Systèmes de Recommandation

Par :
Philippine RENAUDIN

Master 2 Mathématiques et Applications
Parcours Data Science

Sous la direction de M. Panloup Fabien

2020-2021

Table des matières

Introduction

Contexte

Les problèmes de recommandation sont de plus en plus étudiés car ils répondent à une demande en constante augmentation de consommer. Qu'il s'agisse de livres, de recettes, d'appareils connectés ou de films et séries, les utilisateurs (ou clients) espèrent être les mieux conseillés possible en fonction de leurs préférences, et ce de manière de plus en plus exigeante : la concurrence étant nombreuse dans le monde du e-commerce, il est très facile pour un utilisateur de trouver un autre prestataire si le service ne lui semble pas assez personnalisé. C'est pourquoi les entreprises digitalisées comme Netflix, Amazon, Fnac ou encore Youtube se penchent de près sur ces problèmes. A l'heure actuelle, il existe plusieurs solutions efficaces, regroupées en 2 grandes catégories : les solutions dites de "content-filtering" (filtrage de contenu) et de "collaborative-filtering" (filtrage collaboratif). Nous détaillons ces 2 familles dans la suite, en nous attardant sur le filtrage collaboratif.

Le but commun de ces méthodes est de trouver une fonction qui, à un couple (utilisateur,item), associe une note, correspondant à l'intérêt que pourrait porter l'utilisateur à l'item.
Mathématiquement, on cherche $r : U \times I \mapsto \mathbb{R}$, où U représente l'ensemble des utilisateurs et I l'ensemble des items.

Éthique

Les systèmes de recommandation posent souvent des problèmes éthiques car la collecte d'informations personnelles peut-être faite à l'insu de l'utilisateur en fonction de la réglementation en vigueur. De plus, la création de profils utilisateurs peut mener à des stratégies d'influences (les profils peuvent être revendus ou volés et être utilisés pour influencer une élection, par exemple). Un système de recommandation doit donc respecter la législation et être transparent sur la façon dont sont effectuées les recommandations pour éviter son rejet par les utilisateurs.

Notations

- On notera X la matrice utilisateur-item, matrice regroupant le plus souvent les notes données par les utilisateurs aux items. On fera référence à cette matrice lorsque l'on parlera de *feedback explicite*. Pour faire référence aux notes présentes dans cette matrice, nous utiliserons le terme "valeurs observées" ou encore "observations". Cet ensemble de notes connues sera noté Ω et sera de cardinal m, sauf mention contraire.
- On appellera "matrice utilisateur" la matrice regroupant les informations disponibles sur les utilisateurs. Cette matrice pourra dans certains cas contenir des informations supplémentaires sur l'utilisateur, que celui-ci n'aura pas explicitement donné. Dans ce cas, on fera référence à cette matrice lorsque l'on parlera de *feedback implicite*.
- On appellera "matrice item" la matrice regroupant les informations disponibles sur les items.

I Content-Filtering

Avant de rentrer dans les détails du filtrage de contenu, nous faisons un petit point sur les techniques les plus simples de recommandation. Ces techniques ne sont bien évidemment plus du tout utilisées en pratique mais sont un bon point de départ pour comprendre comment fonctionne un système de recommandation.

I.1 Simple-recommenders

Les "simple-recommenders" (systèmes de recommandation simple) offrent des recommandations générales à tous les utilisateurs, en se basant sur la popularité d'un item. L'idée derrière cela est qu'un item populaire a plus de chance de plaire à un grand nombre de personnes qu'un item peu populaire.
Il est donc nécessaire ici de définir une métrique permettant de quantifier la popularité d'un item. Il n'existe pas de métrique prédéfinie car la construction de celle-ci dépend des données à disposition dans la matrice item. Néanmoins, il est courant d'avoir accès à la note globale d'un item ainsi qu'au nombre de votes. On rencontre alors fréquemment comme métrique une somme pondérée des notes globales. Un exemple d'application (s'inspirant de [9]) se trouve à l'adresse suivante, partie "Simple Recommenders" : [`https://github.com/Phi-gif/Projet_annuel/blob/main/Simple_and_content_recom_syst.ipynb`]

I.2 Content-filtering

Le filtrage de contenu est une méthode de recommandation un peu plus personnalisée que les simple-recommenders. En effet, l'idée ici est de recommander des items similaires aux items déjà consommés par l'utilisateur. Le principe est le suivant :

- Créer un profil pour chaque item en se basant sur les seules données relatives à l'item (matrice item)
- Calculer les similarités des items
- Déterminer les items les plus similaires entre eux et recommander en fonction de l'historique de l'utilisateur

La partie la plus importante de cet algorithme est la création du profil des items. Bien que l'on dispose souvent de beaucoup de métadonnées, il n'est pas toujours facile de trouver des attributs pertinents pour la création du profil. Si les items considérés sont des films et que l'on dispose dans la base de données du résumé de chaque film, alors il est possible de créer une matrice de fréquence de mots-clés où les lignes représentent les films et les colonnes les mots-clés. La valeur en ligne i et colonne j représente le nombre de fois où le mot-clé j apparait dans le résumé du film i, divisé par le nombre d'occurrences du mot-clé j dans tous les résumés. On peut ainsi déterminer les films les plus similaires en fonction

des fréquences de ces mots-clés. Il existe bien évidemment différentes mesures de similarité possibles. On peut évoquer par exemple le coefficient de corrélation de Pearson, le coefficient de corrélation du rang de Spearman ou encore la mesure cosine. Pour effectuer la vectorisation des items, il est très courant d'utiliser l'algorithme tf-idf : Term Frequency x Inverse Document Frequency (voir [10] pour plus de détails). Un exemple d'application se trouve à la même adresse que précédemment, partie "Content-based Recommender".

