode "Popular". À la première ligne, nous bâtissons notre liste de films les plus populaires basé sur les 500 premier utilisateurs. Ensuite, nous effectuons une prédiction des 5 films (n=5) les plus susceptibles d'intéresser les utilisateurs 501 et 502. Voici le résultat:

```
as(recom_pop, "list")
```

```
## $`501`
## [1] "Star Wars (1977)"                    "Silence of the Lambs, The (1991)"
## [3] "Raiders of the Lost Ark (1981)"      "Pulp Fiction (1994)"
## [5] "Shawshank Redemption, The (1994)"
##
## $`502`
## [1] "Star Wars (1977)"                    "Fargo (1996)"
## [3] "Silence of the Lambs, The (1991)"    "Godfather, The (1972)"
## [5] "Raiders of the Lost Ark (1981)"
```

6.1.4.2 La méthode de recommandation "user-based collaborative filtering"

Dans cet exemple, nous allons utiliser la méthode user-based CF détaillée à la section 3.1. À noter que dans l'exemple précédent, nous avions comme sortie une liste des top-5 recommandations. Dans les exemples suivants, nous allons plutôt tenter de prédire les évaluations. Toutefois, il est toujours possible d'obtenir une liste de recommandations.

```
# Second modèle - User-based collaborative filtering
# Temps d'exécution: 4.44511 secs

UBCF_params <- list(method="cosine",
```

```
                    nn = 10,
                    sample = FALSE,
                    normalize="center")

model_UBCF <- Recommender(getData(e_scheme, "train"),
                          method="UBCF",
                          parameter= UBCF_params)

pred_UBCF <- recommenderlab::predict(model_UBCF,
                                getData(e_scheme, "known"),
                                type="ratings")

error_UBCF <- calcPredictionAccuracy(pred_UBCF,
                                     getData(e_scheme, "unknown"))
error_UBCF
```

```
##      RMSE       MSE       MAE
## 1.1437154 1.3080850 0.9061818
```

La première ligne nous permet de définir les paramètres utilisés pour trouver les plus proches voisins (method = "cosine") et le nombre de voisins pris en considérations (nn = 10). Comme nous utilisons les notes accordés par les usagers, il est important de normaliser ces valeurs (normalize = "center"). La seconde ligne permet d'ajuster le modèle. La méthode *getData* permet d'accéder aux données d'entraînement définies à la section 6.1.3. La troisième ligne permet d'effectuer la prédiction pour les données test. Rappelez-vous qu'à la section 6.1.3, nous avons mentionné que 4 notes faisant partie de l'ensemble de test seront donnés. Nous utilisons ainsi *getData* pour fourni les évaluations de test qui ne seront pas cachés ("*known*"). Finalement, nous calculons l'erreur du modèle avec *calcPredictionAccuracy*, sur les évaluations inconnues ("*unknown*").

6.1.4.3 La méthode de recommandation "item-based collaborative filtering"

Nous poursuivons notre analyse avec le filtrage collaboratif basé items, décrits à la section 3.2.

```
# Troisième modèle - Item-based collaborative filtering
# Temps d'exécution: 45.73366 secs
```

```
IBCF_params <- list(method="cosine", k = 10, normalize="center")

model_IBCF <- Recommender(getData(e_scheme, "train"),
                          method="IBCF",
                          parameter= IBCF_params)

pred_IBCF <- recommenderlab::predict(model_IBCF, getData(e_scheme, "known"),
                                     type="ratings")

error_IBCF <- calcPredictionAccuracy(pred_IBCF, getData(e_scheme, "unknown"))
error_IBCF
```

```
##      RMSE       MSE       MAE
## 1.3014905 1.6938776 0.9591837
```

Les étapes sont pratiquement les mêmes que pour le modèle précédent. Nous avons mentionné plus tôt que cette méthode offre généralement des résultats moins intéressants que l'approche basée utilisateurs. C'est ce que nos résultats semblent également indiquer. Par contre, il faut savoir que cette méthode est plus "*scalable*"; il ne faut donc pas se baser uniquement sur la mesure d'erreur afin de faire notre choix.

