Recommandation personnalisée

Sous la direction de Marc Tommasi
Équipe Magnet, INRIA Lille Nord Europe

Xuedong Shang

Département Informatique et Télécommunications
Parcours Recherche et Innovation, L3

Ecole normale supérieure de Rennes et Université de Rennes 1

xuedong.shang@ens-rennes.fr

13 septembre 2015

Table des matières

Résumé

Ceci est un mémoire pour mon stage de L3 à l'Ecole normale supérieure de Rennes. Il s'agit de la recherche sur le système de recommandation pour le site de revendeur électronique *zalando.com*.

Dans ce rapport, nous allons tout d'abord étudier plusieurs approches classiques de type filtrage collaboratif, y compris la factorisation matricielle et le processus de décision markovien, puis proposer de nouveaux modèles qui s'adapteront éventuellement au site de Zalando. Il s'agit de mélanger les deux approches classiques afin de décentraliser le calcul.

Mots-clés. Apprentissage, factorisation matricielle, filtrage collaboratif, descente de gradient stochastique, moindres carrés, chaînes de Markov, N-gramme, processus de décision markovien.

1 Introduction

Le système de recommandation fait l'objet, à ce jour, d'un enjeu important pour les revendeurs électroniques. Un bon système de recommandation permet d'améliorer considérablement l'expérience utilisateur et ainsi permet aussi d'augmenter éventuellement le profit des entreprises.

1.1 Contexte

Le site *zalando.com* est une entreprise de commerce électronique allemande, spécialisée dans la vente de chassures et de vêtements, basée à Berlin. Créée en 2008 par **Rocket Internet**, elle présente dans 14 pays européens.

Zalando a décidé, en 2015, de collaborer avec certains laboratoires informatiques européens afin d'améliorer leur services en ligne. Notre équipe Magnet fait aussi partie de ces collaborations, dont l'objectif est d'améliorer leur système de recommandation actuel.

1.2 Méthodologies

La problèmatique principale récente dans ce domaine est la personnalisation des recommandations. Le problème de recommandation personnalisée consiste, pour un utilisateur donné, à prédire son score pour une certaine liste d'articles, afin que le revendeur puisse lui donner une recommandation. Le score ici désigne ainsi le niveau de préférence d'un utilisateur sur un certain objet.

Il existe principalement deux grandes classes d'approches classiques pour le problème de recommandation. Les premières sont les approches orientées contenu qui consistent à recommander des articles proches à ceux précédemment appréciés par les utilisateurs donnés. L'inconvénient important de cette approche est que des informations externes sont nécessaires, pourtant elles ne sont parfois pas faciles à trouver.

Une approche alternative est celle de type filtrage collaboratif. Il s'agit de recommander à un utilisateur donné des articles que d'autres utilisateurs ont appréciés. Dans notre rapport, nous allons nous concentrer sur l'une des méthodes classiques qui fait partie de type collaboratif, dite factorisation matricielle.

Il existe aussi d'autres types d'approches dans ce domaine. L'une des plus populaires est l'adaptation d'un processus de décision markovien **(MDP)**. Un MDP est un modèle stochastique issu de la théorie de la décision. Il peut être vu comme une chaîne de Markov à laquelle nous rajoutons une composante décisionnelle. Nous allons détailler ces modèles-là dans le rapport.

Une question qui se pose souvent dans les modèles précédents est le passage à l'échelle. Les données de Zalando, par exemple, sont d'une quantité immense. Les performances des algorithmes utilisés vont alors être mises à l'épreuve.

L'approche que nous proposerons ici va donc essayer de décentraliser l'apprentissage en y adaptant une machine d'états finis (une chaîne de Markov ou mieux un processus de décision markovien). Puis nous terminons par un algorithme classique de type factorisation matricielle à chaque état.

Dans ce qui suit, nous allons d'abord introduire la factorisation matricielle, une approche classique de type filtrage collaboratif[4]. Ce, ayant une intuition assez naturelle, a

pourtant été très efficace en pratique. Ensuite, nous allons détailler un peu comment nous nous profitons de la chaîne de Markov pour construire un modèle prédictif et l'utiliser dans un processus de décision markovien **(MDP)**. Ce dernier va nous servir aussi à faire des recommandations aux utilisateurs[6]. Puis dans la dernière partie, nous allons décrire le nouveau modèle que nous proposons.