Un des avantages du content-filtering est que même si de nouveaux items sont ajoutés à la base de données, il est possible de les recommander car la recommandation ne se base que sur la similarité contextuelle de ceux-ci.

Comme énoncé plus haut, le filtrage de contenu et les simple-recommenders sont en pratique très peu ou plus utilisés. En effet, les simple-recommenders n'offrent pas de recommandations personnalisées, ce qui est un gros frein à leur utilisation aujourd'hui. Les méthodes de content-filtering quant à elles personnalisent un peu plus les recommandations mais n'ont pas vraiment la capacité de recommander des nouveautés, ce qui peut sur le long terme leur porter préjudice. Mais le plus gros défaut de ces méthodes est le temps de calcul. Celui-ci devient désastreux dès que le nombre d'items augmente, notamment à cause du calcul de similarité. C'est pourquoi d'autres méthodes ont été développées, dans le but d'améliorer la personnalisation des recommandations et surtout le passage à l'échelle.

II Filtrage Collaboratif

A l'inverse des méthodes présentées précédemment, le filtrage collaboratif se focalise sur la matrice utilisateur-item plutôt que sur la matrice item. Le filtrage collaboratif se base donc sur les interactions passées entre un utilisateur et plusieurs items, ce que l'on appelle plus communément le *feedback explicite*, pour sélectionner et recommander de manière automatique de nouveaux items à un utilisateur.
Bien que se focalisant sur la matrice utilisateur-item, certaines méthodes ont pour avantage de prendre en compte des données extérieures sur les utilisateurs comme les recherches internet, les données des cookies, etc... dans les algorithmes, notamment pour résoudre le problème du *coldstart* sur lequel nous revenons dans la suite. Ces données extérieures représentent le *feedback implicite* et posent quelque fois des problèmes d'éthique.

Il existe plusieurs familles de méthodes en filtrage collaboratif. La première famille a longtemps été l'état de l'art des systèmes de recommandation tant par sa facilité d'implémentation que par ses performances prédictives. Il s'agit des méthodes de voisinage. Cependant, de nouvelles méthodes comme celles basées sur des facteurs latents ont vu le jour et ont tendance à les surpasser, notamment en terme de temps de calcul et donc de passage à l'échelle. C'est à ces méthodes que nous nous sommes le plus intéressé dans ce projet. Néanmoins, il est intéressant de rappeler comment fonctionnent les méthodes de voisinage.

II.1 Méthodes de voisinage

Les méthodes de voisinage sont basées sur des scores de similarité entre utilisateurs ou entre items. On parle alors d'approche *User-User* ou d'approche *Item-Item*.

Approche User-User

Dans cette approche, on fait l'hypothèse que les utilisateurs ayant un profil similaire ont des goûts similaires. On va alors chercher à construire des clusters d'individus pour recommander à un utilisateur du cluster des items en se basant sur l'avis des autres utilisateurs du cluster. Pour cela, il faut construire un score de similarité pour apparier les utilisateurs similaires puis construire une fonction de prédiction pour déterminer qu'elle serait la note donnée par un utilisateur aux différents items présents dans la base de données. Cette fonction de prédiction est en général une moyenne pondérée des notes données par les voisins les plus proches à l'item considéré.

Comme dans le cas du content-filtering, les mesures de similarités utilisées pour déterminer des utilisateurs proches sont en général le coefficient de corrélation de Pearson, le coefficient du rang de Spearman ou la mesure de similarité cosine.
La fonction de prédiction s'écrit alors de la manière suivante : $\widehat{r_{u,i}} = \bar{r_u} + \frac{\sum_{u' \in S_u} s(u,u')(r_{u',i} - \bar{r_{u'}})}{\sum_{u' \in S_u} |s(u,u')|}$
où :
$\widehat{r_{u,i}}$ est la prédiction de la note que donnerait l'utilisateur u à l'item i,
$\bar{r_u}$ est la moyenne des notes données par l'utilisateur u,
$s(u, u')$ est un score de proximité entre l'utilisateur u et l'utilisateur u',
$r_{u',i}$ est la note donnée par l'utilisateur u' à l'item i,
S_u est l'ensemble des utilisateurs similaires à l'utilisateur u.
Pour définir S_u il faut donc choisir un seuil de similarité à partir duquel 2 individus ne sont plus similaires ou alors un nombre maximal d'individus par cluster.

Remarque : Le fait de soustraire $\bar{r_{u'}}$ dans la formule permet de compenser les différences de notation des utilisateurs (certains sont plus difficiles que d'autres dans la notation, par exemple) et donc de rectifier les éventuels biais de notation.

Bien que cette approche soit assez intuitive et facile à implémenter algorithmiquement, elle souffre d'instabilité lors du passage à l'échelle. En effet, la recherche des voisins d'un utilisateur u et le calcul des notes données aux items i est en O(|U||I|), où |U| est le nombre d'utilisateurs dans la base de données et |I| le nombre d'items. On voit donc que dès que le nombre d'utilisateur et/ou d'items devient trop important, le temps de calcul explose. C'est pourquoi l'approche *Item-Item* a été développée.

Approche Item-Item

Dans cette approche, on fait l'hypothèse que les utilisateurs préfèrent des produits similaires à ceux déjà appréciés. Cela rejoint le content-filtering mais ici la similarité des items est déduite des préférences des utilisateurs et non à partir de métadonnées. En effet, deux items seront considérés similaires s'ils sont bien notés (resp. mal notés) par les mêmes utilisateurs. La fonction de prédiction dans cette approche ressemble donc fortement à la précédente.
On obtient donc : $\widehat{r_{u,i}} = \frac{\sum_{j \in S} s(i,j) r_{u,j}}{\sum_{j \in S} |s(i,j)|}$, où :

$s(i,j)$ est un score de similarité entre l'item i et l'item j,
S est l'ensemble des items similaires à l'item i,
$r_{u,j}$ est la note donnée à l'item j par l'utilisateur u.
Si l'utilisateur n'a pas noté l'item j, alors $r_{u,j}$ vaut 0.