6.1.4.4 La méthode SVD

Rappelons-nous que la méthode SVD requiert une matrice complète. Ainsi, dans *recommenderLab*, l'auteur a décidé d'implanter cette méthode en imputant les données manquantes par la moyenne des colonnes afin de faire la décomposition.

```
# Quatrième modèle - SVD (column mean imputation)
# Temps approximatif: 1.022266 secs

svd_params <- list(k=13, maxiter=200, normalize="center")

model_svd <- Recommender(getData(e_scheme, "train"),
                         method="SVD",
                         parameter=svd_params)

pred_svd <- recommenderlab::predict(model_svd,
                                    getData(e_scheme, "known"),
                                    type="ratings")
```

```
error_svd <- calcPredictionAccuracy(pred_svd, getData(e_scheme, "unknown"))
error_svd
```

```
##      RMSE       MSE       MAE
## 1.1513705 1.3256541 0.9124495
```

La méthode SVD a pour objectif de trouver la matrice la plus semblable possible à l'original à partir de matrices à dimensionnalité réduite. Ainsi, nous devons spécifier le nombre d'itérations (*maxiter*) ainsi que le nombre de dimensions latentes (k). Les résultats dépendent hautement de la valeur de l'hyperparamètre k. Ici, nous n'essayons pas d'optimiser k, mais cela devrait être fait afin d'obtenir les meilleurs résultats possibles.

6.1.4.5 Simon Funk's Singular Value Decomposition (FunkSVD)

Nous allons utiliser la factorisation de matrices pour ce modèle; plus particulièrement la technique développée par Simon Funk détaillée à la section 3.3.2.

```
# Cinquième modèle - FunkSVD
# Temps approximatif: 1.869445 mins

funk_params <- list(k = 13, lambda = 0.001, min_epochs=50, max_epochs=100,
                    normalize="center")

model_funk <- Recommender(getData(e_scheme, "train"), method="SVDF",
                          parameter = funk_params)

pred_funk <- recommenderlab::predict(model_funk,
                                     getData(e_scheme, "known"),
                                      type="ratings")

error_funk <- calcPredictionAccuracy(pred_funk,
                                     getData(e_scheme, "unknown"))
error_funk
```

```
##      RMSE       MSE       MAE
## 1.1228698 1.2608366 0.8867373
```

La méthode *FunkSVD* utilise la descente du gradient afin de trouver les valeurs optimales dans les deux matrices pour chaque dimension. Le *lambda* est donc le taux d'apprentissage,

tandis que le *min_epochs* et *max_epochs* représentent les itérations minimales et maximales. Le gamma est le terme de régularisation.

6.1.4.6 Système de recommandation hybride Nous avons vu plusieurs méthodes différentes, chacune ayant ses forces et faiblesses. *RecommenderLab* offre la possibilité de jumeler plusieurs méthodes afin de générer une seule prédiction, basé sur la méthode pondérée.

```
# Sixième modèle - Système de recommandation hybride
# Temps d'exécution : 43.18847 secs

h_model <- HybridRecommender(Recommender(MovieLense, method="POPULAR"),
                             Recommender(MovieLense, method="UBCF"),
                             Recommender(MovieLense, method="IBCF"),
                             Recommender(MovieLense, method="RERECOMMEND"),
                             Recommender(MovieLense, method="RANDOM"),
                             weights = c(0.3, 0.2, 0.3, 0.1, 0.1)
                             )

getList(recommenderlab::predict(h_model, 501:502, MovieLense, n=5))
```

```
## $`501`
## [1] "Aiqing wansui (1994)"
## [2] "Saint of Fort Washington, The (1993)"
## [3] "Marlene Dietrich: Shadow and Light (1996) "
## [4] "Entertaining Angels: The Dorothy Day Story (1996)"
## [5] "Mrs. Dalloway (1997)"
##
## $`502`
## [1] "Ed's Next Move (1996)"
## [2] "Stripes (1981)"
## [3] "That Old Feeling (1997)"
## [4] "Fresh (1994)"
## [5] "I Don't Want to Talk About It (De eso no se habla) (1993)"
```

6.1.5 Évaluation des modèles

6.1.5.1 Évaluations des listes de recommandations Nous avons précédemment démontré comment obtenir les meilleures recommandations pour chaque usagers. Nous pouvons

comparer la performance des modèles en utilisant une matrice de confusion. *RecommenderLab* nous permet de facilement évaluer les recommandations de plusieurs modèles avec la fonction *evaluate()*.