2 Etat de l'art

2.1 Factorisation matricielle

L'idée principale de la factorisation matricielle est très intuitive, c'est-à-dire de trouver deux matrices dont la multiplication est égale à la matrice originelle. Cela nous permet de trouver des variables latentes cachées entre deux différents objets, dans notre cas, ça sera les utilisateurs et les articles. Ces variables latentes jouent des différents rôles lorsque un utilisateur choisit son article. Par exemple, deux utilisateurs qui sont fans du même acteur auront probablement la tendance à donner un score remarquable au même film.

2.1.1 Un peu de maths

Passons maintenant à des choses plus théoriques.

Supposons que nous disposons d'une matrice de scores R où se trouvent les scores d'un produit donnés par les utilisateurs, un ensemble d'utilisateurs U et un ensemble d'articles I. Supposons aussi que nous disposons d'un ensemble de variables latentes F. Notre objectif est donc de trouver deux matrices P, correspondant aux utilisateurs, et Q, correspondant aux articles, P sera de taille $|U| \times |F|$, et Q sera de taille $|I| \times |F|$ telles que :

$$\hat{R} = P^T \times Q \approx R$$

Ainsi la prédiction du vote de l'article q_i donné par l'utilisateur p_u est tout simplement le produit scalaire des vecteurs associés à p_u et q_i :

$$\hat{r}_{ui} = p_u^T q_i$$

Dans des travaux précedents, nous utilisons souvent l'imputation pour remplir la matrice creuse, mais cela peut causer des surapprentissages. Donc, dans notre modèle, nous n'allons utiliser que des données effectives, en y rajoutant un terme de régularisation pour éviter le surapprentissage. Ainsi nous cherchons à minimiser l'erreur quadratique moyenne :

$$\underset{P,Q}{\arg\min} \sum_{(u,i)\in\kappa} (r_{ui} - \sum_{f=1}^{F} p_{uf} q_{fi})^2 + \lambda(||p_u||^2 + ||q_i||^2)$$

Ici, κ désigne l'ensemble des couples (u,i) où r_{ui} est connu. Le terme de régularisation est souvent determiné par la validation croisée.

2.1.2 Algorithmes d'apprentissage

Il y a deux méthodes pour minimiser l'équation précédente.

Descente de gradient stochastique La descente de gradient stochastique **(Stochastic Gradient Descent)** est l'approche plus populaire pour minimiser notre fonctionnelle.

Nous posons $e_{ui}^2 = (r_{ui} - \sum_{f=1}^{F} p_{uf} q_{fi})^2 + \lambda \sum_{f=1}^{F} (||p_{uf}||^2 + ||q_{fi}||^2)$. Pour minimiser l'erreur, il faut décider dans quelle direction nous allons modifier la valeur de p_{uf} et q_{fi}, ainsi nous devons calculer les dérivées partielles de l'expression précédente par rapport à chaque variable p_{uf} et q_{fi}. Nous avons :

$$\frac{\partial}{\partial p_{uf}} e_{ui}^2 = -2(r_{ui} - \hat{r}_{ui})(q_{fi}) + 2\lambda p_{uf} = -2e_{ui} q_{fi} + 2\lambda p_{uf}$$

$$\frac{\partial}{\partial q_{fi}} e_{ui}^2 = -2(r_{ui} - \hat{r}_{ui})(p_{uf}) + 2\lambda q_{fi} = -2e_{ui} p_{uf} + 2\lambda q_{fi}$$

Par conséquent, nous mettons à jour les variables p_{uf} et q_{fi} dans la direction opposée à celle du gradient avec un pas γ :

$$p_{uf} \leftarrow p_{uf} + \gamma(e_{ui} q_{fi} - \lambda p_{uf})$$

$$q_{fi} \leftarrow q_{fi} + \gamma(e_{ui} p_{uf} - \lambda q_{fi})$$

Moindres carrés alternés Un algorithme alternatif est d'appliquer l'algorithme des moindres carrés alternés **(Alternating Least Square)** par rapport aux p_u en fixant les q_i, puis échanger leur rôles et appliquer de nouveau cet algorithme.