Remarque : La mesure de similarité la plus fréquente dans l'approche *Item-Item* est la similarité cosine.

L'avantage majeur de cette approche par rapport à l'approche précédente est quelle supporte un peu mieux le passage à l'échelle. En revanche, dès que la dimension de l'ensemble des items I augmente, le calcul de la matrice de similarité item-item devient très important, ce qui est impraticable quand on sait qu'un utilisateur accepte d'attendre en moyenne 3 secondes maximum pour avoir une recommandation. Un exemple d'application de ces deux méthodes (s'inspirant de [3]) est présent à l'adresse suivante : [`https://github.com/Phi-gif/Projet_annuel/blob/main/collab_filt_neighbors.ipynb`]
Pour plus de détails sur les méthodes de voisinage, le lecteur intéressé pourra consulter [13].

Un des principaux défauts de ces méthode avec le passage à l'échelle est le problème du *coldstart* ("départ à froid"). Le *coldstart* est l'incapacité de faire de bonnes recommandations à un nouvel utilisateur pour lequel on dispose de très peu d'informations (très peu ou même pas du tout d'item noté). On rencontre le même problème lors de l'introduction d'un nouvel item dans la base de donnée : celui-ci étant très peu voir pas du tout noté, il sera difficile d'obtenir une note élevée dans les deux approches.
D'autres défauts peuvent être mentionnés comme la personnalisation médiocre de ces systèmes de recommandations. En effet, le système doit être capable de recommander des nouveautés, or dans l'approche *Item-Item* on a tendance à recommander des items similaires à ceux déjà consommés. La prédiction de la note en elle-même n'est pas si importante, c'est surtout le fait qu'il n'y ait pas de faux positifs qui compte. Or ici, les mesures de similarité entre utilisateurs n'ont pas vraiment la capacité de "comprendre" leurs préférences. On risque alors de perdre la confiance de l'utilisateur dans le système de recommandation, ce qui peut sur le long terme le pousser à perdre son intérêt pour l'entreprise (ou le service) considérée.
Pour finir, on peut noter que la parcimonie de la matrice utilisateur-item rend les calculs de recommandation longs "pour pas grand chose". En effet, dans la vraie vie peu d'items

sont notés. Il est donc un peu "inutile", notamment dans l'approche *User-User*, de calculer la note que donnerait l'utilisateur u à tous les items i.
Ces méthodes par voisinage ne profitent donc pas de cette propriété de parcimonie de la matrice utilisateur-item.

II.2 Méthodes à facteurs latents

Le but des méthodes à facteurs latents est, comme précédemment, de prédire les valeurs manquantes de la matrice utilisateur-item X. Pour cela, on va ici identifier des variables cachées (latentes) par factorisation de matrice afin de déterminer un profil ou de déduire un comportement de groupe.
Ces méthodes sont très utilisées en pratique car elles tirent profit de la parcimonie de la matrice utilisateur-item et supportent plutôt bien le passage à l'échelle, tout en ayant des performances prédictives importantes. De plus, elles offrent la possibilité de modéliser des situations de la vie réelle en prenant en charge le *feedback implicite*, les effets temporels ou encore les biais de notations.
Les méthodes à facteurs latents reposent donc sur la factorisation de matrices non négatives (la matrice utilisateur-item ne comporte que des valeurs positives ou nulles) qui est un domaine mathématique très largement étudié et qui offre notamment beaucoup d'algorithmes. Pour ne pas alourdir ce rapport, nous ne présenterons que les algorithmes les plus utilisés dans le cas précis des systèmes de recommandation.

Beaucoup de résultats connus sur les systèmes de recommandation découlent des problèmes de complétion de matrice. En effet, la prédiction des valeurs manquantes de X utilise les valeurs connues de X (les valeurs observées). On peut donc tout à fait envisager, pour prédire ces valeurs manquantes, de reconstruire entièrement la matrice X, ce qui a pour avantage d'exprimer le problème comme un problème de complétion de matrice sous contrainte.
Nous allons donc dans un premier temps rappeler quelques résultats sur la complétion de matrice, puis nous intéresser au domaine de factorisation de matrices non négatives pour aboutir enfin sur les méthodes de recommandation par facteurs latents.

Complétion de matrice

Les problèmes de complétion de matrices se décomposent en deux catégories : la complétion de matrices de faible rang et la complétion de matrices de haut rang. Les préférences des utilisateurs pouvant souvent être décrites par un petit nombre de facteurs (latents), la matrice X est très souvent de faible rang. *L'humain est certes un être complexe mais quand il s'agit de ses préférences, il sait ne pas se montrer trop compliqué.*
C'est pourquoi nous nous focalisons sur la complétion de matrices de faible rang dans la suite.
Le but des problèmes de complétion de matrices de faible rang est donc de trouver une matrice de rang minimal r (connu ou non) qui "match" les valeurs observées.

Comme énoncé précédemment, les systèmes de recommandation sont une application de la complétion de matrice. D'autres applications connues sont la reconstruction d'images (computer vision) et l'apprentissage multi-classe. Les problèmes de complétion sont souvent NP-difficiles mais il est possible sous certaines conditions de trouver des algorithmes performants reconstruisant de manière exacte la matrice avec grande probabilité.

