PSC MAP05 : Dynamic pricing et système de recommandation pour lutter contre le gaspillage alimentaire sur les plateformes de vente de nourriture

Dan BERREBBI, Lionel GAKO, Nicolas HUYNH, Ismael OUATTARA
Tuteur : Pascal BENCHIMOL

Avril 2020

Table des matières

Première partie
Introduction

L'avènement et l'utilisation massive de plateformes de livraison alimentaire telles qu'Uber Eats ou Deliveroo sont l'une des plus grandes transformations de la société au cours de ces dix dernières années. Ces plateformes proposent de plus en plus de nouvelles fonctionnalités : promotion pour les clients, minimisation de trajet pour la livraison, recommandation de plus en plus pertinente... Cependant, elles ne prennent pas en compte les stocks disponibles dans les différents restaurants et donc rien n'est fait pour éviter le gaspillage alimentaire.

Il nous est alors venu une idée pour notre projet : si, à une heure avancée dans la soirée, un restaurant a un stock trop important qu'il ne va pas réussir à écouler, la plateforme peut modifier le prix des plats, ce qui permettra de maximiser son profit et de diminuer le gaspillage. Augmenter la visibilité du produit sur la plateforme est aussi une possibilité pour permettre d'écouler un stock trop important. Comment recommander des produits aux utilisateurs ? Comment modifier le prix d'un produit afin de minimiser les stocks dans le restaurant en fin de soirée tout en maximisant le revenu du restaurateur ? Comment simuler un scénario réaliste de clients arrivant sur une plateforme, et choisissant de consommer l'un des produits recommandés, ou décidant de ne pas consommer ?

Notre objectif dans cette étude est d'apporter les réponses à ce questionnement. Pour cela, nous nous intéressons d'abord au problème de la recommandation. Ainsi, nous pouvons proposer aux utilisateurs des classements personnalisés de différents plats présents sur la plateforme. Il nous faut ensuite pouvoir augmenter la consommation de certains plats. La deuxième partie de notre projet est alors consacrée à la recherche d'une politique de prix optimal ayant pour objectif de minimiser les stocks. Les classements fournis par notre système de recommandation et les prix seront pris en compte lors de l'élaboration d'un scénario visant à simuler la consommation de clients pendant plusieurs soirées afin d' évaluer les performances de notre modèle.

Deuxième partie

Recommander des plats aux utilisateurs

1 Introduction

L'élément clé au départ de notre projet concerne la recommandation. En effet, la plateforme que nous voulons réaliser doit pouvoir avant toute chose proposer des **classements personnalisés** de produits aux utilisateurs. Chaque utilisateur doit, lorsqu'il se rend sur la plateforme, découvrir une sélection de produits qu'il est le plus susceptible d'apprécier et pas seulement une sélection des produits les plus populaires.

Plusieurs problématiques se posent alors :

— Comment deviner les préférences des utilisateurs parmi un vaste ensemble de produits disponibles,et ce à partir de quelques retours sur des produits déjà consommés ?
— Sous quel forme ces retours sont-ils renseignés par les utilisateurs ?
— Y-a-t-il différents types de retours ? Lesquels sont les plus pertinents à évaluer ?
— De plus, comment évaluer la qualité d'un système qui prédit ces préférences ?
— Enfin, si plusieurs démarches sont possibles, laquelle choisir ?

Pour répondre à ces questions et aboutir à un algorithme de recommandation performant, nous allons tout d'abord présenter les différentes familles de systèmes de recommandation, formaliser leur mode de fonctionnement, et enfin présenter les métriques qui nous permettront d'évaluer un système de recommandation.

Nous allons ensuite décrire le principe de la factorisation matricielle, méthode que nous avons choisie, ainsi que les techniques mathématiques utilisées pour réaliser une **factorisation matricielle**.

Après cette partie consacrée à l'étude théorique de la factorisation matricielle, nous présenterons les jeux de données utilisés et les implémentations de nos algorithmes.

Enfin, la dernière partie présentera et exploitera les résultats de nos implémentations afin de sélectionner le meilleur modèle pour recommander, puis d'optimiser ses performances sur les différents jeux de données.

L'étude réalisée dans cette partie répond à plusieurs nécessités dans l'élaboration de notre projet :

• Proposer aux utilisateurs un classement pertinent de produits

• La connaissance de ces classements personnalisés sera prise en compte dans la partie suivante pour prédire la demande et ainsi ajuster les prix afin d'augmenter la consommation de certains produits

2 Les systèmes de recommandation

2.1 Deux types d'approches possibles

Nous présentons deux types d'approche possibles en recommandation : **l'approche basée sur le contenu** et **l'approche collaborative** [1].

• **L'approche basée sur le contenu :** l'idée de l'approche basée sur le contenu est de recommander à un utilisateur des produits qui sont similaires à ceux qu'il a appréciés par le passé. Par exemple, si un utilisateur d'un service de livraison de nourriture commande majoritairement des plats japonais, le but du système est d'apprendre cette préférence puis de recommander à cet utilisateur d'autres plats japonais.

• **L'approche collaborative** le **filtrage collaboratif** est une approche basée sur le partage d'opinions entre les utilisateurs. Ces méthodes supposent que si **des utilisateurs ont les mêmes préférences** sur un ensemble d'items, alors ils auront probablement les mêmes préférences sur un autre ensemble d'items qu'ils n'ont pas encore notés. On parle d'utilisateurs similaires.

Enfin il existe également des approches qualifiées d'hybrides car elles exploitent les similarités entre produits et les similarités de goûts entre personnes. La factorisation matricielle que nous décrirons en partie 3.2 est un exemple de système hybride.

2.2 Formalisation du problème

Remarque : nous utiliserons le mot anglais "item" pour désigner un produit (plat, livre, film...), et le mot "rating" pour désigner une note de manière interchangeable. De même, nous utiliserons "dataset" pour jeu de données.
Définition : On appelle rating explicite une note donnée par un utilisateur à un produit, et rating implicite une note binaire égale à 1 si l'utilisateur a consommé le produit. Dans notre projet, nous n'avons considéré que des ratings explicites.

On considère un ensemble U d'utilisateurs et un ensemble I d'items. Pour un utilisateur u et un item i, on note $r_{u,i}$ la note donnée par l'utilisateur u à l'item i **lorsque celui renseigne cette note**. Dans notre étude, nous considérerons uniquement des **ratings explicites**, avec $r_{u,i} \in [\![1, 5]\!]$. Nous notons R l'ensemble des notes dont on dispose. Comme nous le verrons lors de la présentation des jeux de données sur lesquels nous avons travaillé, **l'ensemble R est très petit** devant l'ensemble possible des notes, de taille $|U| \times |I|$. En effet, chaque utilisateur n'a **noté qu'un très faible nombre d'items** en comparaison au nombre d'items disponibles sur la plateforme.

La procédure pour recommander est composée de **deux étapes** [2] :
• A partir des notes de l'ensemble R, prédire les notes pour tous les couples (u, i) possibles avec $u \in U$ et $i \in I$. Nous noterons $\hat{r}_{u,i}$ ces notes prédites.
• A partir de ces notes prédites et en fonction des objectifs de la plateforme, recommander à chaque utilisateur certains items.

Bien que l'on souhaite prédire une note $\hat{r}_{u,i} \in [1, 5]$ à partir d'un jeu de données (ensemble de notes R), il ne s'agit pas d'un problème d'apprentissage supervisé classique de régression ou classification. En effet,les différents utilisateurs n'ont pas renseigné des notes pour les mêmes plats et nous ne pouvons donc pas définir un espace de variables explicatives et un espace de variables cibles communs aux utilisateurs. Pour plus de clarté, les différences entre les problèmes classiques de prédiction et notre problème de recommandation peuvent se résumer sur la figure 1 représentant les jeux de données, les '?' représentant les données manquantes à prédire.

Figure 1 – Matrice de données dans le cas d'un problème de régression (gauche) et dans le cas d'un problème de recommandation (droite).

Ainsi, notre objectif est de trouver un moyen de remplir tous les trous de cette matrice de données avec les notes prédites $\hat{r}_{u,i}$ de la manière la plus pertinente possible.

2.3 Evaluation des prédictions

Les plateformes existantes évaluent leurs prédictions en deux étapes principales :
• Sélection des quelques modèles de prédiction les plus performants de manière "offline" à partir de notes déjà renseignées.
• Test de ces modèles sur des ensembles d'utilisateurs ("online") différents. Celui donnant les meilleures recommandations est conservé (principe de l'A/B testing).
Comme nous ne pouvons faire de test "online" avec des utilisateurs réels, nous allons nous concentrer sur des évaluations "offline", à partir de l'échantillon de notes dont nous disposons au départ. Nous décomposerons cet échantillon en deux parties : une partie pour l'entraînement et une partie pour le test. Dans la pratique nous utiliserons la validation croisée K-fold dans notre implémentation (partie 4.3).
Nous pouvons alors introduire plusieurs métriques.

2.3.1 Exactitude

Deux métriques usuelles et intuitives pour évaluer l'exactitude de nos prédictions sont **l'Erreur Quadratique Moyenne** (EQM) et **l'Erreur Absolue Moyenne (EAM)** ([3] partie 2.3.3), toutes deux moyennées sur les échantillons de tests utilisés lors de la validation croisée.
Voici l'expression mathématique de ces quantités :

$$\boldsymbol{EQM = \frac{1}{|R_{test}|} \cdot \sum_{r_{u,i} \in R_{test}} (\hat{r_{u,i}} - r_{u,i})^2}$$

$$\boldsymbol{EAM = \frac{1}{|R_{test}|} \cdot \sum_{r_{u,i} \in R_{test}} |\hat{r_{u,i}} - r_{u,i}|}$$

L'Erreur Absolue Moyenne est celle que l'on peut interpréter le plus facilement : elle nous renseigne sur l'écart moyen entre la note prédite et la note réelle renseignée par l'utilisateur. Cependant, cette métrique ne pénalise pas plus les grandes erreurs que les petites. Considérons deux algorithmes que l'on évalue sur un échantillon R_{test} tel que $|R_{test}| = 1000$. Prenons $r_{u,i} \in [\![1, 5]\!]$. Le premier algorithme est parfaitement exact sur 900 éléments et se trompe totalement sur les 100 restants en faisant une erreur absolue de 3 ($|\hat{r_{u,i}} - r_{u,i}| = 3$). Le deuxième se trompe légèrement sur tout l'échantillon de test avec $|\hat{r_{u,i}} - r_{u,i}| = 0.5$. On a $EAM_1 = 0,3 < EAM_2 = 0,5$ alors que l'algorithme

2 semble bien plus pertinent.

L'Erreur Quadratique Moyenne, quant à elle, pénalise les grandes erreurs et minimise le poids des petites erreurs. La situation précédente donne par exemple $EAQ_1 = 0,9 > EQM_2 = 0,25$. C'est pourquoi nous privilégierons l'EQM à l'EAM pour évaluer l'exactitude de nos prédictions. De plus, l'EQM est convexe et différentiable, ce qui est non négligeable dans un problème d'optimisation.

L'Erreur Quadratique Moyenne est donc très importante, néanmoins elle peut être trompeuse et rendue assez faible même par un très mauvais algorithme. C'est le cas par exemple d'un algorithme qui prédirait pour tout couple utilisateur-item une note égale à la moyenne des notes renseignées. En effet, les notes sont comprises entre 1 et 5, mais la plupart des utilisateurs donnent une note plutôt positive (entre 3,5 et 4,5). On observe alors une variance des notes renseignées d'environ 1 seulement, et ainsi, l'algorithme qui ne ferait que prédire la moyenne par exemple aurait une EQM d'environ 1, ce qui n'est pas si mauvais. Il va donc falloir s'intéresser à d'autres métriques pour compléter l'EQM et évaluer notre algorithme de prédiction.