Cette approche est particulièrement efficace lorsque la parallélisation est appliquée ou quand il s'agit d'une matrice de votes non creusée.

2.1.3 Améliorations

La factorisation matricielle est une approche assez flexible, c'est-à-dire que nous pouvons y rajouter des différents aspects de données afin qu'elle puisse s'adapter à des différentes applications.

Ajout des biais Le premier élément que nous allons étudier sont des biais. Les biais sont des facteurs qui ne concernent soit les utilisateurs, soit les produits.

Prenons un exemple pour illustrer cette notion. Disons que la note moyenne de tous les films sur Netflix est 3/5. Le film Titanic est un film qui est considéré comme un bon film, donc recevra en général une note plus haute que la moyenne, disons que 3,5/5, ainsi le 0,5 est le biais pour cet article. D'autre côté, moi je suis un spectateur strict, je donne souvent 1 point de moins que la moyenne, ainsi le 1 est le biais pour moi en tant que utilisateur. Par conséquent, la note que je donne à Titanic sera $3 + 0,5 - 1 = 2,5$.

Nous pouvons formaliser ce propos facilement. Posons μ la moyenne globale, b_i le biais pour l'article i, b_u pour l'utilisateur u, alors la nouvelle note approchée sera donnée par la formule suivante :

$$\hat{r}_{ui} = \mu + b_i + b_u + p_u^T q_i$$

Nous cherchons donc à minimiser une nouvlle fonctionnelle :

$$\underset{P,Q,BI,BU}{\operatorname{argmin}} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_i - b_u - \sum_{f=1}^{F} p_{uf} q_{fi})^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_i^2 + b_u^2)$$

Retour de pertinence implicite En réalité, nous rencontrons souvent le problème de "cold start". Il y a des utilisateurs qui ne fournissent que très peu de votes, ce qui fait que c'est très difficile à prédire leur goût.

Nous utiliserons ainsi des retours de pertinence implicites **(implicit feedback)** pour démarrer notre apprentissage. Pour ce faire, nous introduisons un ensemble $N(u)$ qui est l'ensemble des articles préférés implicitement par l'utilisateur u. Il faut introduire aussi un nouveau vecteur de facteurs $x_i \in \mathbb{R}^{|F|}$ pour chaque article, ainsi la nouvelle préférence de l'utilisateur u est donnée par la somme $p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i$. Ici, nous avons normalisé la somme des x_i [3] [5].

Finalement, la nouvelle note approchée sera donnée par :

$$\hat{r}_{ui} = \mu + b_i + b_u + (p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i)^T q_i$$

Dynamiques temporelles Jusqu'à présent, notre modèle reste toujours statique, mais en réalité, le goût des utilisateurs évoluent au cours du temps, ainsi que la popularité des articles. Nous devrons ainsi prendre aussi en compte la nature dynamique des interactions utilisateurs-produits.

Le modèle de la factorisation matricielle est très adapté à ce faire. Il suffit en effet d'évaluer le vote en fonction du temps, ainsi que ses composants. Le seul terme qui ne varie pas au cours du temps est le vecteur q_i, en effet, la nature des produits ne change pas.

Nous obtenons ainsi une nouvelle estimation du vote :

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + p_u(t)^T q_i$$

Des entrées pondérées Un autre problème dans la réalité est que tous les votes ne sont pas aussi importants. Par exemple, il y a souvent des votes publicitaires qui ne reflètent pas vraiment la popularité d'un certain article. Ce problème s'introduit aussi dans le cas où le retour implicite se présente. En effet, le retour implicite ne donne pas des informations exactes sur la préférence de l'utilisateur. La seule information que nous disposons est qu'il est probablement intéressé ou non par certains produits. Ainsi, il faut discerner quelles sont les informations qui ont plus d'influences sur les votes. Pour résoudre ce problème, une idée logique est de présenter un poids, c'est-à-dire que nous cherchons à minimiser la fonctionnelle :

$$\sum_{(u,i) \in \kappa} c_{ui}(r_{ui} - \mu - b_i - b_u - \sum_{f=1}^{F} p_{uf} q_{fi})^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_i^2 + b_u^2)$$

Pour des vraies applications de cet effet, nous pouvons nous référer à [2].

