CentraleSupélec

université PARIS-SACLAY

Algorithmes de recommandation

Rapport final

Pôle IA

Projet P10.04

Elias Al Bouzidi
Eliott Binard
Rémy Court
Abderrahmane Dkour
Laure Garreau

4 juin 2023

Chapitre 1

Introduction

1.1 Présentation du client

Le client est un élève en deuxième année de *CentraleSupélec*, Christophe Hache. Avec une amie étudiante à l'école des Ponts, il a fondé sa start-up, *IzyLeaf*. L'idée : rendre simple et ludique le choix d'un véhicule d'occasion. Le client a en effet remarqué à quel point il était pénible de rechercher un nouveau véhicule : multiplicité des sites, annonces non fidèles voire erronnées... Pour y remédier, il a alors eu l'idée d'une application dans le style "Tinder" : après avoir renseigné quelques informations, l'utilisateur se voit présenter des fiches descriptives de véhicules. Si il "aime" le véhicule, celui ci est enregistré pour être consulté plus facilement. Cette nouvelle approche modernise la recherche de véhicule en ligne et supprime l'aspect rébarbatif : pas besoin d'éplucher des centaines de sites différents, tous les véhicules sont réunis sur une seule et même plateforme.

Tout l'intérêt de l'application est de proposer des véhicules ciblés en fonction de ceux "likés" par l'utilisateur. Au fur et à mesure que l'utilisateur passe du temps sur l'application, ses préférences vont s'affiner et l'application proposera des suggestions de plus en plus proches de ses goûts. On a donc affaire un algorithme de recommandation. Ce type d'algorithme est omniprésent de nos jours (Amazon, Spotify, YouTube...) et permet à l'utilisateur de gagner un temps précieux. Le client nous a donc contacté pour mettre en place et développer des algorithmes de recommandation pertinents.

1.2 Problème posé et bénéfice attendu

Le principal problème observé est la diversité des sites et des annonces proposés sur Internet pour les véhicules d'occasion. D'où l'idée du client de tous les regrouper sur une application, ce qui permet d'appliquer derrière des algorithmes de recommandation. Un des premiers objectifs de l'application est donc de constituer une base de données conséquentes et mise à jour régulièrement pour éviter les problèmes rencontrés sur les sites traditionnels. Cet objectif incombe au client, qui nous demande de réaliser le deuxième objectif : à partir de cette base de données et des likes d'un utilisateur, proposer des véhicules de plus en plus pertinents afin de minimiser le temps passé à rechercher un véhicule.

L'interface graphique de l'application, la database ainsi que les interactions de base ont déjà été implémentées. Notre mission ne concerne donc que le *back-end*, à savoir les algorithmes de recommandation. Le client, ayant déjà travaillé dans ce pôle projet l'année dernière, nous a donné quelques conseils sur comment aborder le problème, des pistes de recherches mais également trois contraintes principales :

— Qu'on utilise des réseaux de neurones. Nous traiterons néanmoins des nombreuses techniques usuelles des algorithmes de recommandation : méthodes de similarité, factorisation des matrices, ... Cela permettra d'avoir un recul nécessaire sur la réalisation de notre tâche et de mener à bien l'implémentation du réseau de neurones.

— Des contraintes sur la performance / le coût de calcul de l'algorithme. En prenant plusieurs facteurs en considération, tels que le format de l'application (une séquence de swipes), notre algorithme de recommandation ne doit pas imposer beaucoup de latence entre l'interaction et l'affichage du swipe suivant (car cela peut, par exemple, nuire à la durée d'attention de l'utilisateur). D'autre part, comme il s'agit d'une startup, il est inutile de proposer des algorithmes qui nécessitent une grande infrastructure pour être maintenus et développés à grande échelle. On essayera donc de réduire au maximum le temps d'exécution de nos algorithmes, tout en conservant des résultats cohérents.

— Des limitations sur la taille des données que nous pouvons utiliser. En effet, la startup étant assez récente, le client n'a pas pu réunir un nombre important de data. Celles ci ont été obtenus à travers une phase *bêta* de l'application réunissant une centaine de participants. Cela nous oblige donc à mettre certaines technologies de côté et de nous concentrer sur des algorithmes moins complexes et appliquables à notre échelle.

1.3 Dimension éthique du projet

Les algorithmes de recommandation ont récemment fait l'objet d'un débat animé, d'abord avec la poursuite judiciaire de Gonzalez et l'audience ultérieure devant la Cour suprême concernant l'entité responsable du contenu recommandé par les entreprises de technologie, puis avec l'influence considérable de TikTok sur les enfants, en grande partie en raison de la nature addictive du contenu recommandé. (On peut soutenir que l'algorithme de recommandation est, en soi, un contenu). On ne peut donc nier le côté éthique des algorithmes de recommandation, et il semble donc normal de s'interroger sur la nature de notre projet.

Néanmoins, pour IzyLeaf, l'algorithme de recommandation (et les swipes inspirées de Tinder) est un moteur de recherche glorifié. Il n'est pas là pour maximiser le temps d'utilisation. En fait, l'objectif est de minimiser le nombre de glissements nécessaires pour que l'utilisateur trouve la voiture qui convient le mieux. Le but ici n'est donc pas de maximiser le temps de fonctionnement de l'application ou de créer un état de dépendance chez l'utilisateur, mais simplement à accomplir une tâche donnée. En tant que tel, nous croyons que l'algorithme que nous proposons sera éthique sur le plan conceptuel.

Les données utilisées pour entraîner le modèle sont également éthiquement sourcées. Les gens ont participé, en pleine connaissance de l'objectif final, à un petit défi proposé par le créateur de l'application. De cette campagne, les seules données récupérés sont les "likes" des utilisateurs. On ne connaît donc que leurs préférences de véhicules. Les données sont donc peu sensibles et peu sujette à un "leak". Enfin, elles ont été anonymisées de telle sorte que nous n'avons vraiment pas beaucoup de connaissances sur les personnes qui ont participé à l'alpha fermée.

Chapitre 2

Etat de l'art

2.1 Introduction

A la demande du client, on se servira comme base de l'état de l'art celui fait par ce dernier l'année précédente que l'on étoffera bien sûr, notamment dans les parties qu'il nous a lui même indiqué.

Les algorithmes de recommandation ont pour objectif de fournir de manière optimale à un utilisateur des informations ou des ressources qui sont susceptibles de lui plaire ou de répondre à la demande qu'il a émise. Cela permet à l'utilisateur un gain de temps dû au fait que sa recherche est effectuée par un algorithme, mais également de trouver des réponses auxquelles il n'avait pas pensé.

De nos jours, ces systèmes sont présents un peu partout sur le Web : sur les sites de streaming comme Netflix, afin de proposer à l'utilisateur les films/séries les plus susceptibles de lui plaire ; sur les sites d'e-commerce comme Amazon, afin de proposer à l'acheteur des produits qui correspondent au mieux à ses recherches ; et encore dans bien d'autres domaines.

Différents types de systèmes de recommandation ont vu le jour. Ces derniers se distinguent par les facteurs qu'ils utilisent pour réaliser leurs recommandations : connaissance de l'utilisateur, connaissance des objets, lien et similitude d'un utilisateur avec d'autres, lien et similitude d'un objet avec d'autres, historique d'un utilisateur. Il existe trois grandes classes d'algorithmes de recommandation : les filtrages basés sur le contenu, les filtrages collaboratifs, et les filtrages hybrides.

Dans la suite, nous considérons un ensemble $\mathcal{U}$ d'utilisateurs et un ensemble $\mathcal{I}$ d'objets. Pour tout objet $i \in \mathcal{I}$, on note $\mathcal{U}_i$ l'ensemble des utilisateurs ayant noté l'objet i, et pour tout utilisateur x, on note $\mathcal{I}_x$ l'ensemble des objets notés par x.

Le but d'un algorithme de recommandation sera ici d'estimer la notation d'un utilisateur x sur un objet i. Cela permettra par la suite de proposer à l'utilisateur les objets sur lesquels sa notation estimée par l'algorithme sera la plus élevée.

2.2 Filtrage collaboratif User-Based

Le filtrage collaboratif repose sur l'idée suivante : si des individus ont eu des goûts et comportements similaires dans le passé, ils sont plus enclin à partager les mêmes goûts dans le futur. Ainsi, il suffit de trouver, pour un utilisateur x donné, un ensemble N d'utilisateurs partageant les goûts de x, afin de recommander à x les évènements aimés par les utilisateurs appartenant à N.

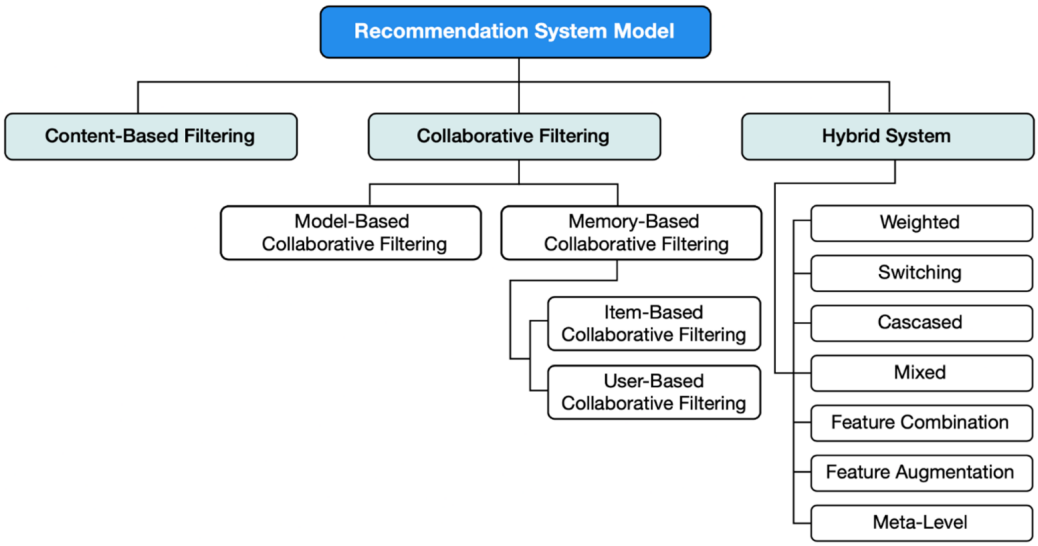


FIGURE 2.1 – Classification des systèmes de recommandation

2.2.1 Mesures de similarité

Différentes mesures permettent de calculer la similarité entre les goûts de deux utilisateurs. A chaque utilisateur x est associé un vecteur de notations r_x représentant les notes attribuées par x aux différents objets. La notation de x sur un objet i sera alors notée $r_{x,i}$. Nous considérons dans la suite deux utilisateurs x et y, et leurs vecteurs de notations r_x et r_y.