2.3.2 Précision, rappel (precision and recall), precison@k et recall@k

La précision et le rappel, en anglais precision and recall, sont deux mesures qui évaluent **la pertinence** des recommandations faites par le système : on choisit un seuil $s \in [1, 5]$ pour les notes, par exemple $s = 4$, et on définit ensuite un item comme pertinent si sa note est au dessus de ce seuil s. On remarque tout de suite que l'on peut définir deux ensembles de pertinence : celui avec les notes renseignées, et celui avec les notes prédites (uniquement pour $r_{u,i} \in R_{test}$ sinon nous ne pouvons comparer avec les vraies notes). On définit alors :

$$\boldsymbol{I^s_{pred} = \{i \in I \quad | \quad \exists u \in U, \quad r_{u,i} \in R_{test}, \quad \hat{r}_{u,i} \geq s\}}$$

$$\boldsymbol{I^s_{reel} = \{i \in I \quad | \quad \exists u \in U, \quad r_{u,i} \in R_{test}, \quad r_{u,i} \geq s\}}$$

La précision est alors définie comme la part d'items jugés à juste titre pertinents après prédiction parmi les items jugés pertinents après prédiction (nombre de vrais positifs sur nombre de positifs prédits).[4]

$$\boldsymbol{precision = \frac{|I^s_{pred} \cap I^s_{reel}|}{|I^s_{pred}|}}$$

Le rappel est défini comme la part d'items jugés à juste titre pertinents après prédiction parmi les items pertinents (nombre de vrais positifs sur nombre de positifs réels).

$$\boldsymbol{rappel = \frac{|I^s_{pred} \cap I^s_{reel}|}{|I^s_{reel}|}}$$

On remarque que si on n'évalue que l'une de ces mesures, cela n'a pas de sens car on peut la rendre arbitrairement grande : considérons par exemple un algorithme qui donne une note de

5 à tous les couples utilisateur-item. Cet algorithme est très mauvais, mais son rappel vaut 1 car $I^s_{reel} \subset I^s_{pred}$ pour tout seuil s. En revanche, sa précision va être égale à $\frac{|I^s_{reel}|}{|I^s_{pred}|}$, et donc pour s pas trop petit on a $|I^s_{reel}| << |I^s_{pred}| = |R_{test}|$, donc une très mauvaise précision. Ainsi, pour souligner cette dépendance entre précision et rappel nous évaluerons aussi une autre métrique, le F-score défini par : $F = 2 \times \frac{precision \cdot recall}{precision \ + \ recall}$ dont la valeur est très faible si la précision ou le rappel est très faible.

De plus, la précision et le rappel utilisent un seuil, cela semble logique car nous allons recommander les produits ayant les meilleures notes, donc ayant les notes au dessus d'un certain seuil s. Néanmoins nous n'allons recommander qu'une dizaine de produits à chaque utilisateur et ce sont **uniquement** ces 10 produits recommandés dont nous devons évaluer la pertinence. C'est pourquoi il semble plus pertinent ici d'introduire une variante de la précision et du rappel plus spécifique à notre problème : la "precision@k" et le "recall@k" [3] (partie 2.3.3) et [4]. Il s'agit des mêmes quantités, mais les ensembles sur lesquels nous travaillons sont uniquement constitués des k items les mieux notés (avec les notes réelles pour $I^{k,s}_{reel}$ et les notes prédites pour $I^{k,s}_{pred}$). Nous comparons donc, en nous restreignant à l'échantillon de test, le top k réel des items avec le top k prédit.

Ainsi, pour évaluer les performances de nos algorithmes nous regarderons les métriques suivantes :
- EQM et EAM
- precison@k, recall@k et F-score@k (avec $k = 10$ en pratique).

3 Les fondements mathématiques de la factorisation matricielle

Nous présentons la méthode de factorisation matricielle car c'est elle qui nous a permis d'obtenir les meilleures performances (partie 5.1). Nous commençons par rappeler le principe de la décomposition en valeurs singulières.

3.1 La Décomposition en Valeurs Singulières (SVD)

Proposition 1 : Les matrices à valeurs réelles $R \in M^{m*n}$ de rang r peuvent être décomposées comme produit de 3 matrices : $R = U\Sigma I^t$ où $U \in M^{m*r}, \Sigma \in M^{r*r}$ est une matrice diagonale, et $I \in M^{n*r}$. U et I sont des matrices orthogonales (c'est à dire telles que $UU^t = \mathbb{I}$).

Cette factorisation est appelée Décomposition en Valeurs Singulières (qu'on désignera par SVD dans la suite du rapport) de R.

On sait comment déterminer les matrices U, I et Σ. Les colonnes de U sont les vecteurs propres de la matrice RR^t, celles de I sont les vecteurs propres de la matrice R^tR, et leurs valeurs propres associées sont le carré des valeurs de Σ, appelées **les valeurs singulières**.

La décomposition peut être écrite $R = U.I^t$, où Σ a été fusionné avec U ou I.

3.2 La factorisation matricielle

La factorisation matricielle est une technique qui s'inspire de la SVD. Elle consiste à **décomposer la matrice note M sous la forme d'un produit de matrices** $U \times V$ où M est de la forme

présentée à la figure 1. U est de taille $n \times l$ et V est de taille $l \times m$, avec l un nombre petit devant la taille des données ($l \ll n$, $l \ll m$) qui est appelé nombre de **facteurs latents**.[5]

Une fois U et V trouvés, on dispose de notes pour **tous les couples** utilisateur-item et on peut donc recommander les "meilleurs" produits pour chaque utilisateur.
Le schéma ci-dessous illustre le principe de la factorisation matricielle pour $n = 4, m = 4, l = 2$.

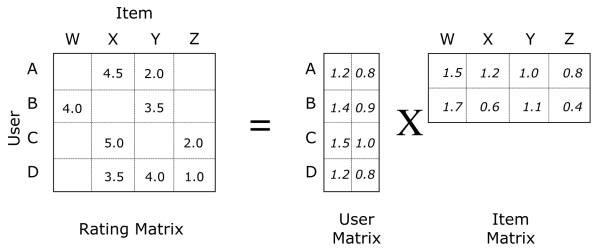

Figure 2 – Illustration de la factorisation matricielle pour la recommandation [6]

La méthode de SVD donne la forme désirée pour une factorisation matricielle. De plus elle a permis d'introduire le concept de facteur latent. Cependant la proposition 1 n'est vraie que pour une matrice dont tous les coefficients sont renseignés, ce qui n'est pas le cas dans le problème de la recommandation. Nous ne pouvons pas utiliser la solution analytique proposée par la théorie de la SVD.

3.3 Adaptation de la SVD

Le problème peut se ramener à la minimisation de la perte suivante :

$$\sum_{r_{u,i} \in R} (r_{ui} - U_i^t.V_u)^2$$

avec R l'ensemble des notes renseignées par les utilisateurs.

Remarques :

• Afin d'éviter le sur-apprentissage et avoir une meilleure prédiction, il faut régulariser cette perte : $\sum_{r_{u,i} \in R} (r_{ui} - U_i^t.V_u)^2 + \alpha \times \|U_i^t\|^2 + \alpha \times \|V_i\|^2$.

• Cette perte n'est pas convexe car U et V varient. Deux techniques sont utilisées dans la pratique pour résoudre ce problème non-convexe. La première est une descente de gradient stochastique, et la seconde est la technique d'"Alternative Least Square" (ALS). Le principe d'ALS est le suivant : à chaque itération, on fixe U et on minimise la perte (convexe) selon V, puis on fixe V et on minimise la perte (convexe) selon U [5].

3.4 Les facteurs latents

La factorisation matricielle de la matrice M (dimension m x n) permet de la décomposer en 2 matrices U (m x k) et V (k x n) où m représente le nombre de users, n le nombre d'items et k le

nombre de facteurs latents. En d'autres termes, on obtient un vecteur de dimension k pour chaque utilisateur et pour chaque item.

Le mot latent veut dire "non observable", et en effet les vecteurs dans cette dimension n'ont aucune signification particulière. Néanmoins les vecteurs des users et items "vivent" maintenant dans le même espace $\mathbb{R}^k$ et on peut calculer les distances (similitudes) user-item, user-user et item-item dans cet espace (k est bien plus petit que m et n, mais ce sont des vecteurs de réels et non pas d'entiers, donc beaucoup plus expressif). Ainsi 2 items proches dans cet espace latent ($\mathbb{R}^k$) possèdent en général des traits en commun (même genre de film, même cuisine...). [5]

4 Les méthodes d'implémentation

4.1 Les données

Pour implémenter et tester des algorithmes de factorisation matricielle, nous avons besoin de jeux de données (datasets) comportant un ensemble d'utilisateurs, un ensemble d'items, et des notes pour certains couples utilisateur-item. La figure 3 montre la forme des datasets (après traitement) qui est le point de départ de nos implémentations :

user_id	book_id	rating
44650	332	2
1420	25	3
45248	478	4
16411	2994	4
22044	3935	4

Figure 3 – Forme typique des données utilisées, extrait du dataset sur les livres [7]

Nous avons retenu pour nos implémentations et tests les datasets suivants : un dataset sur les plats dans les restaurants [8] , les versions '100k' et '1M'(nombre de notes renseignées) de MovieLens [9] [10] (célèbre dataset sur des films), et les versions 'normales' [7] et 'updated' [11] (la version updated contient bien plus de notes que la normale) du dataset 'goodbooks' sur les livres. Ces datasets ne concernent pas forcément le domaine alimentaire mais cela n'a pas d'importance, il nous faut seulement des ensembles cohérents et réels de notes renseignés pour tester nos algorithmes. La figure 4 présente un récapitulatif des caractéristiques de ces datasets.

Datasets	Utilisateurs	Items	Notes renseignées	Taille totale	Plein à
Plats	138	130	1161	17940	6,47%
Films 100k	671	9066	100004	6083286	1,64%
Films 1M	6040	3706	1000209	22384240	4,47%
Livres 1	53424	10000	981756	534240000	0,18%
Livres 2	53424	10000	5976479	534240000	1,12%

Figure 4 – Caractéristiques des ensembles de données utilisés

4.2 Les algorithmes

Nous avons choisi d'implémenter en Python car il existe plusieurs librairies faciles d'utilisation pour implémenter les algorithmes. Nous avons utilisé la librairie "sklearn" (le module "surprise") [12], ainsi que la librairie "keras" pour le réseau de neurones.

Plusieurs méthodes sont possibles pour prédire les notes manquantes : méthodes de clustering, méthodes par plus proches voisins ("KNN"), et méthodes par factorisation matricielle ("SVD", "NMF", ou à l'aide d'un réseau de neurones : ces techniques diffèrent dans leur manière de minimiser la perte).

Remarque : nous donnons sur notre GitHub [13] une implémentation avec un réseau de neurones à une couche (introduction d'une seule fonction non linéaire après avoir fait le produit matriciel). Ce modèle s'est révélé beaucoup moins intuitif et moins performant dans un premier temps, nous ne l'avons donc ni conservé ni comparé aux autres modèles dans les comparaisons finales.

Enfin, nous avons également développé une implémentation complètement personnelle que nous avions codée au début de notre projet afin de mieux comprendre le problème de la prédiction de notes. Cette démarche, certes longue, est une réussite car elle nous a permis de bien comprendre comment faire une factorisation matricielle. De plus l'algorithme fonctionne, mais il est bien plus lent que ceux de sklearn. C'est la raison pour laquelle nous ne le décrirons qu'en annexe A.

4.3 La méthode d'évaluation des algorithmes

Afin d'évaluer les métriques introduites en partie 2.3 pour les différents algorithmes possibles, nous avons utilisé la méthode dite "K-fold cross validation" (validation croisée). Elle consiste à partitionner le jeu de données initial en k ensembles de tailles égales. Ensuite, on entraîne un modèle de prédiction de notes à partir de $(k-1)$ de ces k ensembles. Puis on évalue les performances sur l'échantillon non utilisé pour l'apprentissage, afin d'avoir des mesures de performance sur un échantillon que le modèle n'a jamais rencontré avant la phase de test. On répète l'étape précédente k fois, en prenant un ensemble de tests différent à chaque itération. Enfin, l'évaluation finale de l'algorithme est réalisée en moyennant les évaluations de chacune des itérations. Le fait d'évaluer le modèle en changeant les ensembles d'entraînement et de test donne plus de robustesse à notre évaluation, car cela évite par exemple d'avoir des résultats faussés par un ensemble de test atypique. La validation croisée est déjà implémentée pour certaines métriques usuelles dans sklearn, mais pour la précision@k et le recall@k nous avons dû l'implémenter [14].