2.1.4 Tests

Ce modèle est fait pour participer au concours de Netflix lancé en 2007. Les données sont offertes par Netflix y compris 1.000.000 votes sur 17.000 films. Environs 50.000 utilisateurs anonymes ont attribué à ces votes.

Nous pouvons tester des différents prédicteurs basés sur les différentes méthodes décrites ci-dessus et leur combinaisons.

Le système originel de Netflix a pour un RMSE de 0.9514 (sur un jeu de données fixé), alors que le concours a demandé une amélioration de 10%. Le grand prix a été remporté par l'équipe BellKors's Pragmatic Chaos en 2009.

2.2 Processus de décision markovien

Contrairement aux approches classiques qui traitent le problème de recommandation comme un problème de prédiction, le processus de décision markovien consiste à traiter ce problème comme un problème de décision séquentielle.

En pratique, un tel modèle nécessite un modèle initial très solide, que nous appelons modèle prédictif.

2.2.1 Modèle prédictif

Nous construisons un modèle markovien qui est basé sur le modèle du N-gramme.

N-gramme markovien L'idée du N-gramme semble provenir de la théorie de l'information. Il s'agit, à partir d'une certaine séquence de lettres, de prédire la lettre suivante en y effectuant une étude probabiliste et un corpus d'apprentissage.

Revenons à notre modèle. Comme il s'agit d'une chaîne de Markov, nous allons définir les états et une fonction de transition.

Les états dans notre modèle représentent des informations utiles de nos utilisateurs. Pour l'instant, nous nous restreignons à ne garder que les votes. Ainsi, l'ensemble des états contient toutes les séquences de choix. Cela posera naturellement des questions. Premièrement, les données peuvent être creuses. Deuxièmement, la complexité du calcul sera importante.

Une réduction de dimension semble donc raisonnable. Pour cela, nous nous limitons à des séquences de longueur inférieure à k et nous notons $(x_1, \cdots, x_k)$ un état dans lequel se trouve les k derniers articles que choisissait l'utilisateur. Cette technique est surtout adaptée aux certains domaines commerciaux. En effet, pour certains types de produits, seuls l'historique à court terme compte lorsque l'utilisateur effectue son prochain achat.

Nous pourrons ensuite définir la fonction de transition. Si les k derniers choix pour un utilisateur sont $x_1, \cdots, x_k$, alors la probabilité que le prochain produit choisi soit x' sera donnée par :

$$tr_{MC}((x_1, \cdots, x_k), (x2, \cdots, x_k, x'))$$

Cette fonction est inconnue au début, nous allons donc calculer une estimation du maximum de vraisemblance. Pourtant la performance n'est pas bonne, nous allons proposer donc quelques améliorations dans la section suivante.

Remarque 1. Nous pourrons aussi penser à un modèle non-séquentiel, où nous considérons que les séquences (x, y, z) et (y, z, x) sont le même état par exemple.

Améliorations Pour les améliorations suivantes, supposons que la longueur des états vaille 3.

Skipping Le *skipping* a été proposé par S.F. Chen and J. Goodman[1]. Supposons par exemple qu'une séquence de produits x_1, x_2, x_3 a été présente dans le modèle prédictif, alors il semble que la probabilité que les autres achètent x_3 après x_1 soit augmentée aussi.

Un simple modèle additif marche bien ici. Nous donnons un poids $c = \frac{1}{2^{j-(i+3)}}$ à la transition de l'état (x_i, x_{i+1}, x_{i+2}) à l'état (x_{i+1}, x_{i+2}, x_j) pour tout $i+3 < j \leq n$. En effet, il est raisonnable de considérer que la probabilité d'un *skipping* diminue lorsque le "pas" de *skipping*

augmente. Notons $c(s,s')$ comme le poids associé à la transition de s à s', alors nous pourrons définir des nouvelles probabilités de transition en effectuant une normalisation sur les poids :

$$tr_{MC}(s,s') = \frac{c(s,s')}{\sum_{s"} c(s,s")}$$

Clustering Une seconde amélioration[6] consiste à rassembler des différents états par leur "similarité". Par exemple, l'état (x,y,z) et l'état (w,y,z) sont similaires puisque certains objets apparaîssent à la fois dans les deux états.