2.2.1.1 Mesure de Jaccard

Soient A et B deux ensembles tels que $|A \cup B| \neq 0$. La mesure de similarité de Jaccard entre ces deux ensembles est définie par :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

On définit alors la mesure de similarité de Jaccard entre deux utilisateurs x et y par :

$$sim(x,y) = J(\mathcal{I}_x, \mathcal{I}_y) = \frac{|\mathcal{I}_x \cap \mathcal{I}_y|}{|\mathcal{I}_x \cup \mathcal{I}_y|} \tag{2.2}$$

Cette mesure de similarité est simple mais ne prend pas en compte les valeurs des notes attribuées à chaque objet.

2.2.1.2 Mesure du cosinus

En notant $||.||$ la norme Euclidienne, la mesure de similarité du cosinus entre x et y est définie par le cosinus de l'angle entre r_x et r_y :

$$sim(x,y) = cos(r_x, r_y) = \frac{\langle r_x \mid r_y \rangle}{||r_x||\ ||r_y||} \tag{2.3}$$

Cette mesure renvoie une valeur entre -1 (pour des vecteurs de sens opposés) et 1 (pour des vecteurs de même sens). Dans le cas de vecteurs de notations, elle prend en compte la valeur des notes attribuées à chaque objet, et est donc plus précise que la mesure de Jaccard.

2.2.1.3 Coefficient de corrélation de Pearson

Notons $S_{xy} = \mathcal{I}_x \cap \mathcal{I}_y$ l'ensemble des objets notés à la fois par x et par y, et $\bar{r_x} = \frac{1}{|\mathcal{I}_x|} \sum_{i \in \mathcal{I}_x} r_{x,i}$ la valeur moyenne de notation de l'utilisateur x. Le coefficient de corrélation de Pearson est défini par :

$$sim(x,y) = \frac{\sum_{i \in S_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum_{i \in S_{xy}} (r_{x,i} - \bar{r_x})^2} \sqrt{\sum_{i \in S_{xy}} (r_{y,i} - \bar{r_y})^2}} \tag{2.4}$$

Cette mesure permet de centrer les notes, neutralisant l'influence d'une notation moyenne trop élevée ou trop basse de certains utilisateurs. En revanche, elle ne prend en compte que les objets notés par les deux utilisateurs, ce qui peut poser problème si la matrice de notations est trop creuse.

2.2.1.4 Mesures Jaccard-pondérées

Pour prendre en compte le nombre d'objets notés à la fois par x et par y, il est possible de pondérer la mesure de similarité en la multipliant par la mesure de similarité de Jaccard :

$$wsim(x,y) = J(X,Y) \times sim(x,y) \tag{2.5}$$

Cette méthode permet dans certains cas d'améliorer les performances du filtrage.

2.2.2 Sélection des plus proches voisins

Pour un utilisateur x donné, l'ensemble N des plus proches voisins de x pour la mesure de similarité choisie peut être défini de différentes manières.

2.2.2.1 Seuil de similarité

Une méthode assez répandue est de définir N comme l'ensemble des utilisateurs dont la mesure de similarité avec x dépasse un certain seuil s :

$$N = \{y \in \mathcal{U}, sim(x,y) \geq s\} \tag{2.6}$$

2.2.2.2 kNN

Une autre méthode simple consiste à choisir les k plus proches voisins de x au sens de la similarité, avec $k \in \mathbb{N}$ fixé.

2.2.3 Prédiction

Une fois la mesure de similarité et l'ensemble des plus proches voisins définis, il existe différents prédicteurs permettant de déterminer une prédiction de la notation de x sur un objet i. Nous noterons dans la suite $N_i = N \cap \mathcal{U}_i$ l'ensemble des utilisateurs $y \in N$ ayant noté l'objet i, et $\tilde{r_{x,i}}$ la notation prédite de l'utilisateur x sur l'objet i.

2.2.3.1 Moyenne

La prédiction la plus simple à effectuer est de prendre la moyenne des notations des utilisateurs présents dans N_i :

$$\tilde{r_{x,i}} = \frac{1}{|N_i|} \sum_{y \in N_i} r_{y,i} \tag{2.7}$$

Elle attribue la même importance à tous les utilisateurs $y \in N_i$.

2.2.3.2 Moyenne pondérée

Cette prédiction permet d'attribuer plus d'importance aux notes des utilisateurs les plus similaires à x :

$$\tilde{r_{x,i}} = \frac{\sum_{y \in N_i} sim(x,y) r_{y,i}}{\sum_{y \in N_i} |sim(x,y)|} \tag{2.8}$$

2.2.3.3 Moyenne ajustée

Enfin, cette prédiction permet d'attribuer plus d'importance aux notes des utilisateurs les plus similaires à x, tout en neutralisant l'influence de la notation moyenne des utilisateurs :

$$\tilde{r_{x,i}} = \bar{r_x} + \frac{\sum_{y \in N_i} sim(x,y)(r_{y,i} - \bar{r_y})}{\sum_{y \in N_i} |sim(x,y)|} \tag{2.9}$$

2.2.4 Limitations

Plusieurs problèmes peuvent être rencontrés lors de l'application de ce filtrage.

2.2.4.1 Démarrage à froid

Il est impossible d'effectuer une recommandation pour un nouvel utilisateur : il faut attendre qu'il note certains objets. De même, pour un nouvel objet, il faut attendre que des utilisateurs le notent pour pouvoir le recommander à d'autres.

2.2.4.2 Manque de notations

Lorsque la matrice de notations est trop creuse (chaque objet est noté par un nombre réduit d'utilisateurs), les recommandations effectuées par l'algorithme ne sont pas toujours pertinentes.

2.2.4.3 Mise à l'échelle

Dans le cas d'une grosse base de données (nombreux utilisateurs et objets), une puissance de calcul élevée est nécessaire pour chaque recommandation. Il est donc parfois nécessaire d'utiliser un autre algorithme.

2.3 Filtrage collaboratif Item-Based

Ce filtrage fonctionne de manière similaire au filtrage collaboratif User-Based à la différence que, cette fois, on utilise la similarités entre les objets et non entre les utilisateurs. Ainsi, le principe de ce filtrage est de recommander à un utilisateur x des objets dont la notation par les autres utilisateurs est similaire à celle des objets appréciés par x.

2.3.1 Mesures de similarité

Différentes mesures permettent de calculer la similarité entre deux objets. Ces mesures sont semblables aux mesures de similarité entre deux utilisateurs présentées dans le chapitre précédent, nous ne nous attarderons donc pas sur les explications. Nous considérons dans la suite deux objets i et j. A chaque objet i est associé un vecteur de notations q_i défini par : $\forall x \in \mathcal{U}_i, q_{i,x} = r_{x,i}$

2.3.1.1 Mesure de Jaccard

On définit la mesure de similarité de Jaccard entre i et j par :

$$sim(i,j) = J(\mathcal{U}_i, \mathcal{U}_j) = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{|\mathcal{U}_i \cup \mathcal{U}_j|} \tag{2.10}$$

2.3.1.2 Mesure du cosinus

La mesure de similarité du cosinus entre i et j est définie par :

$$sim(i,j) = cos(q_i, q_j) = \frac{\langle q_i \mid q_j \rangle}{||q_i||\ ||q_j||} \tag{2.11}$$

2.3.1.3 Coefficient de corrélation de Pearson

Notons S_{ij} l'ensemble des utilisateurs ayant noté à la fois i et j, et $\bar{q_i} = \frac{1}{|\mathcal{U}_i|} \sum_{x \in \mathcal{U}_i} q_{i,x}$ la valeur moyenne de notation de l'objet i. Le coefficient de corrélation de Pearson est défini par :

$$sim(i,j) = \frac{\sum_{x \in S_{ij}} (q_{i,x} - \bar{q_i})(q_{j,x} - \bar{q_j})}{\sqrt{\sum_{x \in S_{ij}} (q_{i,x} - \bar{q_i})^2} \sqrt{\sum_{x \in S_{ij}} (q_{j,x} - \bar{q_j})^2}} \tag{2.12}$$

Dans ce cas, cette mesure neutralise l'influence de la notation moyenne de chaque objet, ce qui n'est pas toujours souhaitable (une notation moyenne proche peut être un facteur de similarité).

2.3.1.4 Mesure du cosinus ajustée

Pour neutraliser l'influence de la notation moyenne de chaque utilisateur (et non de chaque objet), nous introduisons la mesure du cosinus ajustée :

$$sim(i,j) = \frac{\sum_{x \in S_{ij}} (q_{i,x} - \bar{r_x})(q_{j,x} - \bar{r_x})}{\sqrt{\sum_{x \in S_{ij}} (q_{i,x} - \bar{r_x})^2} \sqrt{\sum_{x \in S_{ij}} (q_{j,x} - \bar{r_x})^2}} \tag{2.13}$$

Elle a donc la même utilité que la mesure de Pearson dans le cas du filtrage User-Based.