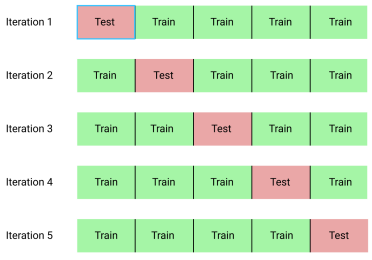

Figure 5 – Illustration du principe de la validation croisée k-fold avec $k = 5$ (site medium.com)

5 Résultats et comparaisons

5.1 Choix de l'algorithme le plus performant

Nous avons comparé les différents algorithmes évoqués en section 4.2 en les exécutant sur tous les datasets présentés en partie 4.1 (GitHub [15] et [14]). Nous avons aussi évalué les performances de l'algorithme naïf qui prédit, pour toute note, la moyenne des notes de l'ensemble d'entraînement, afin de souligner la nécessité d'évaluer le F-score [16]. Les résultats sur les datasets avec 100 000 et 1 million de notes sur les films sont regroupés dans les figures 6 et 7. Les algorithmes itératifs sont tous executés avec 10 itérations, car nous avons remarqué que les valeurs des métriques sont stables après une dizaine d'itérations pour ces algorithmes sur ces datasets.

Algorithme	EQM	EAM	Precision at 10	Recall at 10	F-score	Temps (s)
SVD	0.814	0.697	0.859	0.313	0.459	1.6
KNN	0.861	0.710	0.841	0.343	0.486	0.4
NMF	0.946	0.738	0.851	0.250	0.386	2.1
Clustering	0.948	0.754	0.828	0.349	0.491	4.9
Naïf Constant	1.12	0.850	1	0.030	0.058	0.13

Figure 6 – Comparaison des performances de différents algorithmes dans la prédiction des notes sur le dataset film 100k

Algorithme	EQM	EAM	Precision at 10	Recall at 10	F-score	Temps (s)
SVD	0.822	0.718	0.879	0.359	0.510	18.0
Clustering	0.842	0.719	0.862	0.393	0.540	36.6
KNN	0.870	0.742	0.889	0.326	0.478	45.8
NMF	0.909	0.741	0.905	0.228	0.364	20.0
Naïf Constant	1.25	0.934	1	0.008	0.016	1.52

Figure 7 – Comparaison des performances de différents algorithmes dans la prédiction des notes sur le dataset film 1M

Analyse des résultats :

• La factorisation matricielle par SVD est l'algorithme le plus exact et le plus rapide sur les grands datasets.

• Néanmoins, en analysant plus en détail les notes prédites par l'algorithme de SVD, nous avons remarqué que ces notes étaient assez regroupées autour de la moyenne, la variance des notes prédites étant environ deux fois plus faible que la variance des notes renseignées. C'est pourquoi il est essentiel de vérifier que le F-score de cet algorithme n'est pas faible par rapport aux autres.

• L'algorithme de Clustering semble meilleur au niveau du F-score que le SVD. Cependant, il est moins exact (EQM et EAM) sur des datasets de petite taille comme celui des restaurants ou des films 100k. De plus, une comparaison SVD-Clustering sur le plus grand de nos datasets (livres 2) nous confirme que le SVD est beaucoup plus rapide, plus exact et a même un meilleur F-score sur les datasets de très grande taille (figure 8).

Algorithme	EQM	EAM	Precision at 10	Recall at 10	F-score	Temps (s)
SVD	0.729	0.666	0.878	0.365	0.520	84.6
Clustering	0.749	0.672	0.879	0.333	0.484	154.1

Figure 8 – Comparaison des performances de SVD et Clustering sur le dataset livres 2 avec 5,9 millions de notes renseignées

Ainsi, après ces comparaisons, nous avons choisi d'utiliser l'algorithme de factorisation matricielle par SVD pour prédire les notes manquantes.

5.2 Optimisation des paramètres de la SVD

Maintenant que nous avons choisi l'algorithme de SVD, il faut optimiser ses performances.

Comme nous l'avons expliqué dans la partie 3.3, l'algorithme de factorisation matricielle par SVD minimise la perte $\sum_{r_{u,i} \in R} (r_{ui} - U_i^t . V_u)^2 + \alpha \times \|U_i^t\|^2 + \alpha \times \|V_i\|^2$ en utilisant une descente de gradient stochastique (plus rapide que ALS dans la pratique). Comment fixer les différents paramètres de l'algorithme ? Combien faut-il prendre de facteurs latents ? Quel taux d'apprentissage fixer ? Quel facteur met-on pour régulariser ? Ces paramètres appelés hyperparamètres vont influer sur la performance de notre modèle. Il faut donc trouver, pour un dataset donné, la combinaison de paramètres qui donne les meilleures performances.

Sklearn propose une méthode permettant de tester toutes les combinaisons de paramètres possibles (GridSearchCV), l'utilisateur spécifiant les ensembles dans lesquels il veut faire varier ses paramètres. On peut ensuite sélectionner la "meilleure" combinaison, meilleure pour la métrique choisie.
En choisissant comme métrique 'MSE', c'est à dire l'Erreur Quadratique Moyenne, voici les résultats de cette recherche sur le dataset film 1M (GitHub [17]). La figure 9 montre les paramètres que nous cherchons à fixer et le domaine sur lesquels nous les avons fait varier pendant la GridSearch.

Parametre	Domaine
Facteurs Latents	$\{8, 15, 20\}$
Nombre d'epochs	$\{10, 20\}$
Taux de regularisation	$\{0.01, 0.05\}$

Figure 9 – Domaines de variation des paramètres pendant la GridSearch

On prend ensuite les valeurs des hyperparamètres qui donnent l'EQM la plus faible et on compare les performances à celles que l'on avait obtenu avant ce « tuning des paramètres ».

	EQM	Precision at 10	Recall at 10	F-score
SVD	0.822	0.879	0.359	0.510
SVD optimisé pour MSE	0.753	0.892	0.358	0.511

Figure 10 – Comparaison des performances de SVD avant et après optimisation sur les hyperparamètres sur le dataset film 1M [10]

On remarque que cette recherche sur les hyperparamètres nous a permis de diminuer l'EQM sans diminuer les performances au niveau de la pertinence de l'algorithme (F-score, precision, rappel).

Une fois ce choix optimal des paramètres réalisé, nous pouvons utiliser notre algorithme et prédire les notes efficacement.

Conclusion de la partie II

Ainsi, deviner les préférences d'un utilisateur se traduit en une tâche de prédiction d'un score (la note qu'il aurait donné un produit). Plusieurs types de systèmes permettent de réaliser ces prédictions.
Après comparaison des performances, et après avoir lu de nombreux articles privilégiant cette méthode, nous avons choisi la factorisation matricielle. Une comparaison des performances, cette fois sur deux méthodes (SVD et NMF), nous a amené à choisir une technique de SVD pour implémenter cette factorisation matricielle. Cette étude nous a donc permis d'élaborer un système de recommandation, ce qui nous permettra de fournir aux utilisateurs un classement des produits. Ainsi nous pourrons produire un scénario de demande s'appuyant sur ce classement lors de l'évaluation des performances du modèle final.
Enfin, nous avons fait le choix du rating explicite et non du rating implicite. Nous reviendrons sur ce choix lors de la conclusion finale.

Troisième partie

Pricing dynamique avec recommandation

6 Introduction

La partie précédente s'est intéressée au problème de la recommandation à partir du feedback des utilisateurs. Cette idée de pouvoir capter le comportement des utilisateurs et de pouvoir le moduler via la recommandation est très intéressante. Cependant, tout un volet est encore à prendre en compte : il s'agit de l'influence du prix sur la consommation. L'enjeu principal de cette partie est donc de concilier une stratégie autour du pricing dynamique et une stratégie de recommandation pertinente.

Langage Nous disons que chaque soirée est composée de périodes (par exemple des périodes de 10 minutes).

7 Scénario de simulation : comment générer des données ?

Afin de pallier l'absence de données, nous avons décidé de créer un scénario de simulation qui nous permettrait de mettre en avant des résultats quantitatifs. Ce scénario doit vérifier deux points :

— **(A1)** le scénario doit être le plus proche possible de la réalité.

— **(A2)** le scénario doit être le plus hétérogène possible.

Justifions ces deux assertions.

La première (**A1**) est naturelle car idéalement, la simulation pourrait s'appuyer sur des données réelles, donc pour juger de l'efficacité de la méthode, il nous faut disposer d'un scénario réaliste.

La deuxième assertion (**A2**) provient du fait qu'une estimation de la demande sera nécessaire. Pour cela, il faut modéliser la demande et le risque est d'utiliser le même modèle que celui qui permet la génération des données. Ainsi, il faut un modèle suffisamment hétérogène (donc complexe) pour que le modèle de régression ne soit pas identique au modèle de génération des données.

Nous proposons deux solutions pour répondre à ces assertions :

— nous modélisons le comportement d'un utilisateur par les différentes étapes qu'il suivrait dans la vie réelle : arrivée, choix du plat (si commande il y a)

— nous effectuons des segmentations de la clientèle pour modéliser plusieurs comportements afin d'ajouter de l'hétérogénéité.

Nous nous inspirons de [18], où un scénario de demande est décrit. Notre apport dans cette partie est la prise en compte des notes des utilisateurs ainsi que l'influence du ranking sur le choix des utilisateurs.

7.1 Présentation du scénario de simulation et hypothèses

Figure 11 – Scénario de simulation

Nous faisons les hypothèses suivantes pour notre scénario :
- **(H1)** Le nombre de clients susceptibles d'utiliser la plateforme est fixé au début.
- **(H2)** Le nombre de clients qui arrivent est influencé uniquement par la période.
- **(H3)** Le comportement de l'utilisateur concernant le choix de son plat est motivé par deux éléments : le prix, son goût pour le plat, le classement des plats qui lui est proposé.

Nous justifions les hypothèses mentionnées ci-dessus :
- **(H1)** Dans la vie réelle, pour une plateforme qui vient d'être créée, cette hypothèse n'est pas vraie, mais nous prendrons un nombre de clients suffisamment grand pour négliger l'arrivée de nouveaux clients sur la plateforme.
- **(H2)** Plusieurs autres paramètres auraient pu être pris pour modéliser l'arrivée des utilisateurs (la température extérieure ou la saisonalité par exemple). Cependant, pour des raisons de simplicité du modèle et de la simulation, nous ne tiendrons compte que de la période, qui est la variable la plus pertinente et nous donne de la flexibilité sur le scénario.
- **(H3)** le prix et le goût pour un plat sont les deux variables dont dépend le plus la décision des clients puisqu'on parle de plats qui ont un prix. Il faut aussi prendre en compte la recommandation, qui permet de donner de la visibilité à certains plats.

La figure 11 illustre les différentes étapes de notre scénario.

7.2 Première étape : l'arrivée des clients

Nous commençons par simuler l'arrivée de clients sur la plateforme. Une variable aléatoire suivant une loi de **Poisson** peut être utilisée pour modéliser l'arrivée de clients. Cependant, nous ne voulons pas seulement le nombre de personnes qui arrivent sur la plateforme, nous voulons également savoir quelles personnes exactement sont arrivées. Cela motive la construction suivante :

1. Pour chaque période nous tirons aléatoirement λ selon une loi uniforme dont les deux bornes dépendent uniquement de la période (cela permet de prendre en compte un phénomène de forte affluence à certaines périodes).

2. Pour chaque client u et pour la période donnée, nous simulons son arrivée ou non par une variable X_u de Bernoulli de paramètre $\frac{\lambda}{N_u}$ où N_u est le nombre d'utilisateurs. Ces variables aléatoires seront supposées indépendantes et bien sûr identiquement distribuées.

On remarque alors que :

$$\mathbf{E}(\sum_{u=1}^{N_u} X_u) = \lambda$$

par linéarité de l'espérance.

Pour N_u grand, ce comportement est proche du tirage d'une loi de Poisson de paramètre λ car

$$\mathbf{B}(n, p_n) \xrightarrow[n\to+\infty]{loi} \mathbf{P}(\lambda)$$

avec $\lambda = np_n$ (voir [19])

7.3 Deuxième étape : le choix du plat

Conformément à l'hypothèse ($H3$), nous modélisons le choix du plat d'un consommateur en tenant compte de l'aspect financier (**prix**) et de l'attirance vers un produit (**goût**). Une notion que nous avons jugée pertinente pour tenir compte de cela est la **propension à payer**.

