1 Système de recommandation

1.1 Introduction

En général il existe 2 grandes familles d'algorithme pour un système de recommandation : **la recommandation objet** (*content-based filtering*) et **la recommandation sociale** (*collaborative filtering*).

1.1.1 Recommandation objet

Il s'agit de recommander des objets (ou contenus) en se basant sur les qualités et propriétés intrinsèques de l'objet lui-même et en les corrélant avec les préférences et intérêts de l'utilisateur. Ce type de système va donc extraire un certain nombre de caractéristiques et attributs propres à un contenu, afin de pouvoir recommander à l'utilisateur des contenus additionnels possédant des propriétés similaires. Cette méthode crée un profil pour chaque objet ou contenu, c'est-à-dire un ensemble d'attributs/propriétés qui caractérisent l'objet.

Ce type de recommandation objet n'a pas besoin d'une large communauté d'utilisateurs pour pouvoir effectuer des recommandations. Une liste de recommandations peut être générée même s'il n'y a qu'un seul utilisateur.

Pourtant, ce qui est problématique, c'est l'acquisition de caractéristiques subjectives et qualitatives... Des propriétés d'un livre, comme par exemple le style, le design... peuvent difficilement être acquises automatiquement, et devront plutôt être introduites manuellement avec tout ce que ça implique, comme le coût, les éventuelles erreurs...

1.1.2 Recommandation social

Les systèmes de recommandation sociale (filtrage collaboratif - *FC*) aident un utilisateur à trouver l'information qui l'intéresse à partir des jugements d'autres utilisateurs. La plupart des algorithmes de FC existant considèrent le cas où les jugements sont exprimés sous forme de notes. La taille trop importante des bases de jugements disponibles a amené les concepteurs à se focaliser sur des méthodes d'apprentissage pour prédire les notes inconnues.

Les techniques de ce type sont classées en 2 classes :

— **Memory-based :** prédisent les notes inconnues pour un utilisateur donné en combinant les notes des utilisateurs ayant les goûts les plus similaires. Elles sont très simples à implémanter, mais le temps nécessaire à présenter une recommandation dépend du nombre d'utilisateurs de la base de données. De plus, ces approches souffrent d'un problème de la rareté (*spasity*) : le nombre de produits ou contenus est énorme sur certaines plates-formes, et même les utilisateurs les plus actifs auront noté ou valorisé qu'un très petit sous-ensembe

de toute la base de données. Donc, même l'article le plus populaire n'aura que très peu de bonnes notes. Dans une telle situation, deux utilisateurs auront peu d'articles valorisés en commun, ce qui rend plus difficile la tâche de corrélation.

— **Model-based :** les approches qui sont plus utilisées récemment. Elles souffrent beaucoup moins du problème de la rareté et font preuve d'une grande fiabilité. On peut citer parmi ces approches les méthodes de factorisation matricielle (ce que nous avons appliqué) qui cherchent à extraire des variables latentes à la fois pour les utilisateurs et pour les articles.

1.2 Factorisation matricielle et Optimisation alternée

1.2.1 Problème et notation

On suppose que l'on dispose d'une matrice de données $\mathbf{R}$ concernant m utilisateur et n questions. Pour notre site, Les données de $\mathbf{R}$ sont sous forme *implicite*, ce qui est l'opposition des données *explicites* qui sont basées sur un échelle de 1 à 5, par exemple ceux de Amazon, Ebay... Dans le cas des feedback implicites, les éléments de la matrice sont 1 pour une réponse positive (l'utilisateur l'a déja "upvote"), et 0 sinon : $R = (R_{ij})_{m\times n} \in \{0,1\}^{m\times n}$. Cela rend les choses plus difficiles à recommander parce que l'on est pas sûr si une réponse 0 montre que l'utilsateur n'aime pas la question, ou simplement que l'utilisateur ne l'a pas encore vu.