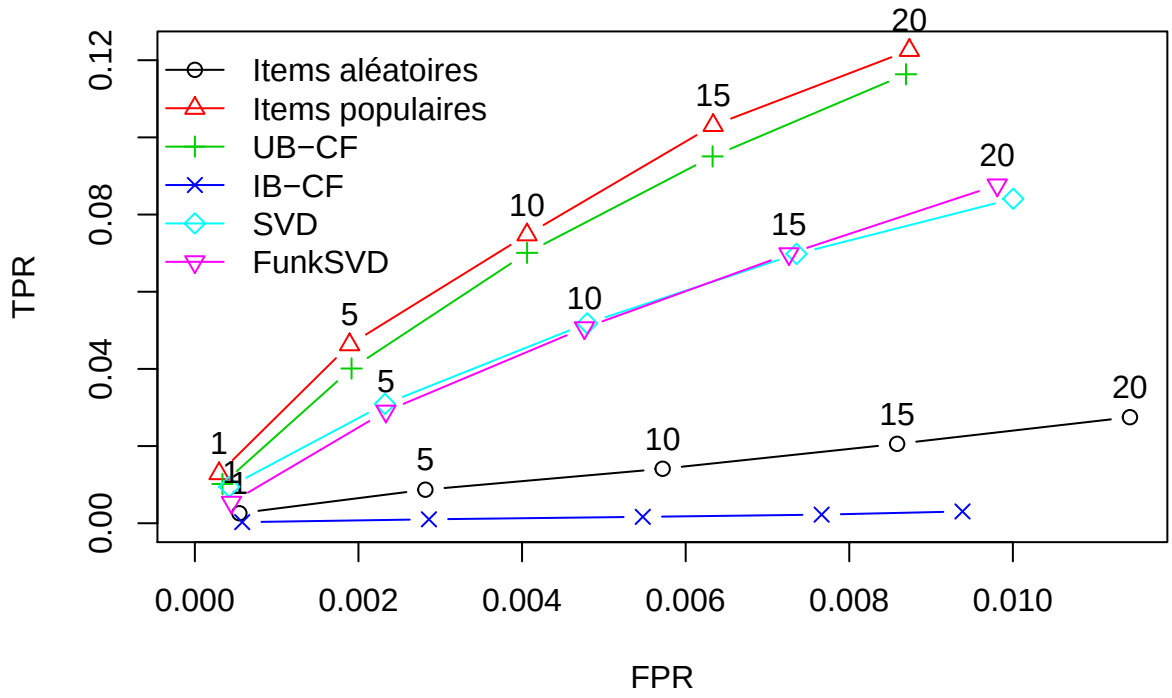
```
# Évaluations des recommandations
# Temps d'exécution: 5.904286 mins

set.seed(12345)
c_scheme <- evaluationScheme(MovieLense,
                             method="cross", k=2, given=4, goodRating=4)

compared_algos <- list( "Items aléatoires" = list(name="RANDOM", param=NULL),
                        "Items populaires" =
                          list(name="POPULAR", param=list(normalize="center")),

                        "UB-CF" =
                          list(name="UBCF", param=list(nn=40,
                                                       method="cosine",
                                                       normalize="center")),

                        "IB-CF" =
                          list(name="IBCF", param=list(k=40,
                                                       normalize="center")),

                        "SVD" =
                          list(name="SVD", param=list(k=13,
                                                      normalize="center")),

                        "FunkSVD" =
                          list(name="SVDF", param=list(k=13,
                                                       normalize="center"))
                        )

# Note: Meme si le k-fold est petit, cette commande prend du temps à executer
# Pour top-N recommandations
topn_comp <- evaluate(c_scheme, compared_algos,
```

```
                        type="topNList", n=c(1,5,10,15,20))
```