Soit un client u, qui a une propension à payer **p(u,i)** pour le plat i. Si le prix du plat i (qu'on notera p_i) vérifie

$$p_i > p(u, i) \tag{1}$$

alors u n'achètera pas ce produit.

Au contraire, si $p_i \leq p(u, i)$ alors l'utilisateur est susceptible de consommer i.

Que se passe-t-il si plusieurs plats vérifient $p_i \leq p(u, i)$?

Dans ce cas, l'utilisateur choisira :

$$i = argmax_j \frac{p(u, j) - p_j}{p_j}$$

où j décrit l'ensemble des plats sur la plateforme vérifiant (1), et on est assuré que la valeur du maximum du rapport est positive.

Remarque : Une autre manière de répondre à la question serait de dire que l'utilsateur choisit le premier plat dans son ranking vérifiant (1)

7.4 Profil des utilisateurs

Tout l'enjeu est donc de générer la propension à payer **p(u,i)**. Considérons d'abord le cas où il n'y a pas de classement/ranking proposé à l'utilisateur. Nous allons simuler **p(u,i)** selon une loi exponentielle (qui a l'avantage d'être positive et donne plus d'importance aux valeurs peu élevées) avec un paramètre qui, pour un utilisateur u, dépend de :

— ses moyens financiers

— son goût pour le plat i

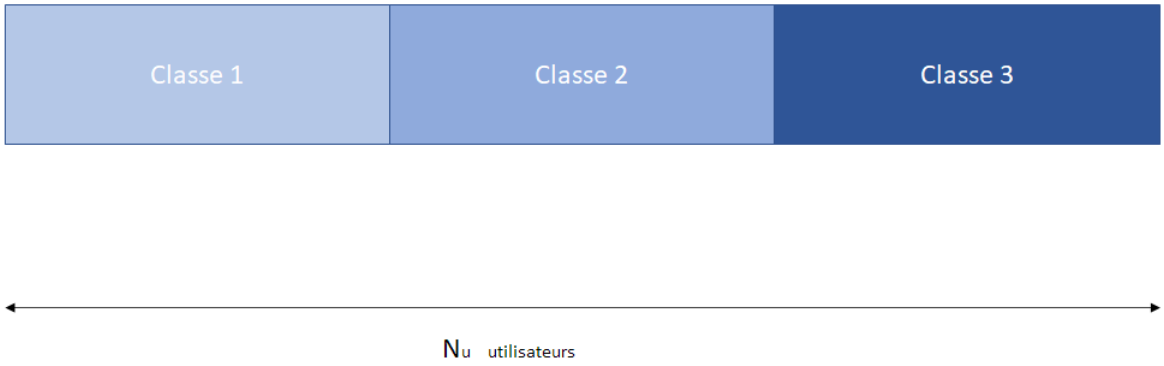

Figure 12 – Segmentation de la clientèle

Pour définir les moyens financiers des clients, nous procédons à une **segmentation**, c'est-à-dire que certains groupes de clients auront les mêmes moyens financiers. Détaillons maintenant comment la segmentation est effectuée. Celle-ci est illustrée sur la figure 12. La population est divisée en trois classes qui peuvent être de tailles différentes :

— classe 1 : pouvoir d'achat le plus faible
— classe 2 : pouvoir d'achat intermédiaire
— classe 3 : pouvoir d'achat le plus élevé

On définit alors une fonction de classe $\alpha(u)$ qui encapsule le pouvoir d'achat d'un utilisateur u donné. La fonction α a pour ensemble d'arrivée $[\![1, 2, 3]\!]$, et associe donc à un utilisateur sa classe.

On définit ensuite une fonction goût $\beta(u, i)$, qui à un utilisateur u et un plat i associe la note que donnerait u à i. Enfin, on définit $\gamma(u, i) = C\alpha(u)\beta(u, i)$ (avec C constante) qui sera le paramètre de la loi exponentielle et permettra donc de simuler la propension à payer $p(u, i)$, pour un utilisateur u et un plat i. Ainsi, il y a deux composantes : une composante qui traduit le pouvoir d'achat de u (encapsulé par $\alpha(u)$) et l'autre qui traduit l'attirance pour un plat i (encapsulé par $\beta(u, i)$).

7.5 Comportement face au ranking

Notre scénario de simulation prend également en entrée les classements des plats proposés aux utilisateurs (recommandation). Nous avons décidé de modéliser l'influence de ce ranking/classement par un facteur multiplicatif supplémentaire dans la formule décrivant $\gamma(u, i)$.

Ainsi, $\gamma(u, i) = C\alpha(u)\beta(u, i)exp(-r_u(i)) * 5$ où r_u est le classement proposé à l'utilisateur u et $r_u(i)$ est la position de i dans ce classement.

7.6 Remplissage de la note sur la plateforme

Dans notre scénario, après avoir consommé un plat i, l'utilisateur u met obligatoirement une note correspondant à ce plat sur la plateforme.

Qu'appelle-t-on note ? On supposera par la suite qu'une note est un entier compris entre 1 et 5 et traduit l'attirance du client pour un plat. Cette note ne prend donc pas en compte les facteurs externes tels que prix, heure, aléas de la commande comme la livraison. Cette assertion est

raisonnable dans le sens où la plateforme peut demander explicitement cette note à l'utilisateur, quitte à séparer les catégories de notations lors du feedback de l'utilisateur.

Dans notre simulation, la note correspond à $\beta(u, i)$

8 Présentation de la stratégie

Armés de ce scénario, nous pouvons maintenant décrire la stratégie de pricing dynamique :
— nous sommes dans un environnement dont nous avons peu de connaissances au tout début,
— nous explorons cet environnement et obtenons des données,
— nous exploitons ces données pour optimiser une certaine **métrique** (revenus, stocks...).

9 Présentation des Processus de Décision Markoviens appliqués à notre cas

9.1 MDP

L'outil mathématique que nous avons jugé adapté à cette situation est le **Processus de Décision Markovien** (que l'on abrègera par **MDP**) ([20]). L'idée est la suivante : nous avons un certain nombre d'états, et il est possible de passer d'un état à un autre. Cependant, les transitions sont aléatoires : en décidant d'effectuer une action, nous ne sommes pas certains d'arriver sur un état particulier. Chaque transition (le passage d'un état à un autre) déclenche une récompense. Notre but est de maximiser la récompense finale, qui serait par définition la somme des récompenses obtenues tout au long d'un épisode. Cependant, les transitions étant aléatoires, nous devrons parler de l'**espérance** de la récompense et non pas d'une valeur déterministe.

9.2 Description des états

Formalisons ce qui vient d'être dit. Notons S l'ensemble des états. On peut alors définir S par

$$S = \{s_{c,t} | c \in [\![stock_{min}, stock_{max}]\!], t \in [\![t_{min}, t_{max}]\!]\}$$

Nous avons **discrétisé** le stock de nourriture (compris entre deux entiers $stock_{min}$ et $stock_{max}$), et la période (comprise entre deux entiers t_{min} et t_{max}). Pour caractériser un état, il faut donc se poser la question suivante : combien de stock me reste-t-il et quelle est la période actuelle ? Ces deux données définissent un état.

Par la suite, nous utiliserons deux fonctions $c(s)$ et $t(s)$, qui à un état s associent le stock correspondant et la période correspondante.

Ainsi, si l'état s correspond à un stock de 20 à la période 12, on aura $c(s) = 20$ et $t(s) = 12$.

9.3 Description des actions

Maintenant que les états sont décrits, pour passer d'un état à un autre, il faut choisir une action. Dans notre cas, l'action correspond au fait de fixer le prix d'un plat. Soit $A(s)$ l'ensemble des actions possibles depuis l'état s. Formellement,

$$A(s) = \{p_{min} + i\frac{p_{max} - p_{min}}{N_p} | i \in [\![0, N_p]\!]\}$$

L'ensemble des prix que l'on peut fixer forme donc une subdivision régulière entre p_{min} et p_{max} de taille $N_p + 1$.

9.4 Transition

Etant donnés deux états s et s' et une action a, on notera $T(s, a, s')$ la probabilité d'arriver sur s' en partant de s et en effectuant l'action a.

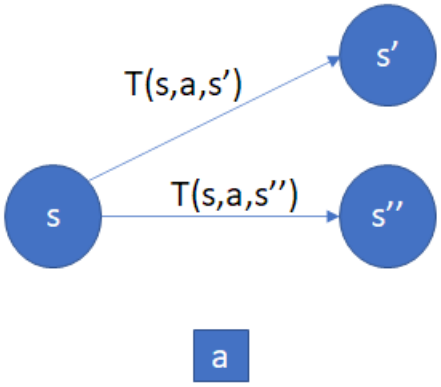

Figure 13 – Illustration d'un MDP

La figure 13 illustre la notion de MDP avec des états (s, s', s") et des transitions selon une action a.

9.5 Politique

Etant donné un état, quelle est la meilleure action à choisir pour maximiser notre objectif, et comment savoir si une action est bénéfique ?

Nous aurons besoin de ce qu'on appelle une **récompense**. Soit deux états s et s', et $a(s)$ une action possible en partant de s. On utilisera la notation $R(s, a(s), s')$, qui est un réel pour décrire cette récompense. Dans notre situation, on définit :

$$R(s_{c_1,t_1}, p, s_{c_2,t_2}) = \begin{cases} 0 & \text{si } t_1 \neq t_2 - 1 \text{ ou } c(s_{c_1,t_1}) < c(s_{c_2,t_2}) \\ p(c(s_{c_1,t_1}) - c(s_{c_2,t_2})) & \text{sinon} \end{cases} \quad (2)$$

Il s'agit simplement des recettes liées à la vente de $c(s_{c_1,t_1}) - c(s_{c_2,t_2})$ plats à un prix unitaire p.

Une politique, notée π, est une fonction dont l'ensemble de départ est S et dont l'ensemble d'arrivée est A. Intuitivement, c'est une fonction qui nous indique, pour un état donné, quelle action effectuer.

Pour illustrer la notion de politique, nous pouvons penser à :

— $\pi(s) = a$, avec a fixé. Il s'agit donc d'une politique **fixe**.

— $\pi(s) = random(A(s))$, où $A(s)$ désigne l'espace des actions possibles en partant de s. Il s'agit d'une politique complètement aléatoire.

9.6 Politique optimale et équation de Bellman

Maintenant, formalisons la notion de **gain espéré**. En appliquant une suite d'actions, la suite des états n'est pas déterministe à cause de l'aléa introduit par les transitions. C'est pourquoi il faut parler d'**espérance**. On donne une description du gain espéré optimal qui se base sur l'équation de Bellmann.

On définit $V^{\pi}(s)$ comme étant le gain espéré si l'on applique la politique π (déterministe) à chaque état, en partant de s. Notons π^{opt} la politique optimale, c'est-à-dire telle que pour tout état s et politique β, $V^{\pi^{opt}}(s) \geq V^{\beta}(s)$ L'équation de Bellman permet de formaliser cette définition.

$$V^{\pi^{opt}}(s) = \sum_{s' \in S} [R(s, \pi^{opt}(s), s') + V^{\pi^{opt}}(s')] T(s, \pi^{opt}(s), s')$$

où π^{opt} est la politique optimale.

Dans notre cas, il s'agit en fait d'une équation de récurrence. Cette équation nous permet alors de calculer les différentes valeurs de $V^{\pi^{opt}}(s)$ à condition que l'on ait des valeurs de base.

Une fois les valeurs de $V^{\pi^{opt}}(s)$ calculées, il suffit de prendre l'argmax pour trouver π^{opt}

9.7 Quelles valeurs fixer ?

Comme nous voulons dans notre étude minimiser le stock, nous allons fixer une valeur $V^{\pi^{opt}}(s)$ infinie pour les états finaux qui ne correspondent pas à un stock nul. Ainsi, cela favorise les actions qui vont vers l'état final avec un stock nul. Plus formellement, on a : $V^{\pi^{opt}}(s_{0,t_{max}}) = 0$ et $V^{\pi^{opt}}(s_{i,t_{max}}) = -\infty$ pour $i > 0$.