2.3.1.5 Mesures Jaccard-pondérées

Pour prendre en compte le nombre d'utilisateurs ayant noté à la fois i et j, il est également possible de pondérer la mesure de similarité en la multipliant par la mesure de similarité de Jaccard :

$$wsim(i,j) = J(\mathcal{U}_i, \mathcal{U}_j) \times sim(i,j) \tag{2.14}$$

2.3.2 Sélection des plus proches voisins

Pour un objet i donné, on peut déterminer l'ensemble N de ses plus proches voisins pour la mesure de similarité choisie de manière analogue à celle utilisée dans le chapitre précédent :

$$N = \{j \in \mathcal{I}, sim(i,j) \geq s\} \tag{2.15}$$

On peut également choisir les k plus proches voisins de x au sens de la similarité, avec $k \in \mathbb{N}$ fixé.

2.3.3 Prédiction

De la même manière qu'au chapitre précedent, nous pouvons désormais calculer une prédiction de la notation de l'utilisateur x sur un objet i. Nous noterons dans la suite $N_x = N \cap \mathcal{I}_x$ l'ensemble des objets $j \in N$ notés par x, et $\tilde{r_{x,i}}$ la notation prédite de l'utilisateur x sur l'objet i.

2.3.3.1 Moyenne

La prédiction la plus simple :

$$\tilde{r_{x,i}} = \frac{1}{|N_x|} \sum_{j \in N_x} q_{i,y} \tag{2.16}$$

2.3.3.2 Moyenne pondérée

Cette prédiction permet d'attribuer plus d'importance aux notes des objets les plus similaires à i :

$$\tilde{r_{x,i}} = \frac{\sum_{j \in N_x} sim(i,j) q_{j,x}}{\sum_{j \in N_x} |sim(i,j)|} \tag{2.17}$$

2.3.3.3 Moyenne ajustée

Cette prédiction permet également de neutraliser l'influence de la notation moyenne attribuée à chaque objet :

$$\tilde{r_{x,i}} = \bar{q_i} + \frac{\sum_{j \in N_x} sim(i,j)(q_{j,x} - \bar{q_j})}{\sum_{j \in N_x} |sim(i,j)|} \tag{2.18}$$

2.3.4 Limitations

Les problèmes rencontrés par ce type de filtrage sont les mêmes que pour le filtrage collaboratif User-Based. Le filtrage collaboratif Item-Based propose toutefois des recommandations plus appropriées en général.

2.4 Filtrage basé sur le contenu

Ce type d'algorithme de recommandation repose sur la proposition à un utilisateur d'objets similaires à d'autres objets qu'il a déjà aimés. Il ne prend aucunement en compte d'autres utilisateurs. Il s'agit du système de recommandation le plus basique, et qui était surtout utilisé dans les premiers algorithmes de recommandation.

Dans ce type de filtrage, un profil de caractéristiques est associé à chaque objet i, sous la forme d'un vecteur, noté ici p_i. Nous noterons $\mathcal{C}$ l'ensemble des caractéristiques, et $\forall c \in \mathcal{C}, p_{i,c}$ désignera la composante du vecteur de caractéristiques correspondant à c. Les objets recommandés à l'utilisateurs seront alors ceux dont le profil est similaire à celui des objets aimés par l'utilisateur.

2.4.1 Différentes mesures

La similarité entre deux objets représente ici la similarité entre leurs vecteurs de caractéristiques. Elle peut être calculée de différentes manières.

2.4.1.1 Distance de Minkowski

Une manière simple de mesurer la similarité entre deux vecteurs est d'utiliser la distance de Minkowski avec $r \in \mathbb{N}$:

$$d_r(i,j) = \left(\sum_{c \in \mathcal{C}} |p_{i,c} - p_{j,c}|^r \right)^{\frac{1}{r}} \tag{2.19}$$

Cette distance généralise par exemple la distance de Manhattan ($r = 1$) ou la distance Euclidienne ($r = 2$). Elle vaut 0 lorsque les deux objets sont identiques.

2.4.1.2 Mesure du cosinus

De la même manière que dans les chapitres précédents, on peut utiliser la mesure de similarité du cosinus pour mesurer la similarité entre les vecteurs de caractéristiques de deux objets :

$$sim(i,j) = cos(p_i, p_j) = \frac{\langle p_i \mid p_j \rangle}{||p_i||\ ||p_j||} \tag{2.20}$$

2.4.1.3 TF-IDF

La mesure statistique TF-IDF (Term Frequency-Inverse Document Frequency) vise à évaluer l'importamce d'un mot dans un ensemble de documents. Cette technique est très utile pour classer les mots en *Natural Language Processing* (NLP). On obtient le résultat par le produit de deux mesures (On notera t le mot, d le document considéré et D, l'ensemble des documents) :

$$\text{tf-idf}(t,d,D) = \text{tf}(t,d) \times \text{idf}(t,D)$$

avec :

— tf la mesure dite *Term Frequency* : On calcule la fréquence d'apparition du mot dans le document :

$$\text{tf}(t,d) = \log(1 + freq(t,d))$$

— idf la mesure dite *nverse Document Frequency* : On mesure si le mot est commun ou rare dans un set de documents : Le mesure tend vers 0 si le mot est commun, et vers 1 si il est rare :

$$\text{idf}(t,D) = \log(\frac{N}{count(t \in d : d \in D)})$$

2.4.2 TextMining

Cette méthode consiste à utiliser des informations qui pourraient se trouver dans des textes descriptifs.

2.4.2.1 Principe

En analysant différents facteurs tels que :
— Fréquence de mots
— Concordance
— Analyse des sentiments
— Détection du sujet
il est possible de trouver des objets similaires. On peut ensuite utiliser un système de Machine Learning :

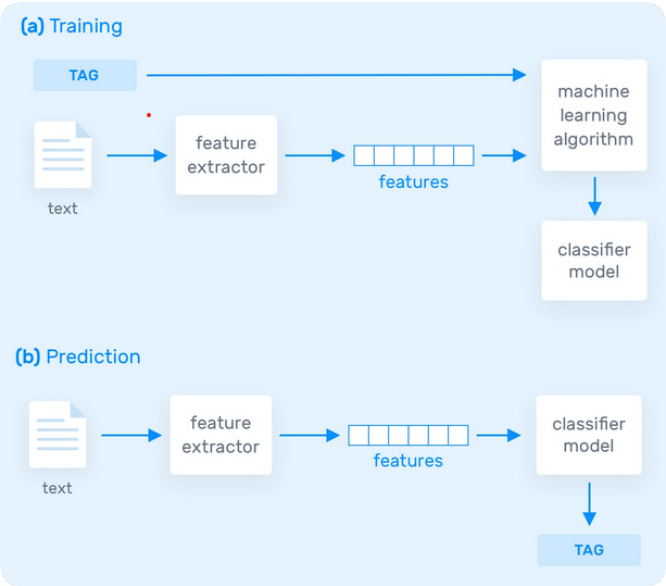


FIGURE 2.2 – Principe de l'algorithme

2.4.2.2 Algorithmes existants

Classification naive de Bayes L'algorithme utilise le théorème de Bayes pour déterminer l'appartenance d'une phrase/description en :

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A) \times \mathbb{P}(A)}{\mathbb{P}(B)}.$$

Par exemple, on souhaite savoir si la description *véhicule ayant très peu roulé* appartient à la catégorie *Neuf*. En supposant "naivement" que chaque mot est indépendant, on obtient :

$$\mathbb{P}(\text{véhicule ayant très peu roulé}|\text{Neuf}) = \mathbb{P}(\text{véhicule}|\text{Neuf}) \times ... \times \mathbb{P}(\text{peu}|\text{Neuf}) \times \mathbb{P}(\text{roulé}|\text{Neuf}).$$

On calcule ensuite la probabilité de chaque mot en fonction de l'apparition dans des descriptions classifiés dans la catégorie *Neuf* (On fera un "Laplace smooting" pour éviter de se retrouver avec un facteur 0 dans le produit). On obitent alors la probabilité d'appartenance à la catégorie de la description. L'algorithme est éevidemment perfectible, notamment en enlevant les mots inutiles ou en utilisant la mesure statistique *TF-IDF*.

L'algorithme possède cependant quelques limites : l'exemple ci-dessus montre clairement que le terme *roulé* dépend du mot précédent (pas le même sens avec *très peu roulé* ou *beaucoup roulé*). Ce modèle est d'ailleurs plus efficace lorsqu'il n'y as pas trop de données d'entraînement...

2.4.3 Prédictions

2.4.3.1 kNN

On introduit une donnée sous forme de points. Le but de l'algorithme vise à classer la donnéee dans une catégorie. On prend pour cela les k points les plus proches (en tutilisant une mesure adéquate). On regarde ensuite parmi ces points lequelles appartiennent en majorité à la même catégorie. On classe ensuite la donnée dans cette catégorie.

2.4.3.2 Classification bayésienne

On utilise la classification naïve de Bayes présenté plus haut pour trouver la classe d'appartenance la plus probable à une donnée.

2.4.3.3 Arbre de décision

On peut créer un arbre de décision, que l'on entraîne avec les voitures déjà "likés" par l'utilisateur. On peut alors proposer des voitures qui sont susceptible de plaire à l'utilisateur.

2.4.3.4 Techniques de Machine Learning

D'autres méthodes de classification plus sophistiquées peuvent être utilisées pour réaliser la prédiction : classification bayésienne, clustering, arbres de décision, réseaux de neurones, ... Actuellement, la plupart des recherches faites sur les systèmes de recommandation sont dans le TextMining et les réseaux de neurones.

2.4.4 Limitations

2.4.4.1 Démarrage à froid

Pour un nouvel utilisateur n'ayant pas noté d'objet, il est impossible d'effectuer une recommandation. En revanche, ce type de filtrage permet d'effectuer une recommandation pour un nouvel objet.