Écriture du problème

Le problème de complétion de matrice de faible rang est donc de trouver une matrice M de rang minimal qui reconstruit de manière exacte les valeurs observées de X avec grande probabilité. Le problème mathématique est donc le suivant :

$$\begin{aligned} \min \quad & \text{rank(M)} \\ \text{s.t.} \quad & M_{ij} = X_{ij}, (i,j) \in \Omega \end{aligned}$$

Cependant, ce problème d'optimisation est NP-difficile et tous les algorithmes connus donnant une solution exacte à ce problème requiert un temps de computation 2 fois exponentiel à la dimension n de matrice (dans le cas d'une matrice carré), tant en théorie qu'en pratique. Candès et Recht [1] ont proposé comme alternative de considérer non pas le rang de M mais sa norme nucléaire. La norme nucléaire étant la somme des valeurs singulières d'une matrice. Or, si M est de rang r, cela signifie qu'elle possède r valeurs singulières non nulles et donc considérer non pas le nombre mais la somme des amplitudes de ces valeurs singulières permet de rendre le problème de minimisation convexe et facile a résoudre par programmation semi-définie. En un sens, la norme nucléaire est au rang ce que la norme l_1 est à la norme l_0 dans le cas des régressions pénalisées, par exemple.

Le problème de complétion se réécrit donc comme suit :

$$\begin{aligned} \min \quad & \|M\|_* \\ \text{s.t.} \quad & M_{ij} = X_{ij}, (i,j) \in \Omega \end{aligned}$$

Néanmoins, même si ce problème de minimisation est beaucoup plus simple à résoudre, il n'est pas garanti que sa résolution donne une solution unique avec grande probabilité. En effet, Candès et Recht ont montré (dans le cas des matrices carrés), que si le nombre de valeurs observées de X est inférieur à un certain seuil alors il est impossible de reconstruire la matrice X de manière unique avec grande probabilité. On voit donc qu'il est nécessaire d'avoir un nombre minimum de valeurs observées. Ce nombre minimal découle d'un résultat de répartition des valeurs dans la matrice. Nous expliquons cela dans la section suivante.

Une condition nécessaire

Comme énoncé précédemment, la répartition des valeurs observées de X joue un rôle capital dans la résolution des problèmes de complétion. Tout d'abord, nous faisons ici l'hypothèse que les valeurs observées sont réparties de manière aléatoire et uniforme. Il est très courant de faire cette hypothèse en pratique car le fait qu'un utilisateur u note un item i ne dépend pas des autres utilisateurs ni items. Le lecteur intéressé pourra se référer à [11] et [12]. En résumé, on fait l'hypothèse que les valeurs observées de X sont "tirées" de manière indépendante et avec remise.

Candès et Recht ont montré que sous l'hypothèse précédente, il est nécessaire d'observer au moins une valeur sur chaque ligne et colonne de X pour pouvoir résoudre le problème de complétion de manière unique. En effet, considérons la décomposition en valeurs singulières de X : $X = U\Sigma V^T$.
Alors si la i^{eme} ligne de X ne contient aucune valeur, on peut facilement voir que le i^{eme} vecteur singulier de V^T, v_i, peut être modifié de manière arbitraire sans pour autant compromettre la reconstruction des valeurs observées.
Considérons par exemple que X est de rang 1 alors : $X = xy^T, x, y \in \mathbb{R}^n$ et $X_{ij} = x_i y_j$.
Si aucune valeur n'est observée sur la première ligne de X alors il est impossible de déterminer x_1, aucune information concernant x_1 n'est disponible. Il en va de même pour n'importe quelle ligne ou colonne ne contenant aucune valeur.
Cette condition donne donc un moyen de trouver le nombre minimal de valeurs observées pour que les problèmes de complétion aient une solution unique avec grande probabilité. En effet, il est possible de faire le lien avec le problème du "Collectionneur de Coupon". Pour rappel, ce problème cherche à trouver le nombre d'achats qu'il faut effectuer avant d'obtenir une collection complète de n coupons, en supposant que l'on "tire" les coupons avec remise et que l'on en a qu'un seul par achat (ex : jouets dans les paquets de céréales).

Le problème du collectionneur de coupons

Considérons la variable aléatoire T_n représentant le nombre de coupons achetés avant d'obtenir les n coupons de la collection. On peut considérer que T_n est un temps : elle représente le temps qu'il faut pour obtenir n coupons différents. Soit t_i le temps supplémentaire pour obtenir le i^{eme} coupon sachant que l'on en a i-1 (distincts). Alors, $T_n = \sum_{i=1}^{n} t_i$. Or, la probabilité de trouver un nouveau coupon lorsque l'on en détient i-1 est de $\frac{n-i+1}{n}$ donc (t_i) suit une loi géométrique de paramètre $\frac{n-i+1}{n}$. On peut donc facilement calculer l'espérance de T_n :

$$\begin{aligned}
\mathbb{E}[T_n] &= \sum_{i=1}^{n} \mathbb{E}[t_i] = \mathbb{E}[t_1] + \mathbb{E}[t_2] + \cdots + \mathbb{E}[t_n] \\
&= \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} \\
&= n.(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} + 1) \\
&= n.H_n
\end{aligned}$$

où H_n est le n^{ieme} nombre harmonique.
En utilisant le développement asymptotique de H_n, on obtient :
$\mathbb{E}[T_n] = n.(\log(n) + \gamma + \frac{1}{2n} + o(\frac{1}{n}))$ et donc plus simplement que $\mathbb{E}[T_n] = O(n\log(n))$.

Remarque : Le γ de la première expression est la constante d'Euler-Mascheroni.