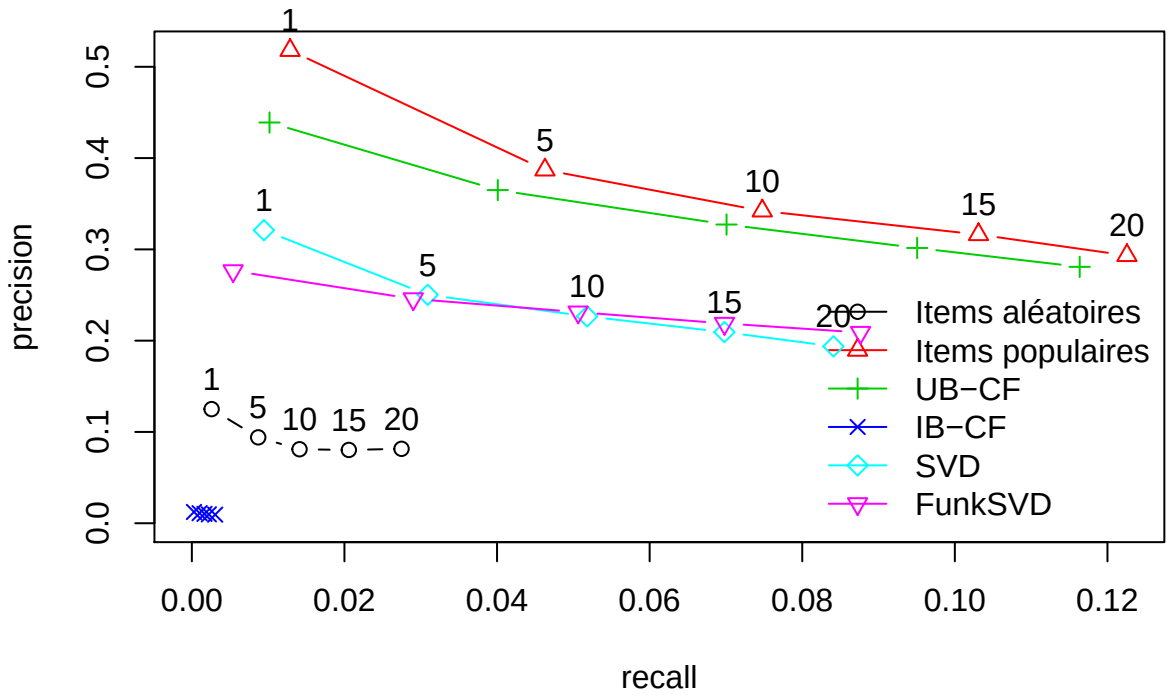
```
## RANDOM run fold/sample [model time/prediction time]
##   1  [0sec/1.34sec]
##   2  [0sec/1.07sec]
## POPULAR run fold/sample [model time/prediction time]
##   1  [0.02sec/4.39sec]
##   2  [0sec/4.07sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/4.5sec]
##   2  [0sec/4.26sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [37.02sec/0.22sec]
##   2  [34.91sec/0.23sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.46sec/0.87sec]
##   2  [0.19sec/1.21sec]
## SVDF run fold/sample [model time/prediction time]
##   1  [85.62sec/56.51sec]
##   2  [72.82sec/42.52sec]
```

Dans cet exemple, nous avons créé un *evaluationScheme* avec validation croisée pour démontrer la fonctionnalité. Par soucis de rapidité, nous avons limité le nombre de *folds*. Nous passons à la fonction *evaluate* notre liste d'algorithmes à tester et demandons en sortie une liste des meilleures recommandations. L'argument n accepte les vecteurs. Ainsi, nous demandons différentes listes avec 1, 5, 10, 15 et 20 recommandations respectivement. Le graphique généré permet de visualiser les résultats sous forme de courbe ROC et précision/rappel.

```
# Graphique de la courbe ROC
plot(topn_comp, annotate=c(1,6,2), legend='topleft')
```




```
# Graphique de precision/rappel
plot(topn_comp, "prec/rec", annotate=c(1,2,5))
```



6.1.5.2 Évaluations des prédictions d'évaluations Nous pouvons également utiliser la fonction *evaluate()* pour comparer les prédictions avec la même liste d'algorithmes que précédemment.