Dit d'une autre manière, cela favorise le comportement suivant : au lieu de maintenir un stock grand à $t_{max} - 1$ et devoir vendre à perte lors de la dernière étape, le comportement pourrait être le suivant : diminuer le stock progressivement au cours de la soirée tout en dégageant des bénéfices. En pratique, dans notre simulation nous ne pourrons pas fixer de valeur infinie car la forme de l'équation de Bellmann utilise des additions et multiplications. Pour pallier à cet inconvénient, nous nous contenterons de fixer une valeur constante négative finie dont la valeur absolue est grande.

9.8 Pourquoi une politique myope peut-elle être sous-optimale ?

Une politique myope est une politique qui, pour tout état s, vérifie :

$$\pi(s) = argmax_a \mathbf{E}(r(s, a))$$

où $r(s, a)$ est la récompense obtenue en partant de s et en faisant a. Nous donnons ici une explication intuitive sur la raison pour laquelle une politique myope peut ne pas être la politique optimale. **La caractéristique d'une politique myope est qu'à chaque étape, elle cherche à maximiser la récompense immédiate.** Elle ne prend donc pas en compte le comportement futur du système.

Dans notre cadre de travail (vente de plats au cours d'une soirée), au tout début de la soirée, il y a une demande très faible. La politique myope peut par exemple mettre des prix bas en début de soirée pour générer des revenus, mais les unités alors vendues auraient pu l'être à un prix plus élevé lors d'un moment de forte affluence.

10 Trouver la politique optimale dans un MDP

Etant donné un MDP, nous nous intéressons au problème de trouver la politique optimale. Deux cas de figure sont à considérer :

— on dispose des transitions de probabilité, et des rewards.

— on ne connaît a priori rien des transitions et des rewards.

Ces deux cas de figure possèdent des solutions différentes.

— dans le premier cas, connaissant les transitions et rewards, on peut calculer π_{opt} par l'équation de Bellman et le calcul des valeurs $V^{\pi^{opt}}(s)$ comme indiqué ci-dessus.

— dans le second cas, il faut effectuer des actions et observer le comportement du système. Là encore, deux solutions sont possibles : Soit on essaye de deviner le modèle sous-jacent au comportement du système. Cela passe par une modélisation des transitions notamment. Soit on ne définit pas de modèle et on observe seulement les réponses du système (états, récompenses) aux actions.

11 Utilisation du MDP pour résoudre notre problème : cas modèle-based

Dans cette partie, nous supposons un certain modèle pour le comportement de notre système. Nous faisons l'hypothèse suivante : **les transitions du modèle dépendent exclusivement de la fonction de demande.**

11.1 Transitions dans notre modèle

Supposons la loi de la fonction de demande connue :

$$D(p,t)$$

avec p le prix et t le temps, qui pour un prix p et une période t, donne le nombre moyen de personnes achetant le plat correspondant. Introduisons l'aléatoire en considérant f la densité d'une variable aléatoire positive d'espérance $D(p,t)$ (on note F sa fonction de répartition). On pose alors :

$$T(s,a,s') = \frac{F(c(s) - c(s') + 1) - F(c(s) - c(s'))}{Z}$$

où Z est une constante de normalisation. Autrement dit, T est reliée à l'aire sous la courbe de f entre des subdivisions entières. La quantité $c(s) - c(s')$ est forcément positive car le stock ne peut que diminuer.

Un exemple de fonction de densité que nous pourrions est celle d'une loi Gamma avec des paramètres bien définis pour avoir une espérance de $D(p,t)$.

11.2 Stratégie pour l'apprentissage de la politique optimale

Le problème s'articule autour de phases successives :

— phase 1 : on dispose de données ou bien on les génère,

— phase 2 : on estime la fonction de demande à partir de ces données,

— phase 3 : on incorpore cette fonction de demande dans un MDP, en calculant les probabilités de transition,

— phase 4 : on obtient le prix (=action) de la politique optimale.

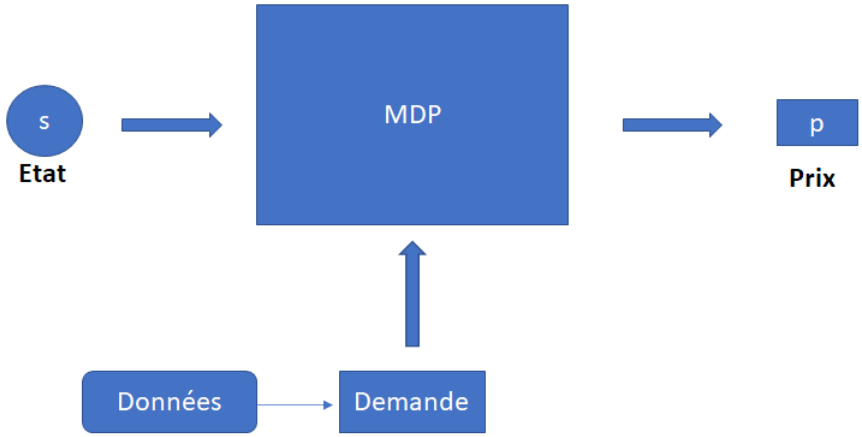

Figure 14 – Phases dans une recherche de pricing dynamique

La figure 14 illustre ces différentes phases.

Cependant, il faut préciser le MDP sur lequel nous travaillons, c'est-à-dire définir les ensembles S (états), et $A(s)$(actions) pour tout $s \in S$.

Rappelons que dans notre projet, nous avons une plateforme où sont présents plusieurs restaurants qui vendent chacun un plat. Chaque restaurant aura donc son propre prix.

Deux pistes sont possibles :

— on crée un MDP global, dont l'espace des actions possibles serait $A = (p_1, p_2, ..., p_n | p_i \in P)$ et l'espace des états $S = (s_1, ..., s_n)$. On a donc fait le "produit cartésien" des n restaurants.

— on a n MDP différents "parallèles" qui sont indépendants, chacun modélisant un restaurant et ayant chacun leur propre espace d'état et d'action.

Les MDP en parallèle présentent l'avantage d'avoir une complexité en temps linéaire en le nombre de restaurant mais traduisent moins bien que le modèle MDP global la réalité de la concurrence entre restaurants. Néanmoins l'espace des états pour le MDP global est très grand et nous ne pourrons l'implémenter. Par exemple pour 3 plats et un stock inital de 100 pour chacun des plats, on aurait alors 100^3 configurations de stocks possibles (cardinal de S), ce qui est très important.

En revanche avec des MDP en parallèle, un restaurant donné n'a pas besoin de prendre en compte les stocks des autres restaurants : il ne s'intéresse qu'à son propre stock. On a donc en quelque sorte une **"indépendance"**.

Par la suite, nous utiliserons donc n MDP en parallèle pour pouvoir expérimenter avec des nombres de restaurants raisonnables.

On fait aussi l'**hypothèse** suivante : étant donné un plat i, les prix des autres plats $j \neq i$ n'influent pas sur la demande du produit i. Cette situation est possible si les restaurants ne se font pas trop de concurrence.

Cette hypothèse cruciale nous permet de modéliser notre problème par des MDP en parallèle et indépendants.

11.3 Estimation de la demande

Nous avons donc admis un modèle pour les probabilités de transition de nos MDP. L'étape suivante consiste à trouver ces probabilités, et donc la **fonction de demande**. Plusieurs méthodes sont possibles.

• 1) Demande globale : Pour cela nous allons réaliser une régression linéaire à partir des données issues du scénario de simulation. Les variables d'entrée principales pour l'estimation de la demande sont les suivantes :

— prix du plat

— heure

De plus, on observe aussi les **notes** des utilisateurs, qui traduisent leur attirance vers un produit. Ces notes sont directement corrélées au ranking moyen du plat (on appelle ranking moyen la moyenne des rangs d'un plat dans les recommandations issues de la matrice factorisation de tous les utilisateurs) qu'on peut aussi prendre comme variable de régression.

Ainsi, on modèlise le problème suivant :

$$D(t,p,r) = g(a_1 t + a_2 p + a_3 r + b) + \epsilon$$

où ϵ est un bruit aléatoire, t désigne la période, p le prix du plat, et r le ranking moyen du plat. De plus, g désigne une fonction **connue** par la plateforme.

On préfèrera faire la régression de la façon suivante, en supposant g inversible :

$$g^{-1}(D) = a_1 t + a_2 p + a_3 r + b + \epsilon' \tag{3}$$

où ϵ' est un bruit **gaussien**. En ayant supposé ce modèle, on peut alors faire une estimation du maximum de vraissemblance (c'est-à-dire méthode des moindres carrés) et on obtient alors a_1, a_2, a_3, b.

• 2) Demande un peu moins globale : On peut se dire que le ranking n'a pas la meme influence sur les clients . On pourrait donc faire une segmentation des clients et avoir un ranking moyen r_j pour chaque classe j .

Dans ce cas , on a la même équation que (3) où a_3 est un vecteur-colonne à n_{seg} lignes où n_{seg} est le nombre de classes .

Quant au ranking moyen r , on a $r = \begin{pmatrix} r_1 \\ r_2 \\ \dots \\ r_{n_{seg}} \end{pmatrix}$.

• 3) Demande individuelle : L'idée est de modéliser le comportement de chaque utilisateur. Cette méthode semble prometteuse en théorie car elle offre un point de vue différent sur la modélisation de la demande ([21] traite ce sujet par exemple). Cependant nous ne l'avons pas implémentée pour le moment, car ce modèle est assez proche du modèle que nous utilisons comme scénario.

Pour cela une méthode assez simple serait de faire une régression logistique avec en entrée toutes les données relatives au client, le prix du plat et le plat puis prédire la probabilité d'achat du plat

par le client. Une deuxième méthode un peu plus détaillée et intéressante consisterait à diviser le problème en deux parties :

— Quelle est la probabilité $P(A_u(t))$ que le client u arrive sur la plateforme au temps t ?
— Quelle est la probabilité $P(B_{i,u}(t)/A_u(t))$ que le client u achète le plat i au temps t sachant qu'il achète sur la plateforme ? Cette partie revient à déterminer le plat que le client u achète, s'il achète sur la plateforme.

Soit $P_{i,u}(t)$ la probabilité que le client u achète le plat i à la période t . On a donc par définition des probabilités conditionnelles :

$$P_{i,u}(t) = P(B_{i,u}(t)/A_u(t)) \times P(A_u(t)$$

On aura alors besoin des quantités suivantes :

— $s_u^{(cate)}(t)$ le score de préference de la plateforme au temps t , $P(A_u(t)) = \sigma(s_u^{(cate)}(t))$ où σ est la fonction sigmoïd .
— $s_{i,u}^{(prod)}$ le score de préference du plat i par le client u au temps t , $P(B_{i,u}(t)/A(t)) = \frac{\exp(s_{i,u}^{(cate)}(t))}{\sum_{i'} \exp(s_{i',u}^{(cate)}(t))}$.

11.4 Apprentissage

Une fois que nous avons cette estimation de la demande, nous pouvons alors calculer les probabilités de transition et donc la politique optimale.