2.4.4.2 Sur-spécialisation

Ce type de filtrage ne permet pas de recommander des objets différents de ce que l'utilisateur a déjà apprécié, négligeant ainsi la possibilité qu'un utilisateur ait plusieurs centres d'intérêt.

2.5 Filtrage collaboratif Model-Based

Le filtrage collaboratif memory-based nécessite le stockage en mémoire d'une quantité très importante de données. Le filtrage collaboratif model-based a alors été créé pour faire face à ce problème. L'idée est de changer la forme des données pour effectuer des recommandations plus rapidement par la suite. Il existe différentes méthodes.

2.5.1 Neural Collaborative Filtering (NCF)

2.5.1.1 Généralisation de la factorisation des matrices

NCF s'appuie sur la factorisation matricielle, qui est entravée par le choix de la fonction d'interaction **produit scalaire** :

$$\hat{y}_{u,i} = f(u, i|\mathbf{p}_u^T, \mathbf{q}_i) = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^{K} p_{uk} q_{ik}$$

Ou

- $\hat{y}(u, i)$: score de prédiction
- $p(u)$: vecteur latent pour l'utilisateur u
- $q(i)$: vecteur latent pour l'élément i
- K : la dimension de l'espace latent

qui peut ne pas être en mesure de capturer entièrement l'interaction utilisateur-élément.

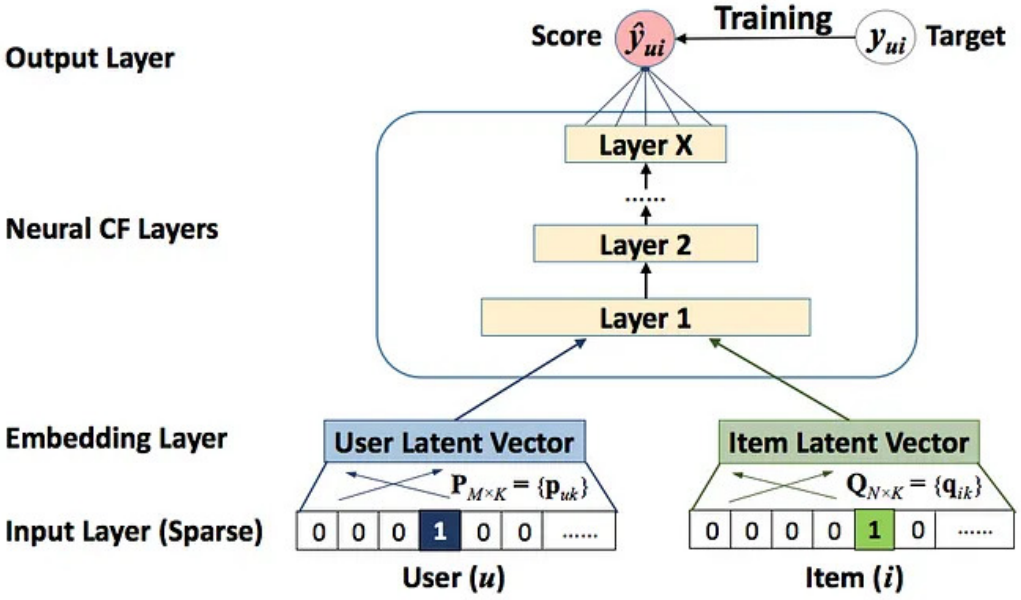
En effet, la raison pour laquelle la similitude est mesurable par le produit interne (ou de manière équivalente le cosinus) est qu'ils sont tous deux mappés dans l'espace latent. Le vecteur latent d'un utilisateur peut représenter son élément "idéal". Il est toujours nécessaire d'augmenter la dimension

de l'espace latent, car cela permet aux éléments d'être plus variables les uns des autres. (Tracer des choses dans une dimension inférieure à celle qui les représente complètement peut entraîner une perte d'informations). Néanmoins, cela nuit également à la capacité de généralisation du modèle. NCF résout ce problème en utilisant les *Multi-Layer Perceptrons* (MLP) pour apprendre la fonction d'interaction (les MLP peuvent **approcher** n'importe quelle fonction continue), cela nous permet également d'exprimer MF comme un cas particulier de NCF.

2.5.1.1.1 Problèmatique Disons que nous avons deux ensembles d'utilisateurs $\mathbb{U} = 1, \ldots, U$ et d'éléments $\mathbb{I} = 1, \ldots, I$ Nous avons également besoin d'avoir un journal des préférences et des interactions de l'utilisateur : (y ne peut pas être défini comme une fonction en raison de données manquantes) $O = (u, i, y)$

D'une certaine manière, le but de NCF est de prédire les interactions non observées entre les utilisateurs et les éléments, et la valeur prédite de l'interaction est : $\hat{y}_{u,i} = f(u, i|\Theta)$

2.5.1.2 Construction du modèle



Initialement, il existe une couche d'entrée pour binariser l'interaction des éléments utilisateur. Ce qui signifie 1 si l'utilisateur a interagi avec l'élément 1 par exemple.

Vient ensuite une couche d'intégration pour créer les représentations latentes des utilisateurs et des éléments. L'un des avantages de l'intégration de couches par rapport à la pure matrice élément-utilisateur est que nous pouvons définir des espaces réservés pour les utilisateurs qui n'apparaissent pas dans l'ensemble de formation et les films qui n'ont été notés par aucun utilisateur en formation.

Après cela, il existe un MLP qui peut être utilisé pour apprendre la fonction f dans notre description précédente.

La couche de sortie finale renvoie le score prédit en minimisant la perte ponctuelle/perte par paire. La perte ponctuelle vise à minimiser la différence entre les scores prédits et cibles, tandis que la perte par paires maximise la marge entre les entrées observées et non observées. Le choix de la fonction de perte dépend du problème spécifique à résoudre. Par exemple, si vous disposez de données de rétroaction explicites telles que des évaluations, la perte ponctuelle peut être plus appropriée. D'autre part, si vous disposez de données de rétroaction implicites telles que des clics ou des achats, la perte par paires peut être plus appropriée.

2.5.1.3 Generalized Matrix Factorization (GMF)

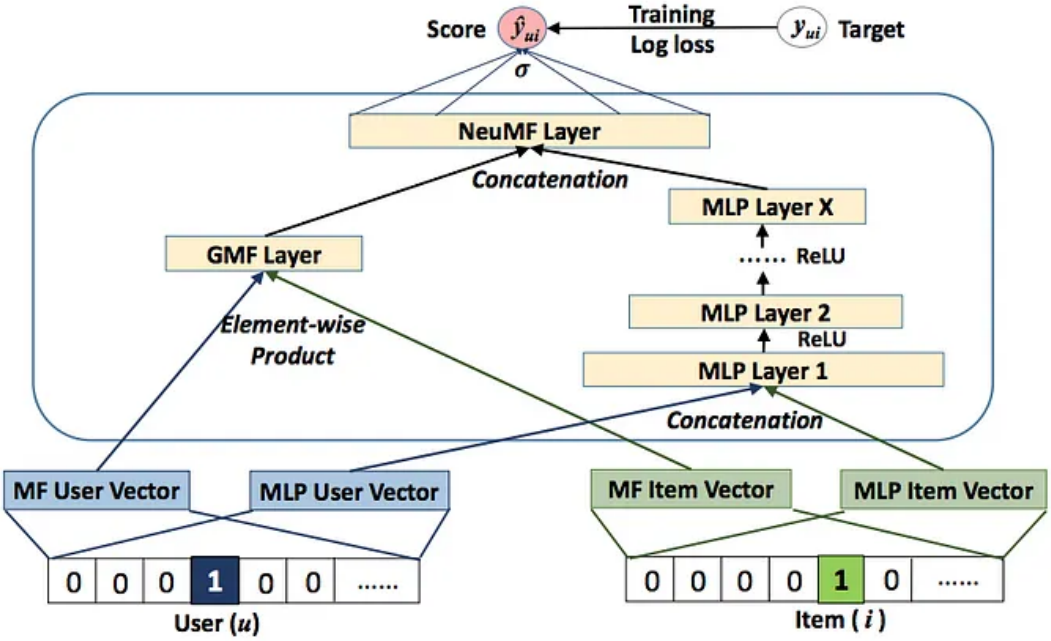
$$\hat{y}_{ui} = a_{out}(h^T(p_u \odot q_i))$$

où a_{out} la fonction d'activation, h les poids des bords de la couche de sortie. (Ce qui prouve encore qu'il s'agit bien d'une généralisation de la factorisation matricielle).

Il faut noter que $p(u)$ et $q(i)$ sont différents de celles de MLP. L'idée est que chacun des deux apprend les plongements appropriés.

2.5.1.4 Modèle final

On combine les deux approches en un seul modèle de manière à obtenir le bien des deux mondes :



2.5.1.5 Inconvénients

Dans un modèle de filtrage collaboratif neuronal (NCF) avec une taille d'entrée fixe, la taille des vecteurs d'entrée signifierait qu'il existe un nombre fixe d'utilisateurs et d'éléments. Cependant, il est possible de mettre à jour le modèle pour accueillir de nouveaux utilisateurs et éléments en le réentraînant sur les données mises à jour. Le recyclage d'un modèle de filtrage collaboratif neuronal (NCF) chaque fois qu'un nouvel utilisateur s'inscrit ou qu'un nouvel élément est ajouté peut en effet nécessiter beaucoup de ressources. Une façon d'atténuer ce problème consiste à utiliser une approche d'apprentissage en ligne dans laquelle le modèle est mis à jour progressivement à mesure que de nouvelles données deviennent disponibles. Une autre approche consiste à recycler le modèle périodiquement, par exemple, une fois par jour ou une fois par semaine, pour intégrer de nouveaux utilisateurs et éléments.