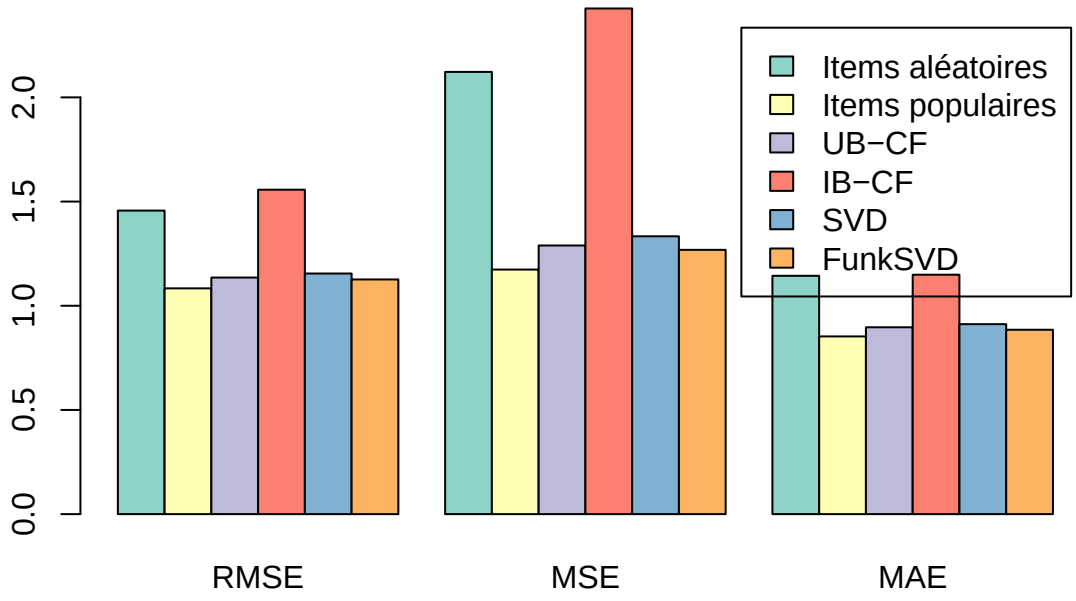
```
coul <- brewer.pal(6, "Set3")
```

```
# Évaluations des prédictions d'évaluations
ratings_comp <- evaluate(c_scheme, compared_algos, type="ratings")
```

```
## RANDOM run fold/sample [model time/prediction time]
##   1  [0sec/0.97sec]
##   2  [0.02sec/0.7sec]
## POPULAR run fold/sample [model time/prediction time]
##   1  [0.01sec/0.46sec]
##   2  [0.02sec/0.28sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.01sec/4.07sec]
##   2  [0.02sec/3.84sec]
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [34.26sec/0.16sec]
##   2  [34.41sec/0.15sec]
## SVD run fold/sample [model time/prediction time]
##   1  [0.24sec/0.58sec]
##   2  [0.18sec/0.85sec]
## SVDF run fold/sample [model time/prediction time]
##   1  [86.58sec/59.7sec]
##   2  [71.31sec/42.72sec]
```

```r
# Histogramme
plot(ratings_comp, col=coul)
```



6.1.5.3 Méthode wALS avec la librairie rrecsys Pour cet exemple, nous allons utiliser la librairie *rrecsys*, que nous considérons moins facile à utiliser et moins flexible que *recommenderlab*. Toutefois, on y trouve quelques modèles intéressants dont une implantation de *weighted ALS* proposé par Pan et al.

```
# Transformation de realRatingMatrix
mat_ml <- as(MovieLense[1:150], "matrix")

# Definition des donnees (transformation)
ml_rrec <- defineData(mat_ml, sparseMatrix=FALSE,
                      binary=FALSE, minimum=1, maximum=5)
```