12 Model-free / Q-learning

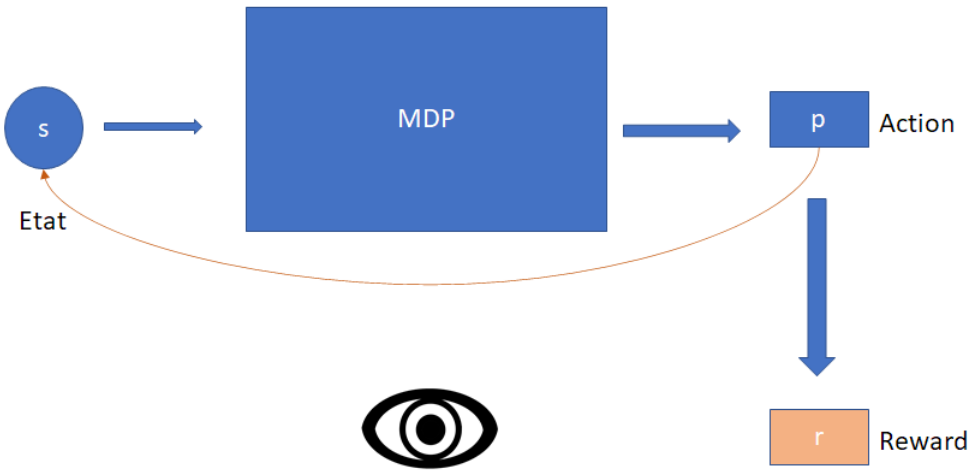

Figure 15 – Q-learning : l'agent observe le nouvel état et le reward (en orange)

Lorsque les probabilités de transition ne sont pas connues et qu'on n'a pas de modèle simple pour les décrire, on peut quand même tenter de trouver la politique optimale. Pour cela, il faut découvrir l'environnement en faisant des actions et en observant les récompenses obtenues suite à

ces actions. Une approche possible est le Q-learning [22]. **Cette méthode consiste à approcher la valeur $V^{pi_{opt}}(s,a)$ par une estimation $\hat{Q}_{opt}(s,a)$ qui se basera sur la forme de l'équation de Bellman** . Plus précisément, voici comment calculer $\hat{Q}_{opt}(s,a)$. Supposons que l'on parte de l'état s, que l'on prenne l'action a, que l'on arrive sur s' et que l'on observe la récompense r. On met alors à jour $\hat{Q}_{opt}(s,a)$ de la façon suivante :

$$\hat{Q}_{opt}(s,a) \leftarrow (1-\eta)\hat{Q}_{opt}(s,a) + \eta(r + \hat{V}_{opt}(s'))$$

avec $\hat{V}_{opt}(s') = max_{a'}\hat{Q}_{opt}(s',a')$

r est la récompense observée, a est l'action prise en partant de s, η est le learning rate.

La question est maintenant de savoir, durant cette phase d'apprentissage, quelle action effectuer pour chaque état. Expliquons comment on choisit l'action a à effectuer quand on est dans l'état s. Pour cela, il faut prendre en compte les considérations suivantes :

— on veut **explorer** l'espace des prix pour apprendre les conséquences d'une action,
— si on a une bonne connaissance globale du comportement en prenant une action, on peut essayer d'affiner cette connaissance en **exploitant**.

Ces deux assertions motivent alors le choix de l'action à entreprendre par une politique ϵ-gloutonne :

$$\pi(s) = \begin{cases} argmax_a \hat{Q}_{opt}(s,a) & \text{avec probabilité } 1-\epsilon \\ random(Action(s)) & \text{avec probabilité } \epsilon \end{cases}$$

Ainsi, ϵ décrit notre tendance à explorer l'espace des actions :
— si ϵ est proche de 0, cela signifie que nous explorons peu,
— si ϵ est proche de 1, cela signifie que nous ne faisons presque qu'explorer.

Pour résumer, l'idée d'un algorithme d'apprentissage reposant sur le Q-learning est la suivante : on essaie d'approximer V_{opt} en faisant des actions et en observant les récompenses obtenues.

Quatrième partie
Simulations

Cette partie est dédiée aux résultats concluant notre projet, obtenus à partir de simulations. Nous répondons à différentes questions et pour cela, nous exhibons les données qui proviennent de nos simulations.

13 Objectifs de la simulation

Nous allons tenter dans cette simulation de répondre aux questions suivantes :
— **(Q1)** L'estimation de la demande telle que proposée en 11.3 est-elle bonne ? Quelle fonction g prendre ?
— **(Q2)** Quelle est l'influence des valeurs initiales 9.7 sur le stock final ?
— **(Q3)** La politique de pricing dynamique couplée à une contrainte sur le stock permet-elle d'avoir des recettes supérieures à une politique de prix fixe ?
— **(Q4)** Le ranking proposé aux utilisateurs modifie-t-il la demande ?
— **(Q5)** L'algorithme Q-learning est-il plus adapté qu'un algorithme se basant sur la prédiction de la demande ?

14 Mode opératoire

Nous détaillons ici le protocole suivi pour mener nos simulations.

Tout d'abord, nous utilisons le scénario de simulation pour générer des données : c'est la **phase de training**.

Ces données consistent en :

— les prix fixés par la plateforme

— la demande pour chaque plat à chaque période suite à ces prix

— les notes des utilisateurs qui ont consommé des plats

Concernant les prix fixés, ceux-ci sont aléatoires durant cette phase dans le but d'explorer l'espace des prix : l'idée est de collecter un maximum d'informations sur les liens entre demande, prix, période et notes.

Les premières notes (initialisation) sont mises de manière aléatoire. Ensuite, les clients renseignent chaque soir les notes prévues par la matrice note du scénario et notre algorithme de SVD permet de compléter la matrice note lacunaire.

Ensuite, nous utilisons ces données pour prédire la demande à partir de la régression détaillée en 11.3.

Cette estimation de la demande nous permet de trouver les probabilités de transition dans nos MDP et donc de pouvoir faire le calcul de la politique optimale.

Nous pouvons alors utiliser cette politique optimale, et observer la demande en réponse à cette politique.

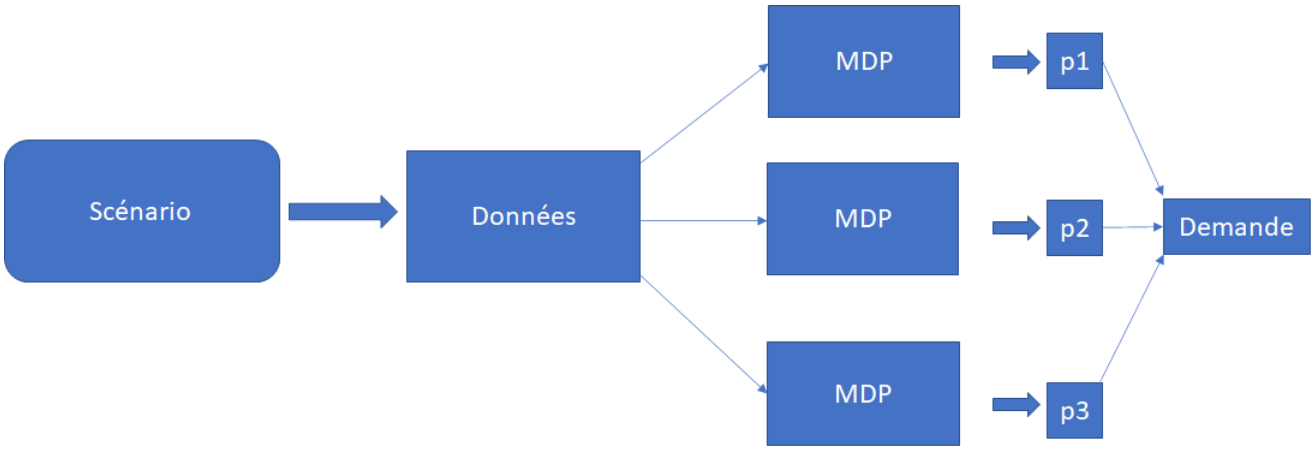

Figure 16 – Schéma représentant les différentes étapes de la simulation

La figure 16 schématise les différentes étapes : du training (à gauche) jusqu'à la phase de test à partir de la politique optimale (à droite).

Répondons maintenant aux différentes questions posées dans la section 13. Dans tout ce qui suit, nous utiliserons les notations suivantes :

— n_{plat} le nombre de plats

— n_{client} le nombre de clients

— n_{train} le nombre de jours de la **phase d'entraînement**

— n_{test} le nombre de jours durant lesquels on applique la politique optimale et on observe les résultats : **phase de test**
— n_{prix} le nombre de prix possibles
— p_{min} le prix minimum
— p_{max} le prix maximum
De plus, chaque soirée est composée de 24 périodes (des périodes de 10 minutes de $20H$ à $24H$).

Les codes de nos simulations sont sur notre GitHub [23].

15 Simulation 1

15.1 Objectif

La simulation 1 a pour but de répondre à la question **(Q1)** : **L'estimation de la demande telle que proposée en 11.3 est-elle satisfaisante ? Quelle fonction g prendre ?**

15.2 Mode opératoire

n_{plat}	n_{client}	n_{train}	n_{test}	n_{prix}	p_{min}	p_{max}
5	1000	100	15	10	1	10

Figure 17 – Paramètres utilisés lors de la simulation 1

Segmentation des clients Nous divisons la population en trois classes de pouvoir d'achat différent. Ces classes ont le même nombre d'utilisateurs. Nous générons les préférences des utilisateurs pour les plats (c'est-à-dire les notes) de façon aléatoire, mais la matrice de note est bien sûr gardée et fixée tout au long de la simulation. Nous proposons à chaque utilisateur le classement issu de la factorisation matricielle décrite et implémentée en partie 2.

Nous testons différentes fonctions de régression g : $g(x) = exp(x)$, $g(x) = x$, $g(x) = x^2$.

15.3 Résulats de la simulation

fonction	$exp(x)$	x	x^2
Score R^2	0.552	0.439	0.563

Figure 18 – Scores de la régression pour les différentes fonctions testées, moyennés sur les 5 derniers jours de test

Fonctions	ordonnée à l'origine	temps	prix	ranking moyen
$exp(x)$	3.307	-0.115	-0.246	-0.115
x	18.483	-0.706	-1.610	-0.652
x^2	4.761	-0.160	-0.361	-0.179

Figure 19 – Comparaison des coefficients de régression pour différentes fonctions de régression.

Tout d'abord, le score qui est mentionné dans la figure 18 est le score R^2, aussi appelé coefficient de détermination. Plus ce score est proche de 1, meilleure est la régression ([24]). On remarque que les scores de régression pour l'exponentielle et la fonction carré sont les plus élevés. Cependant, un facteur nous fait privilégier la fonction exponentielle : avec la fonction carré, pour un prix qui tend vers l'infini, on aurait une demande tendant également vers l'infini, ce qui est absurde et ne correspond pas à un comportement réel.

Le choix de l'exponentielle comme fonction semble donc être naturel, car au contraire, pour un prix tendant vers l'infini, et par négativité du coefficient correspondant au prix, on aurait une demande qui tend vers 0.

Discutons maintenant du poids relatif des différentes variables, donné par les coefficients de la régression. On remarque que l'influence du prix est deux fois moindre par rapport à l'influence du ranking moyen. Cette influence a pour origine le lien entre propension à payer, classe, goût et ranking modélisé dans le scénario de simulation.

Ainsi nous prenons par la suite $g(x) = exp(x)$. A noter qu'avec 2 plats, on obtient un score de régression de l'ordre de 0.77, donc plus proche de 1.

16 Simulation 2

16.1 Objectif

La simulation 2 a pour but de répondre à la question **(Q2)** : **quelle est l'influence des valeurs initiales 9.7 sur le stock final ?** Dans la sous-section 9.7, nous avons vu qu'il fallait fixer des valeurs initiales pour $V^\pi(s_{i,t_{max}})$ pour $i \geq 0$. Nous voulons maintenant étudier l'influence de ces valeurs initiales sur le stock final, c'est-à-dire le stock en fin de soirée.

16.2 Mode opératoire

n_{plat}	n_{client}	n_{train}	n_{test}	n_{prix}	p_{min}	p_{max}
2	1000	100	15	10	1	10

Figure 20 – Paramètres utilisés lors de la simulation 2

Segmentation des clients Nous divisons la population en trois classes de pouvoir d'achat différent. Ces classes ont le même nombre d'utilisateurs. Nous générons les préférences des utilsateurs pour les plats (c'est-à-dire les notes) de façon aléatoire. Néanmoins nous conservons les notes

renseignées d'un jour à l'autre lors de la simulation. Nous proposons à chaque utilisateur le classement issu de la factorisation matricielle décrite en partie 2.

Nous comparons deux initialisations :
1) On fixe les valeurs initiales $V^{\pi}(s_{i,t_{max}}) = 0$ pour $i > 0$.
2) On fixe les valeurs initiales $V^{\pi}(s_{i,t_{max}}) = -5000$ pour $i > 0$.