Notre tâche est de identifier les réponses positives potentielles dans $\mathbf{R}$. L'idée principale est d'apprendre un modèle latent d'utilisateurs $\mathbf{U} \in \Re^{m\times d}$ et d'articles $\mathbf{V} \in \Re^{n\times d}$ (avec d le nombre de latents) de sorte que la construction $\tilde{R}_{ij} = U_{i.}V_{j.}^T$ entre un utilisateur i et une question j estime le terme R_{ij} en minimisant la fonction objectif (le coût) suivante :

$$L(U,V) = \Sigma_{ij} W_{ij}(R_{ij} - U_{i.}V_{j.}^T)^2 + \lambda(\|U_{i.}\|_F^2 + \|V_{i.}\|_F^2) \qquad (1)$$

La terme $\lambda(\|U_{i.}\|_F^2 + \|V_{i.}\|_F^2)$ est nécessaire pour régulariser le modèle afin d'éviter le sur-apprentissage. λ est un paramètre d'équilibrage dont le réglage établi le niveau de consensus entre les deux critères. Sa valeur exacte dépend des données initiales et doit être déterminée par la *validation croisé*.

W_{ij} est appelé le "poids" (*w*eight) des réponses. Il réflecte la confiance que l'on accorde à la prédiction $\tilde{R}_{ij}$ ayant observé R_{ij}. Il a pour but de corriger le biais causé par les données manquantes. Pour une réponse positive($R_{ij} = 1$), on est assez sûr que c'est vrai, d'où un $W_{ij} = 1$. Pour la reste, la valeur de W_{ij} dépend de la nature de notre matrice initiale et de chacun des utilisateurs, sachant que $W_{ij} \in \{0,1\}$. Pour faciliter les calculs, nous avons supposé que tous les utilisateurs sont pareils, et donc pour tous les cas, nous avons assigné un $W_{ij} = 0.1$.

1.2.2 Optimisation alternée

Dans un problème de factorisation, lorsque le coût est différentiable, les méthodes les plus simples et les plus efficaces en terme de temps de calcul sont ceux de type descente de gradient. Cependant, on remarque que l'équation (1) contient $m \times n$ termes, et pour une base de données typique, ce produit peut être quelque milliards. Cette quantité immense nous empêche d'utiliser des techniques directes ci-dessus. Ainsi, nous avons appliqué un autre processus : **l'optimisation alternée** (*ou moindre carré alterné - alternating least-square*).

L'idée de cette méthode de résolution de factorisation est d'obtenir les règles de mise à jour des facteurs en calculant le gradient par rapport à ce facteur, et en trouvant une solution analytique. Ceci se fait généralement en annulant le gradient et en isolant le terme à mettre à jour pour connaître sa règle de mise à jour.

On commence avec la matrice latent d'utilisateurs $\mathbf{U}$. Le gradient par rapport à $U_{i.}$ se calcule :

$$\frac{1}{2}\frac{\partial L(U,V)}{\partial U_{i,k}} = \Sigma_j W_{ij}(U_{i.}V_{j.}^T - R_{ij})\, V_{jk} + \lambda(\Sigma_j W_{ij})\, U_{ik}, \quad \forall 1 \leq i \leq m, 1 \leq k \leq d. \tag{2}$$

On a donc

$$\frac{1}{2}\frac{\partial L(U,V)}{\partial U_{i,.}} = \frac{1}{2}(\frac{\partial L(U,V)}{\partial U_{i,1}}, \ldots, \frac{\partial L(U,V)}{\partial U_{i,d}})$$

$$= U_{i.}(V^T \tilde{W}_{i.} V + \lambda(\Sigma_j W_{ij})I) - R_{i.}\tilde{W}_{i.}V, \tag{3}$$

avec $\tilde{\mathbf{W}}_{i.} \in \Re^{n \times d}$ une matrice diagonale dont les coefficients de la diagonale principale sont $W_{i.}$, et $\mathbf{I}$ est une matrice identité $d \times d$.

Ainsi en annulant la dérivée et en isolant $U_{i.}$ de l'équation (3) on obtient :

$$U_{i.} = R_{i.}\tilde{W}_{i.}V(V^T\tilde{W}_{i.}V + \lambda(\Sigma_j W_{ij})I)^{-1}, \quad \forall 1 \leq i \leq m. \tag{4}$$

L'équation (4) est la règle de mise à jour de $U_{i.}$ (la i^{eme} ligne de $\mathbf{U}$) dans notre algorithme.