Nous utilisons les données de MovieLense provenant de *recommenderlab* afin de démontrer la transformation de *realRatingMatrix* à une matrice régulière, mais le même jeu de données est intégré à *rrecsys*. Puisque l'execution prend beaucoup de temps, nous allons restreindre la quantité de données, ce qui nuira à la qualité des recommandations. La fonction *defineData* est une fonction de *rrecsys* permettant de rendre utilisable les données dans les modèles de la librairie.

```
# Definition de l'evaluation du modele
set.seed(12345)
als_cv <- evalModel(ml_rrec, folds = 3)
```

La méthode *evalModel* permet de spécifier la méthode d'évaluation (similaire à *evaluationScheme* de *recommenderlab*). L'évaluation est faite selon la validation-croisée et le paramètre *folds* est le paramètre k dans *recommenderlab*. Grâce à l'échantillonage stratifié effectué par la fonction, chaque usagers sont représentés uniformément dans chaque *folds*.

On peut maintenant procéder à la sélection des paramètres de notre modèle *wALS*. Rappelons-nous qu'il y a plusieurs façons de quantifier la confiance envers les retours d'informations implicites et explicites. Pour le *weighted Alternated Least Squares* avec *rrecsys*, nous avons les options suivantes pour les valeurs manquantes, basé sur Pan et al.:

Methode	Description
Uniform (uni)	Les données manquantes ont sont attribuées la valeur delta.
User-oriented (uo)	La confiance est déterminée en fonction du nombre d'évaluations (ou d'intéractions dans le cas implicite) de l'usager. On assume ainsi qu'un usager ayant beaucoup d'évaluations connait bien la librairie. Il est donc plus probable que les items qu'il n'a pas consommés l'intéressent moins.
Item-oriented (io)	La confiance est basée sur l'absence ou l'existance d'observations pour cet item. Si un item est moins souvent évalulé (ou consommé), alors la confiance que l'usager aime cet item sera plus basse.

```r
# Évaluation des predictions
# Temps d'exécution: 6.848136 mins

eval_als <- evalRec(als_cv, "wALS", k = 15,
                    scheme="uo",
                    positiveThreshold=4, topN=5)
```

```
## Evaluating top- 5  recommendation with  wALS .
## Total execution time: 151.3925 seconds.
##
## Fold: 1 / 3  elapsed. Time: 151.7926
##
## Total execution time: 152.9131 seconds.
##
## Fold: 2 / 3  elapsed. Time: 153.0308
##
## Total execution time: 144.9833 seconds.
##
## Fold: 3 / 3  elapsed. Time: 145.097
```

```r
eval_als
```

```
## Algorithm:  wALS
## Configuration:
##    k regCoef scheme
## 1 15    0.01     uo
##
##         precision    recall  ex.time
## 1-fold  0.3266667 0.1439099 151.7926
## 2-fold  0.3253333 0.1265340 153.0298
## 3-fold  0.3253333 0.1332946 145.0960
## Average 0.3257778 0.1345795 149.9728
```

Dans notre exemple, nous utilisons la méthode uniforme (uo). Comme dans *recommenderlab*, *positiveThreshold* permet d'indiquer à partir de quelle note l'on considère une évaluation positive. Le *topN* indique que nous souhaitons avons les dix meilleures recommandations pour chaque usagers.

6.2 Évaluations implicites

6.2.1 Description des données

Nous avons beaucoup abordé le sujet d'évaluations implicites, sans toutefois travailler directement avec ce genre de données. Ainsi, nous avons jugé pertinent d'inclure au moins une analyse.