16.3 Résultats de la simulation

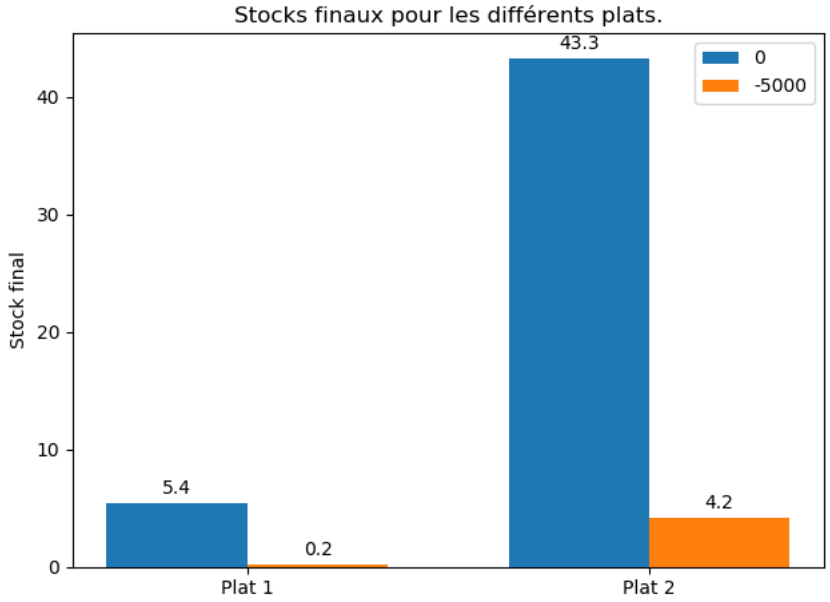

Figure 21 – Comparaison des stocks finaux en fonction de l'initialisation choisie.

Expliquons d'abord ce graphe : dans la simulation correspondant à '0' ,les valeurs $V^{\pi}(s_{i,t_{max}})$ sont fixées à 0 et dans la simulation correspondant à '-5000', les valeurs de $V^{\pi}(s_{i,t_{max}})$ ont été fixées à -5000.

Les valeurs représentant les hauteurs des barres sont en fait la moyenne des stocks finaux sur les jours de test (donc 15 jours) pour chaque plat.

Nous remarquons qu'en fixant des valeurs négatives de grande valeurs absolues (comme -5000), nous encourageons le phénomène suivant : **le stock est très faible en fin de soirée**. Cela montre donc toute la puissance des MDP et de la politique optimale. Celle-ci, décrite par l'équation de Bellman, prend en compte le résultat des actions futures.

En pratique, quelle est la différence entre la politique de la simulation à initialisation 0 et la politique de la simulation à initialisation -5000 ? Nous avons remarqué que pour cette dernière, le prix de 1 était appliqué en fin de soirée. Ceci est bien sûr dans le but d'augmenter la demande et donc de diminuer les stocks.

Ainsi, nous avons montré qu'il était possible de répondre à une des problématiques majeures de notre projet : **réduire le gaspillage alimentaire en limitant les stocks finaux, c'est-à-dire en fin de soirée.** Cependant, cela augmente-t-il les recettes du restaurateur ?

17 Simulation 3

17.1 Objectif

La simulation 3 a pour but de répondre à la question **(Q3) : La politique de pricing dynamique couplée à une contrainte sur le stock permet-elle d'avoir des recettes supérieures à une politique de prix fixe lors d'une soirée ?**

17.2 Mode opératoire

n_{plat}	n_{client}	n_{train}	n_{test}	n_{prix}	p_{min}	p_{max}
5	1000	100	15	10	1	10

Figure 22 – Paramètres utilisés lors de la simulation 3

Nous nous intéressons aux politiques de prix fixe, définies comme suit. On dit qu'une politique est $k-$fixe si elle consiste à fixer les prix de tous les plats de la plateforme au même prix k, durant toute la soirée. Ces politiques ne constituent pas l'ensemble de toutes les politiques de prix fixe possible. En effet, on pourrait mettre des prix fixes différents pour chaque plat. Cependant, le nombre de telles combinaisons devient bien trop grand dès que l'on augmente le nombre de plats. C'est pourquoi nous avons décidé de nous restreindre aux politiques $k-$fixes.

Durant notre simulation, nous testons toutes les politiques $k-$fixes pour k décrivant l'ensemble des prix possibles.

17.3 Résultats de la simulation

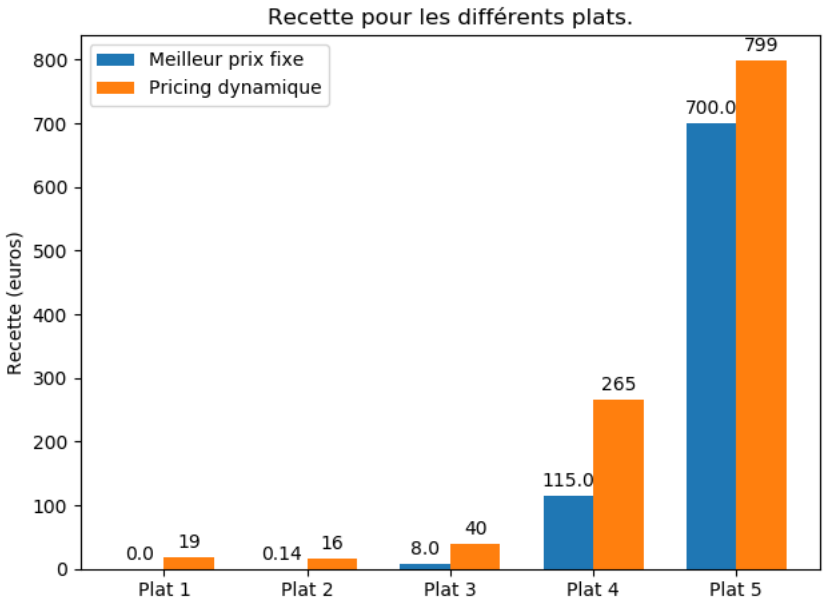

Figure 23 – Comparaison des recettes pour les différentes politiques de pricing

Lors de la simulation, nous avons remarqué que la meilleure stratégie de pricing $k-$fixe était la politique $7-$**fixe**. Nous comparons alors cette politique de prix fixe avec la politique de pricing dynamique suggérée par nos MDP lors de la phase de test. Les résultats sont illustrés sur la figure 23.

Nous remarquons qu'avec la politique de pricing dynamique, nous augmentons de **38%** les recettes sur l'ensemble des 5 plats.

Cela montre donc un résultat très important dans le cadre de notre projet : **une politique de pricing dynamique permet de concilier la réduction des stocks et l'obtention de revenus rivalisant avec des politiques de prix fixe.**

Remarque : Au cours des jours de training, la matrice note se remplit peu à peu et notre algorithme de SVD prédit de manière plus exacte les notes.

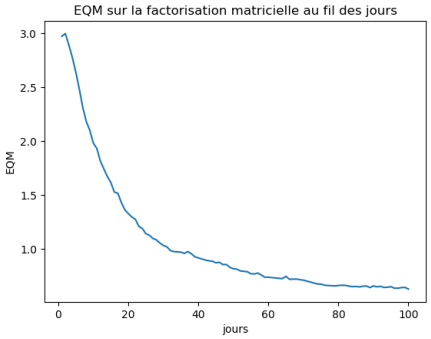

Figure 24 – Erreur Quadratique Moyenne lors de la factorisation matricielle pendant la phase d'entraînement.

18 Simulation 4

18.1 Objectif

La simulation 4 a pour but de répondre à la question **(Q4)** : **quel est l'impact du ranking proposé aux utilisateurs sur la demande ?**

n_{plat}	n_{client}	n_{train}	n_{test}	n_{prix}	p_{min}	p_{max}
2	1000	100	15	10	1	10

Figure 25 – Paramètres utilisés lors de la simulation 4

On lance deux simulations :
— lors de la première, le ranking affiché pour chaque utilisateur est toujours [plat1, plat2].
— lors de la seconde, le ranking affiché pour chaque utilisateur est toujours [plat2, plat1]

18.2 Résultats de la simulation

A l'issue de la première simulation, le stock moyen (sur 5 soirées de test) restant à l'issue d'une soirée pour le plat 1 est de 0.2 et le stock moyen restant à l'issue d'une soirée pour le plat 2 est de

20.4.

A l'issue de la deuxième simulation, le stock moyen (sur 5 soirées de test) restant à l'issue d'une soirée pour le plat 1 est de 19.2 et le stock moyen restant à l'issue d'une soirée pour le plat 2 est de 0.4.

On remarque donc une corrélation entre ranking proposé aux utilisateurs et stocks finaux :**un plat classé haut dans les rankings voit sa demande augmenter**.

19 Simulation 5

19.1 Objectif

Dans cette simulation, nous voulions comparer l'algorithme du Q-learning avec l'algorithme qui se base sur la prédiction de la demande.

19.2 Résultat

Nous avons observé des revenus bien inférieurs dans le cas du Q-learning. De plus, la courbe représentant les revenus par rapport au nombre d'épisodes n'avait pas du tout une allure croissante. **Ainsi, l'algorithme Q-learning n'est pas adapté à notre scénario de simulation.**

20 Limitations

Nous nous sommes appuyés sur un **scénario de simulation** pour pouvoir générer des données. En utilisant ce scénario, nous avons mis en avant plusieurs avantages de la politique de pricing dynamique. Cependant, cette démarche n'est pas suffisante pour conclure à l'optimalité de la politique de pricing dynamique. En effet, il y a le risque que le modèle de régression soit trop proche du modèle qui nous a permis de générer les données : dans ce cas, les résultats sont peu interprétables. C'est pourquoi nous avons fait le choix d'un modèle de régression assez général sur la demande "globale". Idéalement, nous devons mettre à l'épreuve notre politique avec des données issues de la vie réelle. Malheuresemeent, nous n'avons pas trouvé de dataset libre d'accès réunissant tous les éléments dont nous avions besoin. La seule solution, hormis la création d'un scénario de simulation est alors de déployer notre système de pricing et de recommandation dans la vie réelle.

Cinquième partie
Conclusion

Ce projet s'est articulé autour de différents volets. Tous ces volets se sont retrouvés dans nos simulations finales et ont permis de mettre en évidence l'idée suivante : **une politique de pricing dynamique couplée à un système de recommandation permettent de dégager des recettes à la hauteur d'une politique de prix fixe tout en assurant des stocks finaux faibles**. Cela permet donc à la plateforme de combattre le gaspillage alimentaire, sans pour autant négliger l'aspect financier.

Plusieurs pistes peuvent être explorées afin de prolonger ce travail :
• Nous pouvons tout d'abord nous intéresser à un modèle de demande différent de celui utilisé en 11.3. Il serait alors intéressant de comparer les performances sur le même scénario de simulation.

• Il serait également intéressant de pouvoir tester notre modèle sur un jeu de données réel, et donc de le déployer sur une plateforme existante afin d'évaluer la pertinence de notre politique. De plus, pour des raisons liées au temps d'implémentation nous n'avons pas pu tester notre simulation pour plus que 100 produits.

• Nous avons considéré des ratings explicites. Cependant, dans la pratique très peu d'utilisateurs renseignent des notes. Les plateformes existantes se focalisent presque exlusivement sur des ratings implicites. Cette approche est différente, néanmoins les outils de prédiction restent les mêmes.

• Nous n'avons pas pris en compte la concurence entre les restaurants au niveau du MDP. Comparer nos résultats avec ceux obtenus avec une approche MDP global aurait été intéressant.

Ce projet nous a permis d'élargir notre champ de connaissances concernant les outils mathématiques. En effet, la factorisation matricielle par exemple n'est pas une technique courante en dehors du cadre de la recommandation.

Manipuler des outils comme les MDP dans un cadre qui peut s'inscrire dans la vie réelle a été très enrichissant et motivant.

De plus le fait d'implémenter ces algorithmes de bout en bout sans aucune assurance de résultat s'est révélé être extrêmement formateur.

Sixième partie
ANNEXE

A Implémentations personnelles

Cette annexe présente brièvement l'implémentation que nous avons faite sans aucune librairie de Python autre que Numpy. Nous cherchons à minimiser $\|M - UV\|^2$ où M désigne la matrice note.