La règle de mise à jour de $V_{j.}$ s'obtient de la même façon :

$$V_{j.} = R_{j.}^T\tilde{W}_{.j}U(U^T\tilde{W}_{.j}U + \lambda(\Sigma_j W_{ij})I)^{-1}, \quad \forall 1 \leq j \leq n. \tag{5}$$

avec $\tilde{\mathbf{W}}_{.j} \in \Re^{n \times d}$ une matrice diagonale dont les coefficients de la diagonale principale sont $W_{.j}$.

On en déduit l'algorithme d'optimisation qui à chaque itération met à jour l'ensemble des vecteurs $U_{i.}$ et ensuite l'ensemble des vecteurs $V_{j.}$ jusqu'à convergence.

1.3 Application dans notre programme :

1.3.1 Algorithme

Optimisation alternée

@params matrice **R**, matrice de confiance **W**, nombre de latent **d**
Initialiser V (distribution Gaussian, moyenne = 0 et variance = 0.01)
Répéter
    Calculer $U_{i.}$, $\forall i$ avec l'équation (4)
    Calculer $V_{j.}$, $\forall j$ avec l'équation (5)
Jusqu'à ce que U,V convergent.
@return matrice approximé $\tilde{R}_{ij} = U_{i.}V_{j.}^T$

.

Après avoir obtenu la matrice approximée, pour un utilisateur donné, on choisi les 5 questions non votés ayant les plus grandes valeurs. Ce seront les questions recommendées pour cet utilisateur.

Exemple : Supposons que l'on a une matrice :

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Ici on a choisit le nombre de latents d = 3, le poids des réponses $W_{ij} = 0.18$ et le nombre de itérations = 10. La matrice de l'approximation est donc :

$$\begin{pmatrix} 1.054 & 0.842 & 0.782 & -0.116 & 0.291 \\ 0.308 & -0.148 & 0.702 & -0.143 & 0.282 \\ 0.248 & 0.818 & 0.01 & 1.03 & 0.809 \\ 0.739 & 0.278 & 1.12 & 0.139 & 0.724 \\ -0.168 & 0.168 & -0.149 & 0.628 & 0.427 \\ 0.799 & 0.982 & 0.165 & -0.016 & 0.022 \end{pmatrix}$$

Alors, par exemple pour l'utilisateur numéro 5, si on veut lui recommander une question, en probabilité ça sera la question numéro 5.

1.3.2 L'implémentation

1. Dans le fichier *array.php*, on fait une requête pour récupérer *la matrice utilisateur-question*. On l'encode en JSON et l'envoie au fichier principal *recommandation.js*.
2. Dans le fichier *indexUser.php*, on fait une autre requête pour récupérer *l'indice de l'utilisateur dans la matrice utilisateur-question* (ie. la ligne représentant l'utilisateur dans cette matrice). On l'encode en JSON et l'envoie au fichier *recommandation.js*.

3. Dans *recommandation.js*, on utilise AJAX et JQUERY pour récupérer les informations ci-dessus. On applique ensuite l'algorithme de l'optimisation alternée pour trouver la liste des résultats, qui est une liste d'identifiants correspondant aux questions recommandées aux utilisateur. Grâce à l'AJAX, on envoie ensuite la liste au fichier *recom.php* pour faire une requête et récupérer toutes les informations concernant ces questions pour l'affichage sur la page d'accueil *home.php*.

1.3.3 Remarque

1. Notre implémentation est seulement un prototype pour montrer la fonctionnalité de l'algorithme de l'optimisation alterné. Il nous manque encore une étape importante : *la régularisation des paramètres* (le nombre de latents, le poids des réponses), ce qui dépend essentiellement de la nature de la base de données. Étant donnée que l'on ne possède pas une "vraie" base de donnée, on s'arrête ici.
2. Vu que le PHP ne possède pas de bibliothèques pour les opérations matricielles, nous avons dû choisir le Javascript. C'est la première fois que nous l'avons utilisé et donc nous n'avons pas bien compris les concepts AJAX/QUERY pour communiquer entre JS, PHP et HTML. Effectivement, dans le tableau "Suggestion" de la page d'accueil, nous n'avons toujours pas réussi à afficher la colonne "Last Answer" (il y avait une erreur au niveau du format), d'où notre décision de la supprimer du tableau.