Le jeu de données est issu de *last.fm*, un site web qui permet d'écouter de la musique. Une des particularitées qui a fait la renommé de l'entreprise est le "*Audioscrobbler*", un système de recommandation qui permet de construire un profil détaillé des goûts musicaux de chaque utilisateur afin d'effectuer des recommandations. L'entreprise a rendu libre d'accès plusieurs jeux de données, dont celle qui fera l'objet de notre analyse. Cependant, nous avons uniquement sélectionné un petit sous-ensemble, car la taille de l'original est démesurée. Si vous êtes intéressés à obtenir les jeux de données complets, vous pouvez utiliser l'*API* de *last.fm*.

```
# Exploration des données
head(rawdata)
```

userid	artistid	artistname
00000c289a1829a808ac09c00daf10bc3c4e223b	f2fb0ff0-5679-42ec-a55c-15109ce6e320	die Ã„rzte
00000c289a1829a808ac09c00daf10bc3c4e223b	b3ae82c2-e60b-4551-a76d-6620f1b456aa	melissa etherid
00000c289a1829a808ac09c00daf10bc3c4e223b	3d6bbeb7-f90e-4d10-b440-e153c0d10b53	elvenking
00000c289a1829a808ac09c00daf10bc3c4e223b	bbd2ffd7-17f4-4506-8572-c1ea58c3f9a8	juliette & the l
00000c289a1829a808ac09c00daf10bc3c4e223b	8bfac288-ccc5-448d-9573-c33ea2aa5c30	red hot chili pe
00000c289a1829a808ac09c00daf10bc3c4e223b	6531c8b1-76ea-4141-b270-eb1ac5b41375	magica

```
# Nombre unique de client
length(unique(rawdata[['userid']]))
```

```
## [1] 8875
```

```
# Nombre d'artistes
length(unique(rawdata[['artistid']]))
```

```
## [1] 46547
```

```
# Pour notre analyse, on retire le nom de l'artiste
lastfm <- rawdata[,-3]
```

Nous avons donc quatre colonnes; l'ID de l'usager, l'ID de l'artiste (à noter que dans la base

de données originale, certains artistes n'ont pas d'ID), le nom de l'artiste et le nombre de lectures. Nous retirons le nom de l'artiste de notre analyse.

6.2.2 Préparation des données

```
# Construction de la matrice d'interaction.
matlastfm <- sparseMatrix(i = as.integer(lastfm$userid),
                          j = as.integer(lastfm$artistid),
                          x = lastfm$plays)

# On retire les usagers qui ont moins de 15 observations d'écoute
matlastfm <- matlastfm[tabulate(summary(matlastfm)$i) > 15, , drop=FALSE]
dim(matlastfm)
```

```
## [1]  8826 46547
```

Considérant que nos données sont sous forme de *dataframe*, nous devons les transformers en *sparsematrix*. Puisque *userid* et *artistid* sont des facteurs, nous devons les convertir en nombres. De plus, nous retirons les usagers n'ayant pas assez d'observations.

6.2.3 Modèle ALS basé sur Hu et al.

```
# Base sur Hu, et al., fonction permettant d'ajouter une
# mesure de confiance

adjust_confidence <- function(x, alpha){
  mat_conf <- x
  mat_conf@x <- 1 + alpha * x@x
  mat_conf
}

conflastfm <- adjust_confidence(matlastfm, 0.1)
```

À la section sur ALS, nous avons introduit le concept de confiance, présenté par Hu et al. Nous créons donc une fonction permettant d'ajuster les entrées dans la matrice.

Pour cet exemple, nous utilisons la librairie *Rsparse*, plutôt que *rrecsys* ou *recommenderlab*. La rapidité de calcul est nettement supérieur aux autres librairies. En effet, dans l'exemple ci-dessous, nous allons tenter de prédire les 20 meilleures recommandations. Avec recommen-