2.5.2 Session-based recommendations based on transformer models

2.5.3 Algorithmes de clustering

Afin d'éviter de chercher les similarités entre tous les utilisateurs pour chaque recommandation, les utilisateurs sont regroupés dans des sous groupes, appelés clusters. On recherche alors les similarités après ce premier tri. Il existe différents algorithmes de Clustering, c'est à dire différentes façons de créer les clusters à partir d'un ensemble d'utilisateurs.

2.5.3.1 Affinity Propagation

Il s'agit d'une technique de clustering qui ne nécéssite pas de savoir à l'avance le nombre de clusters voulu. L'algorithme fonctionne selon un envoi de messages entre les différents utilisateurs, qui indiquent : l'attractivité entre l'émetteur et le récepteur, la disponibilité entre l'émetteur et le récepteur. Tous les utilisateurs communiquent entre eux jusqu'à qu'un consensus d'attractivité soit donné, cela forme les k clusters avec k un nombre adapté à la situation.

2.5.3.2 Clustering Hiérarchique

Dans cette méthode, il s'agit d'obtenir un arbre de clusters, ou chaque cluster a un parent et peut avoir jusqu'à deux enfants. Il existe alors deux façons de créer cet arbre.

Algorithme Descendant A l'état initial, il n"existe qu'un seul cluster regroupant tous les utilisateurs ; on cherche à le diviser pour obtenir des sous clusters.

Algorithme Ascendant Il y a à l'origine n clusters qui correspondent au nombre d'utilisateurs, et on cherche à fusionner les clusters.

A chaque étape, l'opération est faite de sorte à faire des clusters comprenant des utilisateurs les plus similaires possible.

Cette méthode permet d'avoir un nombre flexible de clusters, il est possible de le choisir. En revanche, il a une grande complexité spatiale et temporelle donc n'est possible que sur les petits échantillons.

2.5.3.3 k-means

Idée générale C'est le plus simple et plus connu des algorithmes de clustering. Dans un ensemble de n utilisateurs, on choisit au début un nombre de clusters, k (avec $k \leq n$).
On considère ici que chaque utilisateur est associé à un vecteur "caractéristique".

On initialise l'algorithme en choisissant k vecteurs aléatoires qui représentent les k clusters. Ensuite, chaque utilisateur est attribué au cluster le plus proche à l'aide d'une fonction distance. Une fois chaque utilisateur affecté, on calcule le vecteur moyen de chaque cluster, et on recommence l'attribution avec ces nouveaux clusters. On réitère jusqu'à convergence de l'algorithme, c'est à dire qu'à chaque étape, le vecteur associé à chaque cluster ainsi que les utilisateurs qui lui sont attribués sont stables.

Description précise On considère un ensemble de vecteurs de E, un espace métrique muni de la distance d. Notons les $(x_1, x_2, .., x_n)$.

On choisit dans un premier temps k vecteurs aléatoires de E, $(y_1, y_2, .., y_k)$, ils représentent les centres des k clusters.

Ensuite, chaque point x_i est attribué au cluster dont il est le plus proche. Cela donne les clusters $S_1, S_2, .., S_k$ où $S_i = \{x_j \in E \mid d(x_j, y_i) \leq d(x_j, y_{i^*}) \; \forall \; i^* \in [1; k]\}$

On met à jour les y_i : $y_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j$

On réitère ce processus jusqu'à que les clusters soient identiques à chaque nouvelle itération.

2.5.4 Factorisation de matrices

2.5.4.1 Fonctionnement

On considère un groupe d'utilisateurs, noté U, de taille m et un groupe d'objets, noté I, de taille l. A ces deux groupes on associe une matrice d'avis R telle que r_{kn} est la note donnée par l'utilisateur k à l'objet n. Le but est de créer un système qui prédit avec précision les données présentes (les entrées non vide de R), et ensuite extrapoler qu'il devrait prédire aussi les données manquantes. Pour construire ledit système on se base sur le principe des vecteurs latents, c'est à dire qu'on cherche des matrices A et B telles que :

$$R = AB^\top \tag{2.21}$$

avec $A \in \mathbb{R}^{m*k}$ and $B \in \mathbb{R}^{l*k}$. k représente la dimension de l'espace latent, c'est à dire un espace fictif de représentation des utilisateurs et des objets, qui deviennent des vecteurs latents dans cet espace. Par exemple le vecteur [a_{11},...,a_{1k}] représente l'utilisateur 1 dans l'espace latent.
Cette décomposition permet de calculer les entrées de la matrice R par simple produit scalaire.

$$\hat{r}_{ij} = \sum_{p=1}^{k} a_{ip} b_{pj} \tag{2.22}$$

Il s'agit alors de déterminer les matrices A et B, pour calculer les entrées manquantes dans la matrice.

2.5.4.2 Descente de gradient

La descente de gradient est un outil permettant de converger vers le minimum de fonctions convexes. Il fait parti des techniques les plus utilisées dans le cadre de la détermination des matrices A et B.
Le principe est simple, et peut s'expliquer à travers une analogie géographique. Pour trouver le bas d'une vallée, on cherche la direction de plus grande pente et on se déplace d'un "pas" α dans la direction opposée. Ce cheminement se traduit par un calcul de gradient (la pente), et le choix d'un α appelé pas d'apprentissage. On l'utilise pour minimiser la "root min square error" :

$$e = \sum_{(i,j)} (r_{i,j} - A_i.B_j^\top)^2 \tag{2.23}$$

à laquelle on ajoute des facteurs de normalisation pour éviter un phénomène d'"over-fitting", c'est à dire que le modèle obtenu soit trop spécifique aux données que lequel il a été entraîné. Cela donne l'équation suivante :

$$E = \sum_{(i,j), r_{i,j} \neq 0} (r_{i,j} - A_i.B_j^\top)^2 + \lambda_A \|A_i\|^2 + \lambda_B \|B_j\|^2 \tag{2.24}$$

La descente de gradient permet de calculer de manière itérative chaque ligne ou colonne des matrices A et B pour converger vers le minimum de E, par la formule :

$$A'_i = A_i + \alpha(e_{i,j}B_j - \lambda_A A_i) \tag{2.25}$$

$$B'_j = B_j + \alpha(e_{i,j}A_i - \lambda_B B_j) \tag{2.26}$$

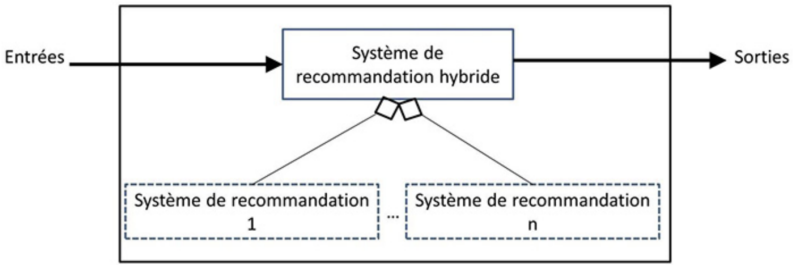
où α est le pas d'apprentissage, λ est le facteur de régularisation et $e_{i,j}$ est l'erreur dans la prédiction. ($e_{i,j} = r_{i,j}$-$A_i.B_j^\top$)

2.5.4.3 Caractéristiques notables

On peut remarquer que dans l'équation (2.24), la somme se fait seulement sur les entrées non vide de la matrice. Cette technique permet donc de prendre avantage du fait que la matrice soit creuse, c'est à dire principalement constituée d'entrées vides, pour réduire le temps de calcul.
Cependant à chaque fois qu'un nouvel user arrive sur la plateforme il faut tout recalculer pour pouvoir lui faire des propositions, et cette méthode est victime du "cold start", elle fonctionne mal sur un nouvel utilisateur qui n'a pas encore émis de préférences. De plus il faut choisir les différents paramètres (k, λ,...) ce qui nécessite plusieurs tests.

2.6 Filtrages Hybrides

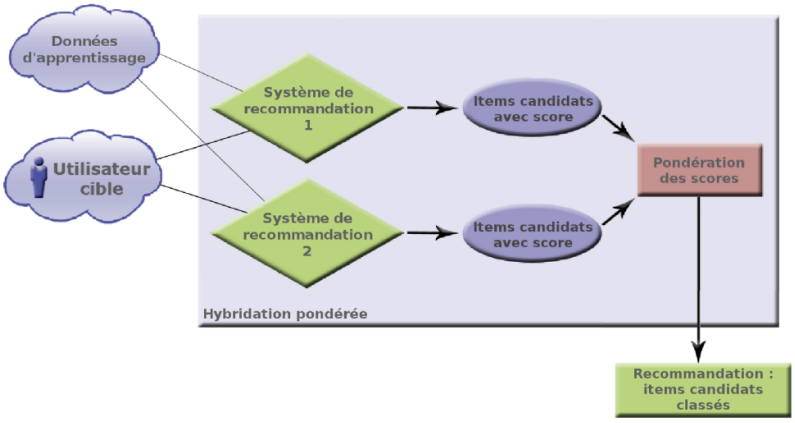
Les systèmes de recommandation basés sur le contenu et ceux basés sur le filtrage collaboratif présentent différentes limitations. C'est pourquoi les systèmes hybrides ont été créés : ils permettent une amélioration des finesses des recommandantions en combinant différents types de filtrages. Il existe un très large éventail de systèmes hybrides. L'ensemble de ces techniques permet d'affiner les résultats des systèmes de recommandation en résolvant les problèmes de rareté et de non-homogénéité des données disponibles.