On voit donc que le nombre d'achats qu'il faut effectuer pour obtenir une collection complète de n coupons est en $O(n\log(n))$.
Dans le cas des problèmes de complétion de matrice, on a vu qu'il faut avoir au moins 1 observation par ligne et colonne de X. Notons ici m_1, m_2 le nombre de lignes et colonnes de X. Posons $n = max(m_1, m_2)$. Alors le nombre minimal de valeurs qu'il faut observer pour pouvoir recouvrir X de manière unique avec grande probabilité s'apparente au problème du collectionneur de coupons mentionné ci-dessus. Il faut donc observer un nombre de valeurs en $O(n\log(n))$. Cependant, si l'on s'arrête ici, il n'est pas garanti d'obtenir une matrice reconstruite M de faible rang. En effet, dans le meilleur des cas, M est de rang p ($p = min(m_1, m_2)$), ce qui n'est pas cohérent avec l'hypothèse de faible rang. C'est pourquoi en pratique, le nombre de valeur observées est en $O(nr\log(n))$, où r est le rang de X.

Factorisation de Matrices Non Négatives (NMF)

Comme il a été introduit en début de section, les méthodes à facteurs latents reposent sur la factorisation de matrice et plus particulièrement la factorisation de matrices non négatives. Nous présentons brièvement le principe de la NMF dans cette sous-section.

Nous considérons toujours la matrice creuse X (utilisateur-item). Le principe de la NMF est de trouver deux matrices positives W et H , $W \in \mathbb{R}^{m_1 \times r}$, $H \in \mathbb{R}^{m_2 \times r}$ telles que $\widehat{X} = WH^T$. Pour trouver ces deux matrices, on résout un problème de minimisation de la forme :

$$\begin{aligned} &\underset{W,H}{\operatorname{argmin}}\ L(X; W, H) + \lambda P(W, H) \\ &\text{subject to } W, H \geqslant 0 \\ &\text{suject to } rang(W) = rang(H) = r \end{aligned}$$

où :
L est une fonction d'attache aux données mesurant la qualité de prédiction de X par WH^T,
λ est une constante de pénalisation,
P est une fonction de régularisation/pénalisation.

Les fonctions d'attaches aux données les plus utilisées sont la norme de Frobenius, la divergence de Kullback-Leibler ou encore la divergence d'Itakura-Saito. La fonction de pénalisation est ajoutée pour éviter le sur-apprentissage sur les données observées et pour introduire de la parcimonie. Il s'agit en général d'une norme. La constante de pénalisation λ est trouvée le plus souvent par validation croisée.

Les algorithmes les plus utilisés pour résoudre ce problème de minimisation sont les algorithmes dits "multiplicatifs" (pour plus de détails voir [4]), les algorithmes de descente de gradient et les algorithmes de moindres carrés alternés.

Méthodes à facteurs latents

Les méthodes à facteurs latents sont donc des cas particuliers de factorisation de matrice non négatives car on cherche ici un petit nombre de facteurs expliquant les préférences des utilisateurs. On reprend donc des résultats vus dans les deux sous-sections précédentes.
Dans cette section, on note $X_{i,j}$ une observation connue de X et $Y_{i,j}$ une observation bruitée de $\bar{X}_{i,j}$ où $\bar{X}$ est la matrice X complétée inconnue. Le bruit peut être additif auquel cas $Y_{i,j} = \bar{X}_{i,j} + \epsilon_{i,j}$ avec $\epsilon_{i,j}$ centré, appartenir à la famille exponentielle ou être un bruit généralisé. Le problème de minimisation dans le cas du bruit additif est donc : $\widehat{X} = WH^T$, où W et H sont trouvées par résolution de :

$$\begin{aligned} &\underset{W,H}{\operatorname{argmin}}\ L(X; W, H) + \lambda(\|W\|_* + \|H\|_*) \\ &\text{subject to } W, H \geqslant 0 \\ &\text{suject to } min(rang(W), rang(H)) \leqslant r \end{aligned} \tag{1}$$

Remarque : Si le bruit est gaussien, la fonction d'attache aux données est souvent la norme de Frobenius. En revanche, si le bruit n'est pas gaussien (distribution de Poisson par exemple), la fonction d'attache aux données sera plutôt la divergence de Kullback-Leibler ou d'Itakura-Saito. De plus, il est courant de rencontrer dans certaines librairies Python ou R des fonctions n'incluant pas de pénalisation dans le problème de minimisation précédent. Nous ne nous attardons pas sur ces formulations mais le lecteur intéressé pourra consulter [7] pour plus de détails et résultats.

Dans le cas du bruit généralisé (qui englobe le cas du bruit additif), on suppose que $Y_{i,j}|X_{i,j} \sim P_{\bar{X}_{i,j}}$. Une application importante de cela est le cas où $Y_{i,j} \in \{0, 1\}$. Dans ce cas, on trouve que $P(Y_{i,j}|X_{i,j}) = f(\bar{X}_{i,j})$ où f est une fonction de lien (typiquement un logit ou un probit). Alors la fonction d'attache aux données n'est plus forcément la perte quadratique mais une autre fonction plus adaptée comme la log-vraisemblance négative de la distribution. Pour plus de détails, on pourra consulter Davenport et al. (2014) et [7].

Algorithmes

Nous nous intéressons ici aux deux principales familles d'algorithmes couramment utilisées en système de recommandation. La première famille est celle des descentes de gradient et la deuxième celle des descentes de coordonnées. L'algorithme le plus utilisé en descente de gradient est la descente de gradient stochastique (Stochastic Gradient Descente) et celui le plus utilisé en descente de coordonnées est l'algorithme des moindres carrés alternés (Alternated Least Square). Nous détaillons ci-dessous ces deux algorithmes.