Densité mélange La densité mélange consiste à mélanger les résultats issus des différents longueur k. En effet, avec un "grand" k, nous pouvons avoir plus d'informations sur ces résultats, mais pas assez de statistiques, ce qui est contraire au cas où la longueur k est trop "petite". Ainsi la densité mélange nous permet d'équilibrer cet effet en prenant la moyenne des différents résultats.

2.2.2 Une stratégie markovienne

Nous allons voir dans cette section comment nous construisons un système de recommandation en utilisant le processus de décision markovien **(Markov decision process)**, et comment y adapter le modèle prédictif décrit précédemment.

Définitions

Définition 2. Un processus de décision markovien est un quintuplet (S,A,R,tr,γ) où :

- $S = \{s_1,\cdots,s_{|S|}\}$ est l'ensemble fini discret des états possibles du système à contrôler
- $A = \{a_1,\cdots,a_{|A|}\}$ est l'ensemble fini discret des actions que nous pouvons effectuer pour contrôler le système
- $R : S \times A \times S \to \mathbb{R}$ est la fonction de récompense, elle indique la valeur obtenue lorsque nous effectuons une action a pour passer de l'état s à l'état $s^{'}$
- $tr : S \times A \times S \to [0;1]$ est la fonction de transition du système en réaction aux actions de contrôle, en général, c'est une probabilité
- $\gamma \in [0;1]$ est le facteur de "discount", ce facteur est important puisque les récompenses de future ont moins d'importances que les présents, généralement il est proche de 1

Ici, dans notre modèle, les états seront les mêmes que ceux du modèle prédictif. Une action correspondra à une recommandation d'un produit.

La fonction de récompense dépend des utilités définies par le site. Par exemple, la récompense pour un état (x_1,x_2,x_3) peut être le profit tiré de la vente du dernier article dans cette séquence. L'objectif de notre MDP est donc de maximiser la récompense au fils du temps.

Les utilisateurs peuvent soit accepter la recommandation, soit acheter un autre article. Pour la simplicité de notations, nous nous fixons à $k = 3$, et nous allons noter donc la fonction de transition comme suit :

$$tr_{MDP}((x_1,x_2,x_3),x',(x_2,x_3,x''))$$

Cela représente la probabilité que l'utilisateur achète le produit x'' étant donnée x' comme recommandation. À savoir que x'' peut être égal à x' ou non.

En raison de la nature des commerces électroniques, notre modèle doit être déjà assez précis dès son premier déploiement. En effet, les revendeurs ne veulent logiquement pas un système qui ne donne que des recommandations hors de sujet. Ainsi, il faut être très prudent lors de l'initialisation de la fonction de transition.

Cela peut se faire à l'aide de notre modèle prédictif. Supposons qu'une transaction ait plus de chances d'avoir lieu si elle est recommandée. Sa probabilité est proportionnelle à la probabilité en l'absence de recommandations :

$$tr_{MDP}((x_1, x_2, x_3), x', (x_2, x_3, x')) = \alpha \times tr_{MC}((x_1, x_2, x_3), (x2, x_3, x'))$$

D'autre part, supposons qu'une transaction ait moins de chances d'avoir lieu si elle n'est pas recommandée. Sa probabilité est aussi proportionnelle à la probabilité en l'absence de recommandations :

$$tr_{MDP}((x_1, x_2, x_3), x', (x_2, x_3, x'')) = \beta \times tr_{MC}((x_1, x_2, x_3), (x2, x_3, x'')), x'' \neq x'$$

Remarquons que dans les deux expressions ci-dessus, α est strictement supérieure à 1, elle est constante pour tous les états initiaux et x'. Alors que β est strictement inférieure à 1, elle, en revanche, est variable selon les différents états et x' de sorte que la somme des transitions vaille 1.

Algorithme Maintenant nous avons définit notre MDP, il faut trouver une politique π qui maximise la récompense. En effet, $\pi : S \rightarrow A$ est une suite qui indique pour chaque état quelle est l'action à effectuer. Ainsi, notre objectif est de trouver π telle que la sommme cumulative des récompenses (éventuellement infinie) soit optimale :

$$\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$$

où nous avons posé $a_t = \pi(s_t)$.