2.6.1 Hybridation pondérée

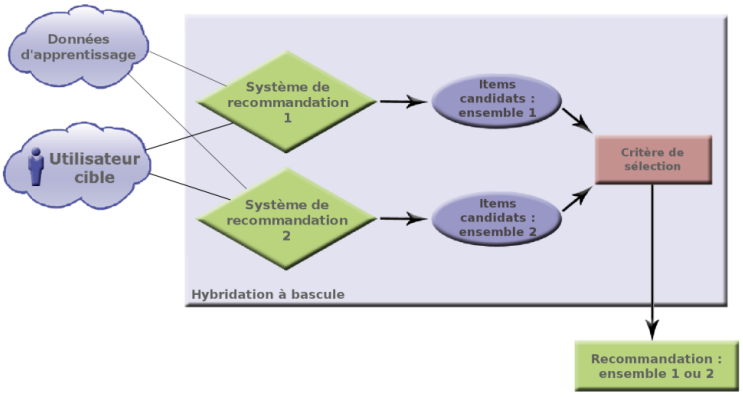
Ce type d'hybridation nécessite que les systèmes de recommandations utilisées effectuent une tâche de prédiction de note. L'hybridation pondérée consiste à calculer le score d'un item grâce à une fonction définie qui est une combinaison convexe des scores calculés avec plusieurs systèmes. Cette fonction peut prendre différentes formes, selon l'importance que l'on donne à chaque sous-système en modifiant l'affectation des poids :

$$\text{score\_final} = \sum_{i=1}^{n} w_i \cdot \text{score\_sys}^{(i)}$$



2.6.2 Hybridation à bascule

Le système alterne entre plusieurs systèmes de recommandation différents selon la situation. On ne combine pas les résultats de plusieurs sous-systèmes, mais on privilégie une approche selon le cas. En cas de résultats insatisfaisants, Ce système peut aussi sélectionner un autre système de notation pour augmenter la qualité de la recommandations. Par exemple, dans le cas où les scores calculés par le système choisi sont trop proches.



2.6.3 Hybridation en cascade

Après avoir obtenu une liste d'objets recommandées par un premier algorithme à partir des données initiales, on rajoute une autre étape de post-traitement des résultats en donnant comme entrée ces derniers à un autre système de recommandation. L'objectif de cette dernière étape est d'obtenir une liste de recommandation plus pertinente et plus affinée.

2.6.4 Hybridation mixée

L'hybridation mixée consiste à obtenir une liste de recommandation de produits en concaténant le résultat de plusieurs systèmes de recommandation. Dans cette approche, chaque système de recommandation produit sa propre liste indépendamment de l'autre système. Ces listes sont ensuite combinées pour construire une liste finale. Cette fusion peut être effectuée sur la base de règles simples, telle que la sélection des éléments les plus fréquents mais aussi avec des algorithmes plus complexe utilisant des réseaux de neurones et des arbres de décision.

2.7 Evaluation des systèmes de recommandation

Différentes méthodes permettent d'évaluer la performance d'un système de recommandation. Cela permet de sélectionner le meilleur système de recommandation à notre disposition, ou d'affiner les paramètres d'un système afin d'améliorer ses performances.

On considère un jeu de données de test $\mathcal{T}$ contenant des notations n'ayant pas servi à entraîner l'algorithme. On note $r_{x,i} \in \mathcal{T}$ la notation de l'utilisateur x sur l'objet i, et $\tilde{r_{x,i}}$ la prédiction de cette notation effectuée par l'algorithme.

2.7.1 Mean Absolute Error (MAE)

Une mesure simple et très utilisée permettant d'évaluer la performance d'un algorithme est de calculer la moyenne des écarts entre la valeur prédite et la valeur réelle sur la base de données de test :

$$MAE = \frac{1}{|\mathcal{T}|} \sum_{r_{x,i} \in \mathcal{T}} |r_{x,i} - \tilde{r_{x,i}}| \tag{2.27}$$

2.7.2 Root Mean Squared Error (RMSE)

Une autre mesure très utilisée est de calculer la racine carrée de la moyenne des carrés des écarts :

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{r_{x,i} \in \mathcal{T}} (r_{x,i} - \tilde{r_{x,i}})^2} \tag{2.28}$$

Cette mesure accorde un poids plus important aux écarts les plus élevés, et est donc plus adaptée que la MAE lorsque des écarts très élevés sont particulièrement indésirables.

Chapitre 3

Description du travail réalisé

3.1 Réalisation de l'état de l'art

3.1.1 Déroulement de la tâche

Cette tâche, a consisté à approfondir l'état de l'art réalisé par nôtre client l'année passée, particulièrement sur les parties de Collaborative Filtering, Model-Based et Hybrides.

Elle a nécessité l'aide de tous les membres du projets, c'est-à-dire que nous nous sommes répartis les chapitres à compléter. Au fur et à mesure de nos recherches, nous écrivions sur les différentes méthodes dans l'état de l'art. Il a fallu à la fin synthétiser nos écrits. Cette tâche nous a pris environ 7 semaines de travail.

3.1.2 Elements techniques

Nous avons donc travaillé sur les réseaux de neurones collaboratifs, le filtrage hybride, le clustering, la factorisation de matrices. Tout ces éléments sont détaillés dans l'état de l'art du rapport.

3.1.3 Livrable

Le livrable était le pdf de l'état de l'art, fourni au client légèrement avant la soutenance intermédiaire.

3.2 Réseau de Neurone récursif

3.2.1 Déroulement de la tâche

Cette tâche a commencée lorsqu'on a reçu les données : nous les avons mobilisées pour construire un réseau de neurones récursif.

Ce tâche a globalement été réalisée par tous les membres du projet, particulièrement Abderrahmane et Rémy. Il a fallu notamment transformer les données en vecteurs, réfléchir à la structure du réseau, à l'entraînement et à l'implémentation sur python, ce qui a été réparti par petits groupes de 2.

La tâche a globalement duré une dizaine de semaines, de la réception des premières données à la fin du projet.

3.2.2 Eléments techniques

Raisonnement Notre premier modèle était une tentative de créer un système de recommandation basé sur les sessions. L'idée est d'avoir une représentation vectorielle de la "voiture idéale" de l'utilisateur, que nous mettrons constamment à jour à mesure que l'utilisateur fait défiler vers la gauche ou la droite afin de mieux refléter ses préférences.

Fonctionnement Ainsi, le modèle aurait 3 entrées :

- Vecteur de voiture : qui est une représentation vectorielle de la voiture que l'utilisateur vient de voir.
- Vecteur de session : ou comme nous l'appelons aussi "vecteur de voiture idéale". Parce que l'idée est d'avoir représenté la voiture idéale pour l'utilisateur, c'est-à-dire la voiture qu'il recherche. C'est ce qui rend cette recommandation basée sur la session.
- Préférence : une valeur binaire 0 (dislike) ou 1 (like).

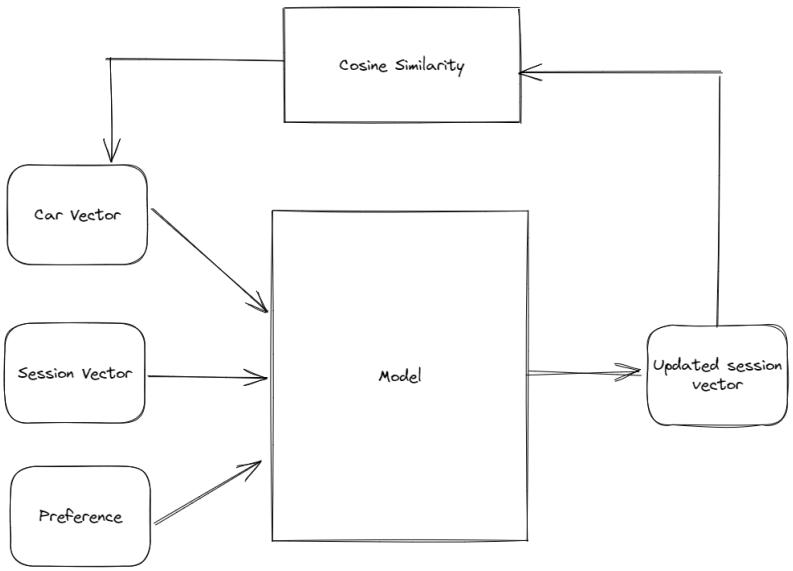


FIGURE 3.1 – Graphe du modèle

Principe En principe, le modèle prends une voiture que l'utilisateur a vu et essaye de mettre à jour le vecteur de session (ou la voiture que nous pensons être la "voiture idéale pour l'utilisateur") en tenant compte de s'il a aimé la voiture recommandée ou non. Pour recommander la voiture suivante, nous utilisons la mesure du cosinus pour rechercher la voiture la plus proche de la nouvelle

voiture idéale de l'utilisateur (vecteur de session mis à jour). En répétant ce processus encore et encore, le vecteur de session devrait finalement représenter avec précision la voiture idéale de l'utilisateur et correspondre mieux à ses préférences.

Qu'est-ce que "model" signifie ici ? Au tout début, nous avons essayé de former un réseau neuronal à propagation avant normal où nous avons joué avec les hyperparamètres pour maximiser ses performances. Pour ce faire, nous avons d'abord concaténé les trois vecteurs. Nous avons transformé "Préférence" en un vecteur en répétant ses valeurs 16 ou 32 fois (volonté de contraindre le modèle à lui accorder plus d'importance).

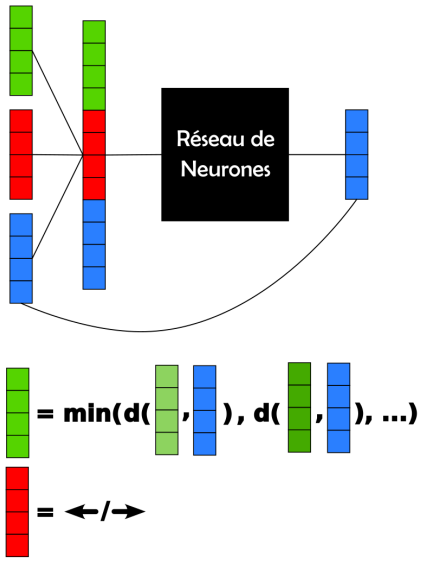


FIGURE 3.2 – Principe de fonctionnement de l'algorithme

Pour améliorer ce modèle et donner plus d'influence aux préférences de l'utilisateur sur le vecteur de session mis à jour, nous avons commencé à expérimenter des approches plus complexes.