Descente de gradient stochastique (SGD)

Les algorithmes de descente de gradient utilise la règle de mise à jour suivante :
soit θ_0 une valeur initiale du vecteur θ, alors $\forall t \in \{1, \cdots, p\}, \theta_{t+1} = \theta_t - \gamma_t \nabla F(\theta_t)$.
Or, lorsque le nombre de valeurs observées m est grand, le calcul exact du gradient peut s'avérer coûteux et il est souvent avantageux d'approximer ce gradient sur un sous échantillon de termes choisis aléatoirement. L'algorithme de descente de gradient stochastique choisi un sous échantillon de taille 1. On détaille ci-dessous cet algorithme dans le cas du problème (1) avec la norme de Frobenius comme fonction d'attache aux données.

Algorithm 1 Descente de gradient stochastique

Init : $X_0 = W_0 H_0^T \in \mathbb{R}^{m_1 \times m_2}$, maxiter $= T$, step $= gamma$
for $t = 1, \cdots, T$ **do**
  Choose $(i, j) \in \Omega$
  for $k \in [m_1]$ **do**
    $(W_t)_{k,\cdot} = (W_{t-1})_{k,\cdot} - \gamma((X_{i,j} - (WH_{t-1}^T)_{i,j})(H_{t-1})_{\cdot,j} + \lambda(W_{t-1})_{k,\cdot})$
  end for
  for $l \in [m_2]$ **do**
    $(H_t)_{l,\cdot} = (H_{t-1})_{l,\cdot} - \gamma((X_{i,j} - (WH_{t-1}^T)_{i,j})(W_{t-1})_{i,\cdot} + \lambda(H_{t-1})_{l,\cdot})$
  end for
end for
return $\widehat{X} = W_T H_T^T$

L'initialisation de W_0 et H_0 est importante pour que l'algorithme converge. Plusieurs méthodes d'initialisation sont implémentées dans les différents packages Python et R. Nous en énonçons rapidement quelques unes :

- "random" : génère des matrices non-négatives aléatoirement
- "nndsvd" : génère des matrices non-négatives par double décomposition en valeurs singulières. Bien adaptée quand souhaite de la parcimonie
- "nndsvda" : NNDSVD où les valeurs nulles sont remplacées par la moyenne de X. Bien adaptée quand on ne souhaite pas de parcimonie
- "nndsvdar" : NNDSVD où les valeurs nulles sont remplacées par de petites valeurs aléatoires . Technique généralement plus rapide mais moins précise que NNDSVDA quand on ne souhaite pas de parcimonie

Le pas γ est choisi constant car la taille des données est souvent très élevée et calculer un nouveau pas à chaque itération ralentirait grandement l'algorithme pour peu de gain de précision.
$(W_t)_{k,\cdot}$ correspond à la ligne k de W_t et $(H_{t-1})_{\cdot,j}$ correspond à la colonne j de H_{t-1}.
On a $\lambda(W_{t-1})_{k,\cdot}$ comme dérivée de la norme nucléaire car cette dernière vérifie :

$$\begin{aligned} &\underset{W,H}{\operatorname{argmin}} \; L(X; W, H) + \frac{\lambda}{2}(\|W\|_2^2 + \|H\|_2^2) \\ &\text{subject to } X = WH^T \end{aligned}$$

Remarque : Nous avons présenté ici le cas particulier de la norme de Frobenius mais cet algorithme est aussi implémenté pour la divergence de Kullback-Leibler. En revanche, il n'existerait à ce jour pas d'implémentation pour la divergence d'Itakura-Saito. De plus, il existe une version parallélisable de cet algorithme (présentée par Recht & Ré en 2013) qui est actuellement l'état de l'art pour les grandes bases de données (Algorithme *Jellyfish*).

Alternated Least Square (ALS)

On détaille ci-dessous cet algorithme dans le cas du problème (1), pour n'importe quelle fonction d'attache aux données.

Algorithm 2 Alternated Least Square

Init : $W_0 \in \mathbb{R}^{m_1 \times r}$, maxiter $= T$
for $t = 1, \cdots, T$ **do**
  Résoudre (1) en H
  Mettre à 0 les termes négatifs de H
  Résoudre (1) en W
  Mettre à 0 les termes négatifs de H
end for
return $\widehat{X} = W_T H_T^T$

Une version améliorée de cet algorithme existe sous le nom FastHALS, nous ne la détaillons pas ici mais le lecteur intéressé pourra consulter [2] ou encore [6]. Il est à noter que la fonction *non_negative_factorization* du package python sklearn.decomposition utilise le FastHALS pour trouver W et H. Ce package utilise de plus les mêmes méthodes d'initialisation que présentées ci-dessus.

Un des avantages de cette famille d'algorithmes est qu'ils sont en général plus performants que les descentes de gradient stochastique quand la matrice sur laquelle on travaille est dense et non plus creuse. Cela arrive lorsque l'on prend en compte le *feedback implicite* pour résoudre les problèmes de *coldstart* par exemple. Nous revenons sur cet aspect dans la suite.
En revanche, un désavantage notable est qu'en général le rang r de X est inconnu, il faut donc au préalable déterminer un rang optimal, par validation croisée par exemple.

Biais, dynamiques temporelles et le problème du coldstart

Le but des méthodes à facteurs latents est de trouver un certain nombre de facteurs implicites expliquant les préférences des utilisateurs. Cependant, chaque utilisateur a une façon différente de noter un item. Certains ont tendance à attribuer très souvent la note maximale ou à l'inverse très souvent des notes basses comparé à la moyenne. On retrouve également ce genre de biais du côté des items. De plus, les habitudes de notations changent avec le temps. En effet, on a tendance à devenir de plus en plus critique avec le temps, une des raisons étant que l'on a consommé un nombre important d'items et que nous avons donc une bonne base de comparaison. C'est pourquoi la prise en compte dans le problème de minimisation (1) de ces biais est très importante pour ne pas déduire de

fausses préférences des utilisateurs à l'égard de certains items.