Des algorithmes classiques de ce genre de problème est basé sur deux tableaux V et π :

$$\pi(s) := \arg\max_a \{\sum_{s'} tr(s, a, s')(R(s, a, s') + \gamma V(s'))\}$$

$$V(s) := \sum_{s'} tr(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V(s'))$$

V est composé des valeurs réelles. À la fin de l'algorithme, le tableau π contiendra la solution que nous cherchons, et $V(s)$ désigne la somme des récompenses optimales en suivant les actions données par la solution à partir de l'état s.

Ici, l'algorithme que nous allons utiliser s'appelle "policy iteration", qui est l'une des variantes parmi les algorithmes classiques. Il s'agit de lancer la première étape pour une fois, puis répéter la deuxième étape jusqu'à ce qu'elle converge, et lancer la première étape encore une fois à la fin.

2.2.3 Modalités d'évaluation

Nous utilisons les deux scores suivants comme notre modalité d'évaluation.

RC (Recommendation Score) Il s'agit de vérifier si un produit est présent dans la liste des recommandations l (cette liste l varie selon les différentes expériences). Le score RC est alors le pourcentage de cas dans lequel ce produit se présente comme un article recommandé.

ED (Exponential Decay Score) Supposons que les utilisateurs aient plus de chance de voir une recommandation si celle-ci est présente parmi les plus hauts de la liste. Plus formellement, supposons que la probabilité qu'un utilisateur voit le m-ième produit vaille :

$$p(m) = 2^{-(m-1)/(\alpha-1)}$$

où α est la période biologique.

Ainsi le score est donné par :

$$100 \cdot \frac{\sum_{c \in C} p(m = pos(t_i|c))}{|C|}$$

où C désigne tous les cas testés, et $pos(t_i|c)$ désigne la position de t_i dans le cas c.

3 Système de recommandation pour Zalando

3.1 Les données de Zalando

Les données de Zalando sont composées de deux parties : une partie de **log data** qui consiste à décrire les comportements des utilisateurs, et une autre partie de **product descriptions** qui consiste à décrire les produits.

3.1.1 Historiques (Log data)

Une ligne typique de données sur les enregistrements est de la forme (remarquons que les variables étoilées sont les variables anonymisées) :

day-number | user* | session* | product-ID(s)* | action | quantity | price-reduction

Ici, nous avons fixé une certaine période de temps. Si le *day-number* égale à *d-2*, cela signifie que c'est le 2-ème jour dans la période courante. Si deux sessions ont lieu dans la même journée, alors elles sont ordonnée en fonction du temps. Ainsi l'ordre d'ouverture des sessions est conservé dans le **Log data**.

3.1.2 Description des produits (Products description)

Une ligne typique de données sur un produit est de la forme :

product ID* | product family ID* | brand* | color | commodity group 1 | commodity group 2 | commodity group 3 | commodity group 4 | commodity group 5 | age group adult | age group baby | age group teen | age group kid | gender male | gender female | price level

3.2 Modèle

Le modèle que nous pensons à tester ici est un mélange des deux approches précédentes, (i.e.) factorisation matricielle et le modèle markovien. Un problème primordial sur les données de Zalando est l'immensité de ces dernières. L'idée principale ici est donc de choisir certains états appropriés, puis leur associer chacun d'une matrice à factoriser afin de réduire la dimension des données. Cette matrice sert ainsi à donner une recommandation à nos utilisateurs. Dans un premier temps, nous allons considérer tout simplement une chaîne de Markov sans prendre en compte des actions dans des MDPs.

3.2.1 Une première idée

La première idée est de prendre l'historique sur les groupes de produits (commodity groups) comme états de notre chaîne de Markov. Comme le 5-ième niveau dans la hiérarchie des groupes de produits est composé de groupes d'une taille assez limitée, cela va nous donner une possibilité d'y adapter une factorisation matricielle.

Ensuite, nous associons à chaque groupe du 5-ième niveau une matrice dont nous expliciterons les caractéristiques plus tard. Ainsi à chaque état, nous obtenons une recommandation issue de la factorisation matricielle.