Shared Embedding Layer : Dans le but d'améliorer la représentation vectorielle, nous avons décidé d'ajouter une couche d'incorporation vectorielle initiale. Vous pouvez vous questionner quant à la pertinence de ce choix étant donnée que nous disposons déjà d'une représentation vectorielle des données. Nous avions ajouté cette étape pour mieux représenter les caractéristiques catégorielles. En effet, prenant l'exemple de la caractéristique couleur de la voiture. Nous avons décidé d'attribuer une étiquette unique à chaque couleur dans notre modèle, mais cette représentation n'est pas idéale ! En réalité, une représentation idéale consisterait à représenter les couleurs dans un espace colorimétrique quelconque (par exemple, RVB) dans l'espoir d'augmenter les performances du modèle. (Dans une représentation RVB, des couleurs similaires seraient proches les unes des autres, ce qui améliorerait les performances du modèle avec ces données). De même, nous pouvons citer la caractéristique marque de la voiture. Il est possible d'avoir des "marques de voitures similaires" voisines dans la représentation vectorielle que nous donnons au modèle.

La couche d'incorporation partagée sera utilisée à la fois par les vecteurs de voiture ET les vecteurs de session, car tous les deux sont des représentations de voitures.

3.2.2.0.1 Couche conditionnelle : Afin de mettre davantage l'accent sur les préférences de l'utilisateur et en raison du fait que le modèle ne prend pas correctement en compte les valeurs d'appréciation, nous avons exploré plusieurs stratégies. L'une d'entre elles est l'utilisation de couches

conditionnelles. L'idée est de faire passer le vecteur de voiture par différentes couches, en fonction de la préférence de l'utilisateur, afin que les couches ultérieures puissent mieux mettre à jour le vecteur de session de manière à refléter cette préférence.

Enfin, on obtient ce modèle 3.3

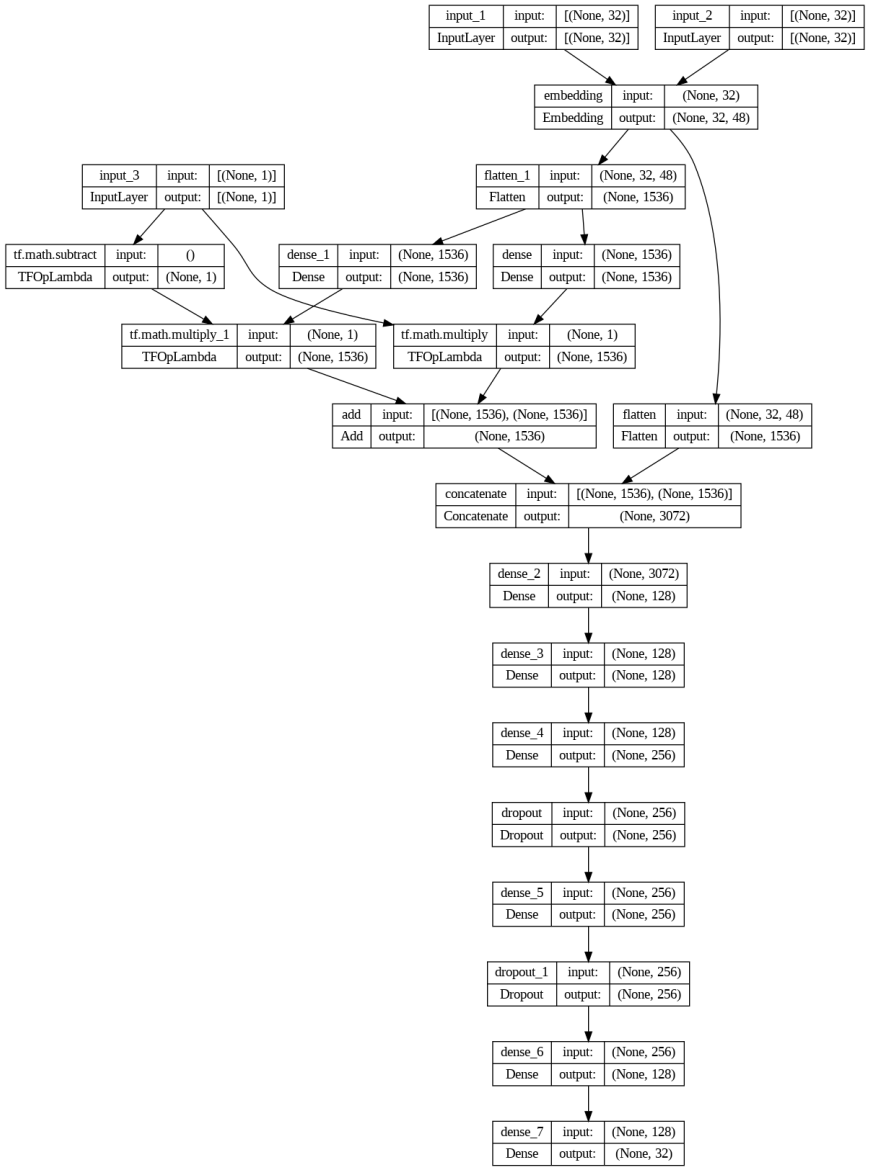


FIGURE 3.3 – Modèle avec une couche conditionnelle et une couche d'embeddings partagé

L'entraînement pour 32 epochs donne le graphe suivant pour la fonction du perte Mean Squared Error (MSE) : 3.4

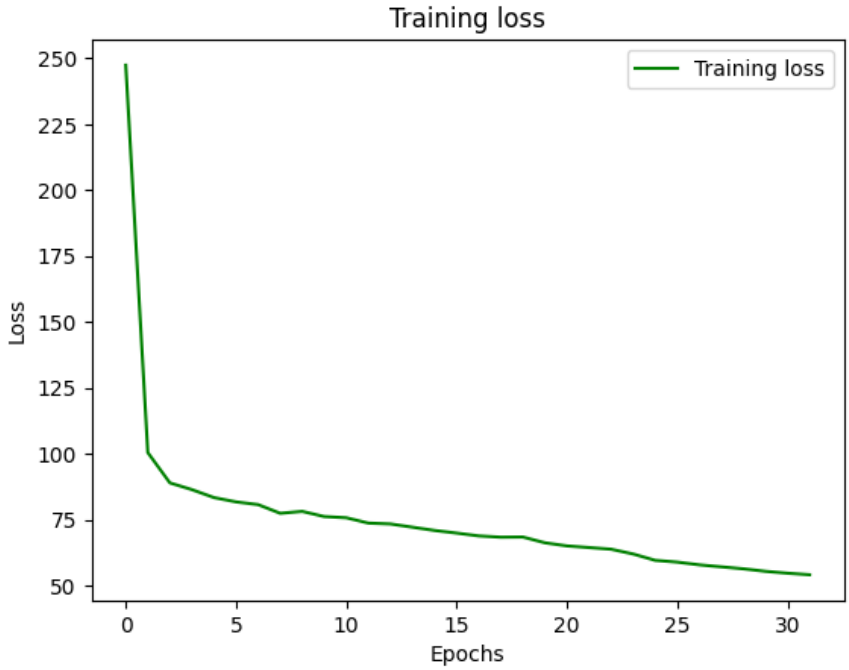


FIGURE 3.4 – Modèle avec une couche conditionnelle et une couche d'embeddings partagé

Couche de rotation : Nous avons consacré beaucoup de temps à réfléchir à la manière dont nous voulons que notre modèle fonctionne essentiellement. Notre objectif est de déplacer le point qui représente la voiture idéale de l'utilisateur dans l'espace, en fonction de ce qu'il pense d'une voiture similaire à cette voiture idéale. C'est pourquoi nous avons pensé à la couche conditionnelle. Cependant, nous nous retrouvons confrontés à un problème de complexité. La couche conditionnelle, bien qu'elle fonctionne très bien, implique également une grande partie des paramètres du modèle. De plus, elle est beaucoup plus difficile à mettre à l'échelle. La couche de rotation est une couche personnalisée très similaire à la couche conditionnelle, sauf qu'elle contient une "matrice de rotation" (pas au sens mathématique strict) qui cherche à faire tourner le vecteur de la voiture idéale dans l'espace afin de l'éloigner d'une voiture qui ne plaît pas à l'utilisateur.

La sortie de cette couche est :

$$Preference \times VecteurDeVoiture + (1 - Preference) \times MatriceDeRotation \times VecteurDeVoiture$$

Bien sûr, nous pouvons également ajouter deux matrices de rotation différentes pour chacune des deux préférences. Cette approche réduit considérablement le nombre de paramètres, car chaque matrice entraîne 32x32 paramètres, comparés aux couches denses de la couche conditionnelle. (Une conditionnelle complexe entraîne environ 5 millions de paramètres, contre environ 160 000 pour le modèle de rotation)

Enfin, on obtient ce modèle (ici on a enlevé les embeddings pour bien souligner la couche du rotation. 3.5