Nous avons remarqué qu'en fixant U ou V cette expression de perte devient convexe et ainsi nous avons utilisé une méthode de descente de gradient pour minimiser cette perte. Pour l'initialisation, nous avons juste fixé tous les coefficients de U et V à $0,1$. Nous avons alors à chaque itération fixé U, puis V, et fait une descente de gradient pour approcher chaque coefficient renseigné de la matrice M. Nous avons pris ces coefficients dans un ordre aléatoire différent à chaque itération afin d'éviter qu'un coefficient soit favorisé si "sa" descente de gradient est effectuée en dernier. De plus, à chaque itération, nous avons fait décroître le taux d'apprentissage. Rappelons que nous avons tenté beaucoup de méthodes différentes pour réaliser cette factorisation et que l'idée de faire décroître le taux d'apprentissage d'étape en étape par exemple a été retenu car en tentant cet artifice nous avons remarqué une diminution importante de la perte.
Cette première phase nous a permis d'obtenir des Erreurs Quadratiques Moyennes très faibles lors de la factorisation de matrices M aléatoires. Néanmoins nous n'avons jusque là "que" factoriser une matrice mais nous n'avons pas régularisé nos descentes de gradient. Ainsi la prédiction de notes était très mauvaise car nous faisions du sur-apprentissage. La dernière étape a donc été de régulariser nos descentes de gradient et d'étudier l'influence des paramètres sur les performances (facteurs latents, taux d'apprentissage décroissant ...). L'algorithme 1 présente le pseudo-code de l'algorithme que nous avons mis en place.

Input : M matrice note de taille $n \times m$, η (taux d'apprentissage), r (régularisation), ϵ, l (facteurs latents), iter
Initialisation : U et V deux matrices de tailles $n \times l$ et $l \times m$ remplies de $0,1$.
pour *nb allant de 1 à iter* **faire**
  $\eta = \eta \times 0.9$
  pour *i,j dans* $[1,n] \times [1,m]$ **faire**
    $a = M_{i,j}$
    $u = U_i$, $v = V_j^t$
    tant que | *gradient selon* U | $\geq \epsilon$ **faire**
      descente de gradient pour minimiser $\|a - u \cdot v\|^2 + r \|u\|^2$ à v fixé
    fin
    tant que | *gradient selon* V | $\geq \epsilon$ **faire**
      descente de gradient pour minimiser $\|a - u \cdot v\|^2 + r \|v\|^2$ à u fixé
    fin
  fin
fin

Algorithme 1 : Pseudo code de notre première implémentation d'un algorithme de recommandation.

Pour tester notre algorithme, nous avons commencé par utiliser une matrice notes "cas limite", facile à prédire. La figure 26 présente l'évaluation de notre algorithme sur la factorisation de cette matrice. (GitHub [25])

$$\begin{pmatrix} 5 & \cdots & 5 & 1 & \cdots & 1 \\ \vdots & & \vdots & \vdots & & \vdots \\ 5 & \cdots & 5 & 1 & \cdots & 1 \\ 1 & \cdots & 1 & 5 & \cdots & 5 \\ \vdots & & \vdots & \vdots & & \vdots \\ 1 & \cdots & 1 & 5 & \cdots & 5 \end{pmatrix}$$

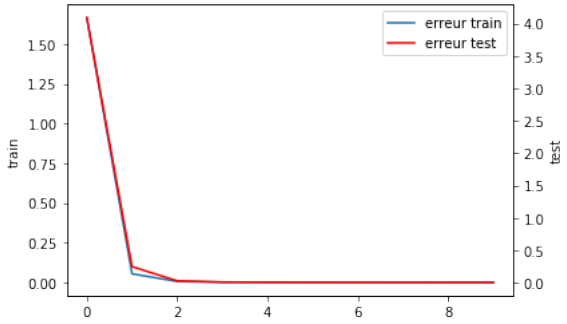

Figure 26 – Perte Quadratique Moyenne sur les ensemble d'entrainement et de test avec latent=5, r=2, $\epsilon = 0.01$ et $\eta = 0.1$

Les codes de notre algorithme, et de nombreuses fonctions auxiliaires (calcul de perte régularisée, calcul de perte non régularisée, calcul de perte plus malin en ne gardant que certaines lignes/colonnes, réorganisation de dataset...) sont sur Github. Ces codes sont assez long car il nous a fallu tout implémenter par nous mêmes. On observe que notre algorithme fonctionne sur ce cas limite : la perte quadratique moyenne tend vers 0 pour l'échantillon d'entrainement comme pour celui de test au bout de très peu d'étapes.
Nous allons maintenant ajouter un peu d'aléa à cette matrice en mettant une note aléatoire entre 3 et 5 à la place des 5 (bonne note) et entre 1 et 2 à la place des 1 (mauvaise note). On remarque qu'en prenant les mêmes paramètres que pour la matrice précédente, l'erreur sur le test devient très importante : l'algorithme a plus de mal à généraliser. Nous avons alors relancé l'algorithme en augmentant le facteur de régularisation. Les figures 27 et 28 présentent les évolutions de la perte en fonction des étapes pour différents facteurs de régularisation.(GitHub [25])

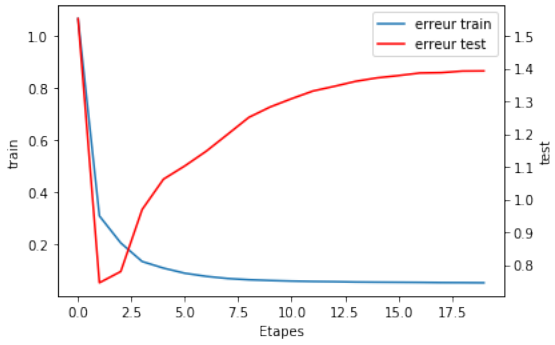

Figure 27 – Perte Quadratique Moyenne sur les ensembles d'entraînement et de test avec latent=5, r=2, $\epsilon = 0.01$ et $\eta = 0.1$

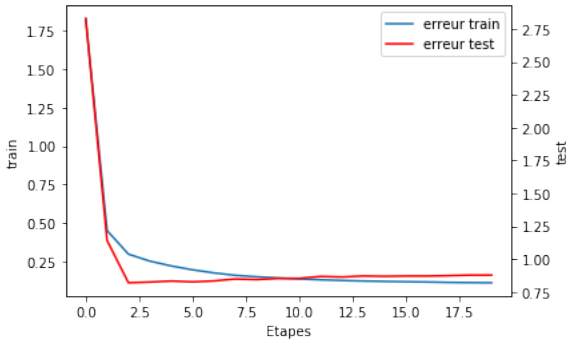

Figure 28 – Perte Quadratique Moyenne sur les ensembles d'entraînement et de test avec latent=5, r=20, $\epsilon = 0.01$ et $\eta = 0.1$

On remarque qu'en prenant un facteur de régularisation 10 fois plus élevé, la prédiction des notes est bien meilleure car l'erreur sur l'ensemble de test est à peine plus élevée que celle sur l'ensemble d'entraînement.

Pour terminer l'évaluation de notre algorithme, nous l'avons testé sur un dataset réel, le plus petit, celui avec les plats. Le programme met une vingtaine de minutes à tourner, et nous obtenons ces résultats.(GitHub [25])

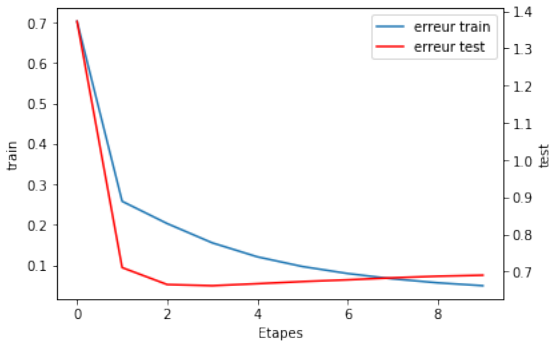

Figure 29 – Performances de notre algorithme sur le dataset des plats (138 utilisateurs, 130 plats)

On remarque que l'Erreur Quadratique Moyenne sur l'échantillon de test est de l'ordre de $0,7$ après une dizaine d'étapes. Pour ce dataset (le seul où les notes sont entre 1 et 3 seulement), l'EQM de l'algorithme de SVD est de l'ordre de $0,4$, soit presque diminuée de moitié. De plus, notre algorithme était bien trop long sur ce "petit" dataset (environ 2 à 3 minutes contre moins d'une seconde pour SVD), c'est pourquoi nous avons ensuite privilégié l'utilisation de librairies spécialisées pour l'apprentissage comme Sklearn et Keras.

Bibliographie

[1] Gerald Ninaus Florian Reinfrank Stefan Reiterer Alexander Felfernig, Michael Jeran and Martin Stettinger. Basic approaches in recommendation systems, 2014.

[2] Hug, Nicolas and Prade, Henri and Richard, Gilles Raisonnement analogique pour la recommandation : premières expérimentations. (2015) In : 9es Journées d'Intelligence Artificielle Fondamentale (IAF 2015) in Plate-forme Intelligence Artificielle 2015, 29 June 2015 - 3 July 2015 (Rennes, France).

[3] Julien Delporte. Factorisation Matricielle, Application à la Recommandation Personnalisée de Préférences. Machine Learning [stat.ML]. INSA de Rouen, 2014. Français.

[4] Wikipedia (version anglaise). https://en.wikipedia.org/wiki/Precision_and_recall.

[5] Robert Bell Yehuda Koren and Chris Volinsky. Matrix Factorization Techniques For Recommender Systems, Published by the IEEE Computer Society, 2009.

[6] Soumya Ghosh. https://medium.com/@connectwithghosh/simple-matrix-factorization-example-on-the-movielens-dataset-using-pyspark-9b7e3f567536.

[7] Zygmunt Zajac. https://www.kaggle.com/zygmunt/goodbooks-10k#ratings.csv.

[8] UCI Machine Learning. https://www.kaggle.com/uciml/restaurant-data-with-consumer-ratings.

[9] grouplens. https://www.kaggle.com/prajitdatta/movielens-100k-dataset#u.data.

[10] grouplens. https://www.kaggle.com/prajitdatta/movielens-100k-dataset#u.data.

[11] Zygmunt Zajac. https://www.kaggle.com/zygmunt/goodbooks-10k.

[12] Sklearn. https://pypi.org/project/scikit-surprise/1.0.1/.

[13] GitHub. https://github.com/PSC-MAP05/factorisation_matricielle/blob/master/keras.ipynb.

[14] GitHub. https://github.com/PSC-MAP05/factorisation_matricielle/blob/master/Precison_recall.ipynb.

[15] GitHub. https://github.com/PSC-MAP05/factorisation_matricielle/blob/master/comparaison_eqm.ipynb.

[16] GitHub. https://github.com/PSC-MAP05/factorisation_matricielle/blob/master/comparaison_algo_constant.ipynb.

[17] GitHub. https://github.com/PSC-MAP05/factorisation_matricielle/blob/master/gridsearch_1M.ipynb.

[18] Ruben van de Geer, Arnoud V. den Boer, Christopher Bayliss, Christine Currie, Andria Ellina, Malte Esders, Alwin Haensel, Xiao Lei, Kyle D. S. Maclean, Antonio Martinez-Sykora, Asbjørn Nilsen Riseth, Fredrik Ødegaard, and Simos Zachariades. Dynamic Pricing and Learning with Competition : Insights from the Dynamic Pricing Challenge at the 2017 INFORMS RM &amp ; Pricing Conference. *arXiv e-prints*, page arXiv :1804.03219, Mar 2018.

[19] BibMath. http://www.bibmath.net/dico/index.php?action=affiche&quoi=./a/approxpoisson.html.

[20] Rice University. https://www.cs.rice.edu/~vardi/dag01/givan1.pdf.

[21] M Wan. http://papers.www2017.com.au.s3-website-ap-southeast-2.amazonaws.com/proceedings/p1103.pdf.

[22] cse.unsw. https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html.

[23] GitHub. https://github.com/PSC-MAP05/SimulationFinale.

[24] Wikipedia. https://en.wikipedia.org/wiki/Coefficient_of_determination.

[25] GitHub. https://github.com/PSC-MAP05/factorisation_matricielle/blob/master/notre_implementation.ipynb.