derLab, cela prend autour de 10 minutes pour 12 000 observations. Avec *Rsparse*, l'algorithme prend 3.5 secondes pour près de 450 000 observations du même jeu de données. Les résultats sont équivalents.

```
# Séparation en entraînement/test
train <- matlastfm[1:5000,]
test <- matlastfm[5001:nrow(matlastfm),]

# Ajustement du modèle
model_alsr <- WRMF$new(rank=15, feedback="implicit", lambda=0.1)
fit_als <- model_alsr$fit_transform(train, n_iter=50)
```

```
## INFO  [20:47:57.695] starting factorization with 4 threads
## INFO  [20:47:57.989] iter 1 loss = 0.3222
## INFO  [20:47:58.091] iter 2 loss = 0.0871
## INFO  [20:47:58.177] iter 3 loss = 0.0663
## INFO  [20:47:58.261] iter 4 loss = 0.0604
## INFO  [20:47:58.345] iter 5 loss = 0.0576
## INFO  [20:47:58.429] iter 6 loss = 0.0559
## INFO  [20:47:58.514] iter 7 loss = 0.0549
## INFO  [20:47:58.599] iter 8 loss = 0.0541
## INFO  [20:47:58.684] iter 9 loss = 0.0535
## INFO  [20:47:58.769] iter 10 loss = 0.0530
## INFO  [20:47:58.850] iter 11 loss = 0.0527
## INFO  [20:47:58.938] iter 12 loss = 0.0524
## INFO  [20:47:59.021] iter 13 loss = 0.0521
## INFO  [20:47:59.022] Converged after 13 iterations
```

Dans *Rsparse*, *rank* signifie le nombre de dimensions latentes. Lambda est le paramètre de régularisation. Nous devons également spécifier le type d'évaluations, car le type d'algorithme utilisé variera en conséquence. D'autres paramètres sont disponibles, se référer à la documentation.

```
# Prediction
# Temps d'exécution: 5.736653 secs

pred_alsr <- model_alsr$predict(test, k=20, not_recommend=NULL)
```

```r
# Calcul de la précision
mean(ap_k(pred_alsr, actual = test))
```

```
## [1] 0.3143703
```

Finalement, nous obtenons la mesure de précision pour les 20 meilleures recommandations. Le paramètre *not_recommend* permet de spécifier les items à ne pas recommander.

Bibliographie

Aberger, Christopher R. s. d. « Recommender: An Analysis of Collaborative Filtering Techniques », 5.

Adomavicius, G., et A. Tuzhilin. 2005. « Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions ». *IEEE Transactions on Knowledge and Data Engineering* 17 (6): 734-49. https://doi.org/10.1109/TKDE.2005.99.

Bendakir, Narimel, et Esma Aïmeur. s. d. « Using Association Rules for Course Recommendation », 10.

Burke, Robin. 2002. « Hybrid Recommender Systems: Survey and Experiments ». *User Modeling and User-Adapted Interaction* 12 (4): 331-70. https://doi.org/10.1023/A:1021240730564.

Hahsler, Michael. 2015. « Recommenderlab: A Framework for Developing and Testing Recommendation Algorithms », février, 40.

Hu, Yifan, Yehuda Koren, et Chris Volinsky. 2008. « Collaborative Filtering for Implicit Feedback Datasets ». Dans *2008 Eighth IEEE International Conference on Data Mining*, 263-72. Pisa, Italy: IEEE. https://doi.org/10.1109/ICDM.2008.22.

Isinkaye, F. O., Y. O. Folajimi, et B. A. Ojokoh. 2015. « Recommendation Systems: Principles, Methods and Evaluation ». *Egyptian Informatics Journal* 16 (3): 261-73. https://doi.org/10.1016/j.eij.2015.06.005.

McAlone, Nathan. 2017. « The exec who replaced Netflix's 5-star rating system with 'thumbs up, thumbs down' explains why. Business Insider ». 5 avril 2017. https://www.businessinsider.com/why-netflix-replaced-its-5-star-rating-system-2017-4.

Mehta, Rachana, et Keyur Rana. 2017. « A review on matrix factorization techniques in recommender systems ». Dans *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, 269-74. https://doi.org/10.1109/CSCITA.2017.8066567.

Michael Ekstrand, et Joe Konstan. s. d. *Introduction to Recommender Systems*. Université du Minnesota. https://www.coursera.org/specializations/recommender-systems.

Pan, Rong, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, et Qiang Yang. 2008. « One-Class Collaborative Filtering ». Dans *2008 Eighth IEEE International Conference on Data Mining*, 502-11. Pisa, Italy: IEEE. https://doi.org/10.1109/ICDM.2008.

16.

Su, Xiaoyuan, et Taghi M. Khoshgoftaar. 2009. « A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence ». Review Article. 2009. https://doi.org/https://doi.org/10.1155/2009/421425.