Posons d'abord quelques notations. Dans un soucis de clarté, nous considérons comme fonction d'attache aux données la norme de Frobenius et utilisons la notation norme 2 de la norme nucléaire. On note i un item et $q_i \in \mathbb{R}^r$ le vecteur des facteurs latents associé à l'item i. On note u un utilisateur et $p_u \in \mathbb{R}^r$ le vecteur des facteurs latents associé à l'utilisateur u. Les coefficients de q_i mesurent dans quelle proportion (positive et négative) l'item i possède les caractéristiques des facteurs latents et les coefficients de p_u mesurent l'appétence de l'utilisateur u pour les items ayant des parts importantes (positives ou négatives) dans la définition des facteurs latents.
Ainsi, le produit $q_i^T p_u$ capture l'appétence globale de l'utilisateur u pour les caractéristiques de l'item i. Cette valeur approxime la note que l'utilisateur u pourrait donner à l'item i, notée r_{ui} d'où : $\widehat{r_{ui}} = q_i^T p_u$.

Le problème (1) se réécrit donc de la manière suivante :

$$\underset{p,q}{\operatorname{argmin}} \sum_{(u,i)\in\Omega} (r_{ui} - q_i^T p_u)^2 + \alpha(\|p_u\|_2^2 + \|q_i\|_2^2) \tag{2}$$

Biais

Considérons maintenant les biais mentionnés précédemment. On note : $b_{ui} = \mu + b_u + b_i$, où :
μ est la moyenne des notes données sur l'ensemble des items par l'ensemble des utilisateurs,
b_u est le biais introduit par l'utilisateur. Il s'agit de la différence entre μ et l'ensemble des notes données par l'utilisateur u,
b_i est le biais introduit par l'item. Il s'agit de la différence entre μ et l'ensemble des notes données à l'item i.

On obtient donc une nouvelle écriture pour $\widehat{r_{ui}}$ et pour le problème (2) :

$$\widehat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u,$$

$$\underset{p,q}{\operatorname{argmin}} \sum_{(u,i)\in\Omega} (r_{ui} - \widehat{r_{ui}})^2 + \alpha(\|p_u\|_2^2 + \|q_i\|_2^2 + b_u^2 + b_i^2) \tag{3}$$

Remarque : Le package Python surprise.prediction_algorithms permet de prendre en compte ces biais grâce aux fonctions *matrix_factorization.SVD* et *matrix_factorization.NMF*. Le problème de minimisation (3) est résolu par Descente de Gradient Stochastique.

Dynamiques temporelles

Comme il a été évoqué, les goûts des utilisateurs ainsi que leur façon de noter évoluent au fil du temps. De même, l'arrivée constante de nouveaux items modifie les schémas de notation. C'est pourquoi, en plus des biais précédents, il est important de considérer et modéliser ces dynamiques temporelles pour améliorer les performances des systèmes de recommandation.
On obtient donc une nouvelle expression de $\widehat{r_{ui}}$ et du problème (3) :

$$\widehat{r_{ui}}(t) = \mu + b_u(t) + b_i(t) + q_i^T p_u(t)$$

$$\underset{p,q}{\operatorname{argmin}} \sum_{(u,i)\in\Omega} (r_{ui} - \widehat{r_{ui}}(t))^2 + \alpha(\|p_u(t)\|_2^2 + \|q_i\|_2^2 + b_u(t)^2 + b_i(t)^2) \tag{4}$$

où :
$b_i(t)$ prend en compte l'évolution de la popularité des items au fil du temps,
$b_u(t)$ prend en compte l'évolution de la notation des utilisateurs au fil du temps,
$p_u(t)$ prend en compte les changements de préférence/goût des utilisateurs au fil du temps.

Coldstart

Pour terminer cette section, nous revenons sur le problème du *coldstart* et sur les moyens de le contrer grâce aux méthodes à facteurs latents. Pour rappel, le problème du *coldstart* est l'incapacité de faire des (bonnes) recommandations à un nouvel utilisateur pour lequel on dispose de très peu d'informations (très peu d'items notés). Comme déjà mentionné, une solution est de faire appel au *feedback implicite*, c'est à dire à des données extérieures sur l'utilisateur, que ces données aient été rentrées par l'utilisateur (matrice utilisateur : âge, sexe, revenu, etc...) ou récupérées dans les cookies ou par des méthodes discutables (historique internet, schémas de recherche, temps passé sur chaque site, nombre de clics, ect...). Concrètement, comment ça marche ?

Grâce à ce *feedback implicite*, il est alors possible de créer deux nouveaux ensembles de facteurs latents x_j et y_a tels que :

$\frac{1}{|N(u)|^{\frac{1}{2}}} \sum_{j\in N(u)} x_j$ représente les préférences de l'utilisateur u pour les items de $N(u)$, $N(u)$ étant un ensemble d'items pour lesquels l'utilisateur a exprimé une préférence implicite. (Par exemple, un item sur lequel il aurait fait des recherches internet mais ne l'aurait pas encore acheté),

$\sum_{a\in A(u)} y_a$ représente les caractéristiques d'un utilisateur u, $A(u)$ étant un ensemble d'attributs. En pratique, ce facteur est très peu utilisé, le premier apportant bien plus d'informations fiables.