Supposons par exemple que nous soyons dans un état $e = (c_1, c_2, \cdots, c_k)$ où les c_i désignent des groupes du 5-ième niveau. Nous obtenons ainsi une recommandation pour l'état actuel. Ensuite, nous pourrons aussi avoir des recommandations issues des éventuels successeurs de e. Nous mélangeons alors ces dernières recommandations en respectant l'ordre décroissant des probabilités que nous passons de l'état actuel à ces états-là. Ceux que nous allons recommander à nos utilisateurs sont donc à la fois des recommandations correspondantes à l'état actuel et des recommandations issues des éventuels successeurs de l'état actuel dans notre chaîne de Markov.

Il reste à préciser le fonctionnement de la factorisation matricielle. Les deux dimensions de la matrice seront tout simplement un ensemble d'utilisateurs et un ensemble d'articles qui est, en l'occasion, l'ensemble des produits dans un certain groupe du 5-ième niveau. Il faut ensuite donner un score à chaque produit. Contrairement aux autres sites de notation comme MovieLens, Netflix, etc, un site comme Zalando ne dispose pas de notes explicites données par les utilisateurs. Nous pensons donc à donner un score implicite à chaque produit.

Le **Log data** dispose de l'information sur les actions des utilisateurs, nous allons en profiter pour construire un système de notation. Il y a au total 8 types d'actions :

- ☐ VIEW
- ☐ CART
- ☐ SALE
- ☐ WISH
- ☐ DELETE_FROM_CART
- ☐ DELETE_FROM_WISHLIST
- ☐ RECOPDS
- ☐ VIEWRECO

Le score implicite donné par un utilisateur à un certain produit dépend de la dernière action qu'il réalise sur le site. Nous pourrons, par exemple, désigner 5/5 pour une action de SALE, 4/5 pour une action de CART, 3/5 pour une action de WISH, 2/5 pour une action de VIEW et 1/5 pour une action de DELETE (que ce soit DELETE_FROM_CART ou DELETE_FROM_WISHLIST).

3.2.2 Timestamps

Dans le modèle précédent, nous n'avons pas pris en compte les timestamps dans les données. En effet, le goût d'un utilisateur peut effectivement changer si ce dernier revient sur le site au bout d'une période considérablement longue. Nous allons essayer de rajouter un facteur de temps dans les états, puis recalculer la fonction de transitions de notre chaîne de Markov.

Pour ce faire, nous allons considérer un état $(c_1, c_2, \cdots, c_k)$ dont les timestamps associés sont respectivement $(t_1, t_2, \cdots, t_k)$. Définissions $\delta = t_k - t_1$ comme le temps caractéristique pour cet état. δ est un nombre réel ce qui n'est pas très pratique, nous pourrons effectuer une discrétisation f sur celui-ci (en étudiant par exemple la distribution de probabilités sur les timestamps). Le nouvel état sera donc $(< c_1, c_2, \cdots, c_k >, f(\delta))$

4 Conclusion

Durant ce stage, nous avons étudié deux approches principales pour les systèmes de recommandation, la factorisation matricielle et le processus de décision markovien. Puis nous avons proposé de nouveaux mod´eles en essayant de mélanger les deux approches. La performance de notre modèle rest pourtant inconnu dû à la courte durée du stage et au problème de NDA entre l'Inria et Zalando.

Il nous reste donc beaucoup de travails à faire dans le future, èvaluer la performance de notre approche, étudier la complexité, proposer encore de nouveaux modèles, etc.

5 Remerciements

Je tiens à remercier toute l'équipe Inria Magnet pour m'avoir accueilli et accompagné pendant ce stage court, pourtant très enrichissant.

Et je tiens à remercier en particulier mon maître de stage Marc Tommasi pour m'avoir guidé pendant ces 2 mois, pour le temps passé à répondre à mes questions et à relire mon mémoire.

Références

[1] S.F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Technical report TR-10-98*, 1998.

[2] Y.F. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. *Proc. IEEE Int'l Conf. Data Mining (ICDM 08), IEEE CS Press*, pages 263–272, 2008.

[3] Y. Koren. Factorization meets the neighborhood : A multifaceted collaborative filtering model. *Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, ACM Press*, pages 426–434, 2008.

[4] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer Society*, (0018-9162) :42–49, 2009.

[5] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proc. KDD Cup and Workshop, ACM Press*, pages 39–42, 2007.

[6] G. Shani, R.I. Brafman, and D. Heckerman. An mdp-based recommender system. *Proc. UAI'02*, pages 453–460, 2002.