En prenant en compte ces informations supplémentaires, $\widehat{r_{ui}}$ se réécrit :

$$\widehat{r_{ui}} = \mu + b_u + b_i + q_i^T(p_u + \frac{1}{|N(u)|^{\frac{1}{2}}} \sum_{j\in N(u)} x_j + \sum_{a\in A(u)} y_a)$$

Remarque : Le package Python surprise.prediction_algorithms permet de prendre en compte le *feedback implicite* grâce à la fonction *matrix_factorization.SVDpp*. En revanche, seul le premier type de feedback est implémenté. Un exemple d'application (inspiré de [8]) est disponible à l'adresse suivante : [`https://github.com/Phi-gif/Projet_annuel/blob/main/Collaborative_filtering_matrix_factorization.ipynb`]

La prise en compte des biais, des tendances temporelles et du feedback implicite a permis aux auteurs de [14] de remporter le concours organisé par Netflix en 2009. Ils montrent dans cet article l'amélioration des performances de leur système de recommandation au fur et à mesure de l'ajout de ces quantités. Le lecteur intéressé pourra se référer à [14].

Conclusion

Des simple-recommenders au filtrage collaboratif en passant par le filtrage de contenu, les entreprises digitalisées ont donc l'embarras du choix pour faire de bonnes recommandations à leurs clients. Néanmoins, au vu de la taille des données dont disposent les entreprises aujourd'hui, il est nécessaire d'utiliser des systèmes supportant cette échelle, c'est pourquoi le filtrage collaboratif est privilégié, notamment les méthodes à facteurs latents. En plus de mieux supporter la dimension, ces méthodes permettent des recommandations plus personnalisées et ont la capacité de recommander des nouveautés en détectant une ou plusieurs préférences implicites des utilisateurs. Pour finir, un avantage important est que ces méthodes peuvent directement prendre en compte le *feedback implicite* lors de l'arrivée de nouveaux utilisateurs ou items et ainsi contrer facilement le problème du *coldstart*. Il est alors possible de faire des recommandations basées sur les préférences des utilisateurs (déduites non pas des notations passées mais de données extérieures) et non sur la seule popularité d'un item ou la ressemblance entre utilisateurs.
Avec l'explosion du commerce en ligne depuis quelques années, renforcé encore depuis le début d'année 2020, les problèmes de recommandation sont donc devenus centraux dans les stratégies marketing des entreprises. En effet, la multiplication d'acteurs a accru la nécessité de se démarquer de ses concurrents. Les recherches sur les systèmes de recommandation et les mathématiques sous-jacentes se sont donc accélérées et il est à ce jour difficile d'avoir accès aux algorithmes des plus grandes entreprises digitalisées. Néanmoins, certains packages Python et R permettent d'appliquer des méthodes déjà assez poussées comme celles présentées dans ce rapport.
Pour finir, il est intéressant de noter que les systèmes de recommandation ne se limitent pas qu'au e-commerce, il existe d'autres domaines d'application comme les biens connus réseaux sociaux. La recommandation d'amis ou de pages d'intérêt sont aussi des enjeux pour ce genre d'entreprise. Le lecteur intéressé pourra se référer à [5] pour une application des graphes dans les systèmes de recommandation adaptés aux réseaux sociaux.

Remerciements

Je souhaite remercier la Faculté des Sciences de l'Université d'Angers pour avoir mis à disposition les outils nécessaires à la réalisation de ce projet et de ce rapport. Je souhaite également remercier Mr Panloup pour son encadrement tout au long de ce projet.

Je remercie enfin toutes les personnes qui m'ont entouré durant cette année scolaire plus que particulière.

Bibliographie

[1] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Applied and Computational Mathematics, Caltech, Pasadena, CA 91125*, 2008.

[2] Andrzej Cichocki and Ahn-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Trans. Fundamentals*, Unknown. `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.214.6398&rep=rep1&type=pdf`.

[3] Brendan Guillouet. Ia frameworks - introduction to recommendation system with collaborative filtering. Github. `https://github.com/wikistat/AI-Frameworks/blob/master/RecomendationSystem/surprise.ipynb`.

[4] IEEE International Workshop on machine learning for signal processing. *Mini-batch stochastic approaches for accelerated multiplicative updates in nonnegative matrix factorisation with beta-divergence*. Romain Serizel, Slim Essid, Gael Richard, Sept 2016. `https://hal.archives-ouvertes.fr/hal-01393964/document`.

[5] Stéphane Canu Julien Delporte, Alexandros Karatzoglou. Apprentissage et factorisation pour le recommandation. INSA ROUEN, LITIS.

[6] Mineichi Kudo Keigo Kimura, Yuzuru Tanaka. A fast hierarchical alternating least squares algorithm for othogonal nonnegative matrix factorization. In Dinh Phung and Hang Li, editors, *ACML*. JMLR : Workshop and Conference Proceedings 39 :129-141, 2014. `http://proceedings.mlr.press/v39/kimura14.pdf`.

[7] Jean Lafond. *Complétion de matrice : aspects statistiques et computationnels*. PhD thesis, Université Paris-Saclay, 2016.

[8] Susan Li. Building and testing recommender systems with surprise, step-by-step, Dec 2018. [`https://towardsdatascience.com/building-and-testing-recommender-systems-with-surprise-step-by-step-d4ba702ef80b`].

[9] Aditya Sharma. Beginner tutorial : Recommender systems in python. Internet. `https://www.datacamp.com/community/tutorials/recommender-systems-python`.

[10] Unknown. Apprentissage pour le filtrage collaboratif. Chapter 20 : "Vers de nouvelles tâches et de nouvelles questions", pages 686-694.

[11] Wikipedia. Factorisation de matrices pour les systèmes de recommandation. `https://fr.wikipedia.org/wiki/Factorisation_de_matrices_pour_les_syst%C3%A8mes_de_recommandation`.

[12] Wikipedia. Matrix completion. `https://en.wikipedia.org/wiki/Matrix_completion`.

[13] WikiStat. Collaborative filtering.

[14] Chris Volinsky Yehuda Koren, Robert Bell. Matrix factorization techniques for recommender systems. *IEEE Computer Society*, pages 42–49, 2009.