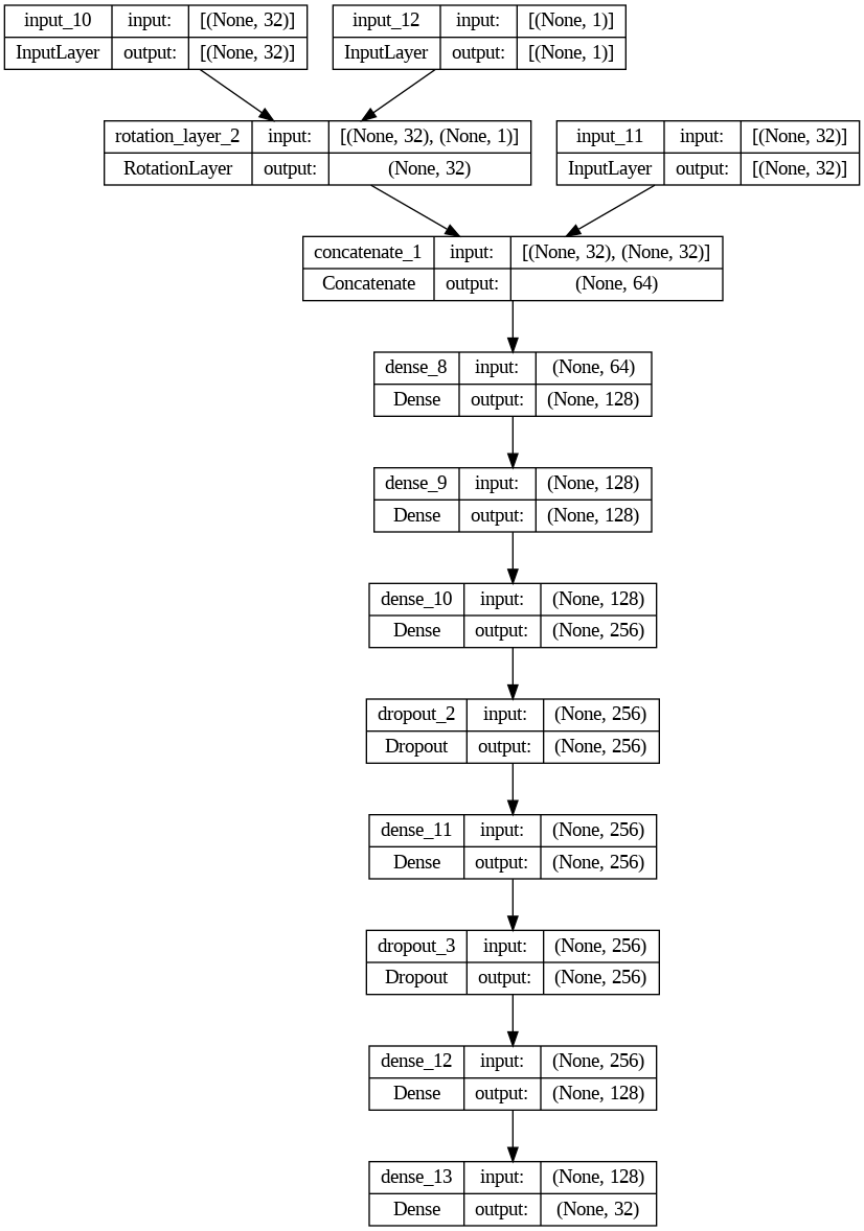


FIGURE 3.5 – Modèle avec une couche rotationelle et sans embeddings

L'entraînement pour 32 epochs donne le graphe suivant pour la fonction de perte Mean Squared Error (MSE)

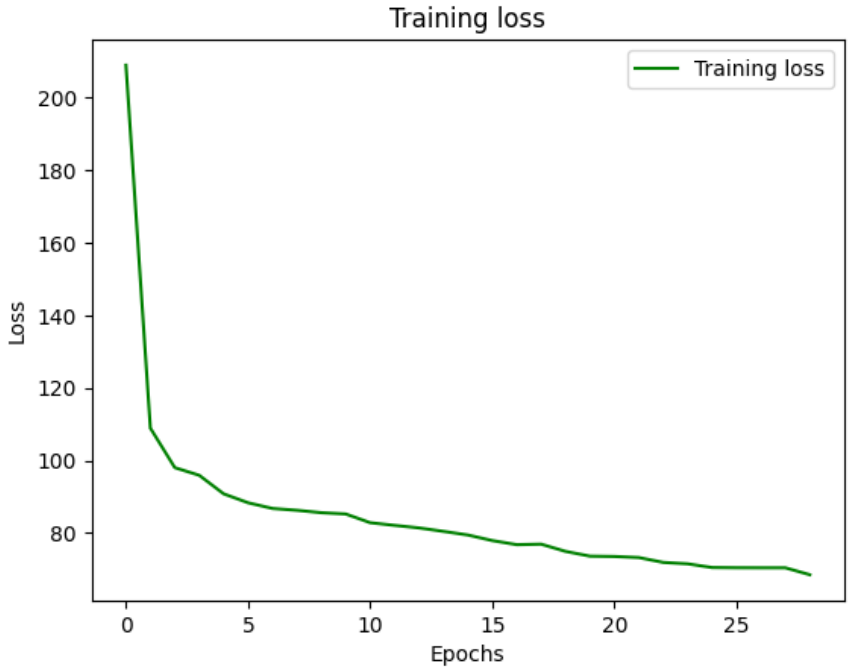


FIGURE 3.6 – Modèle avec une couche conditionnelle et une couche d'embeddings partagé

Entraînement : Lorsqu'il s'agit d'entraîner le modèle, nous avons dû choisir entre deux fonctions de perte pertinentes :

— Erreur quadratique moyenne

— Similarité cosinus

Chacun des deux méthodes a ses avantages et ses inconvénients. Par exemple, la similarité cosinus est assez proche de l'application réelle du modèle (parce que nous utilisons la similarité cosinus pour comparer le vecteur de session mis à jour à toutes les autres voitures de la base de données). En outre, elle conduit également à des valeurs de sortie explosées dans certaines configurations du modèle (parce que contrairement à l'erreur quadratique moyenne, l'amplitude des valeurs n'est pas vraiment prise en compte pendant l'entraînement). L'erreur quadratique moyenne résout ce problème et est aussi beaucoup plus facile à quantifier. Néanmoins, elle n'est pas totalement compatible avec les angles que nous voulons obtenir avec la similarité cosinus.

Performance :

Pour estimer le temps de recommandation, nous avons effectué 100 prédictions avec notre modèle. Nous avons utilisé une instance Google Colab dotée une carte graphique NVidia A100. Nous présentons les temps d'inférences moyens pour les deux modèles : figure 3.7 et figure 3.8

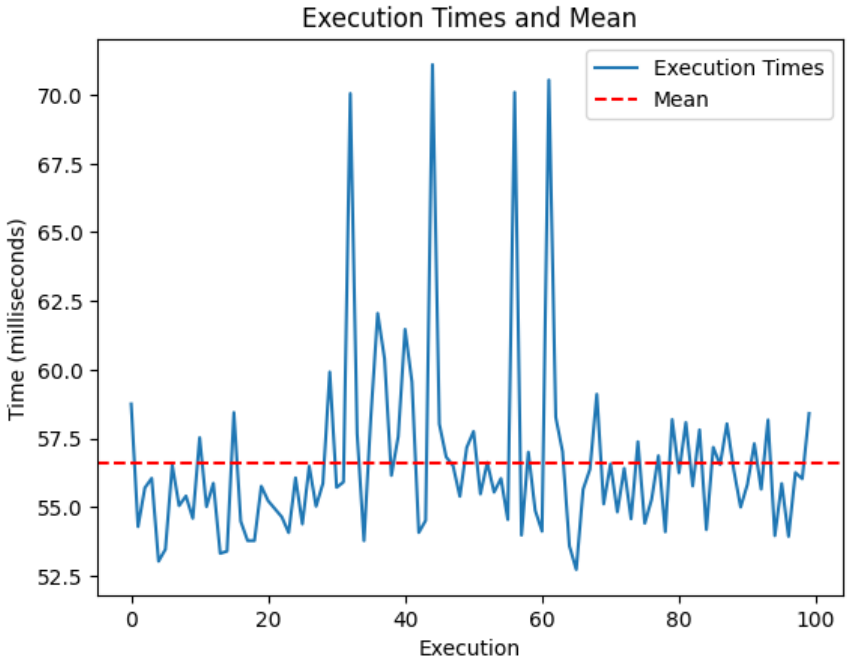


FIGURE 3.7 – Modèle avec une couche rotationelle et sans embeddings

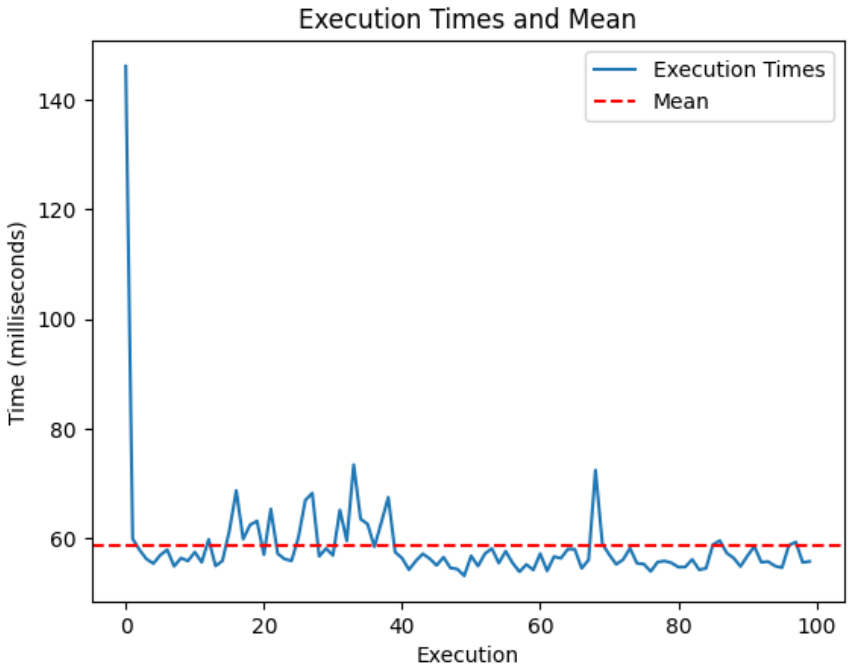


FIGURE 3.8 – Modèle avec une couche conditionnelle et avec embeddings

Notre méthode utilise aussi la mesure du cosinus sur tout le jeu de données, on a testé avec une implémentation naïve.cela donne pour le modèle avec une couche rotationelle et sans embeddings 3.9 :

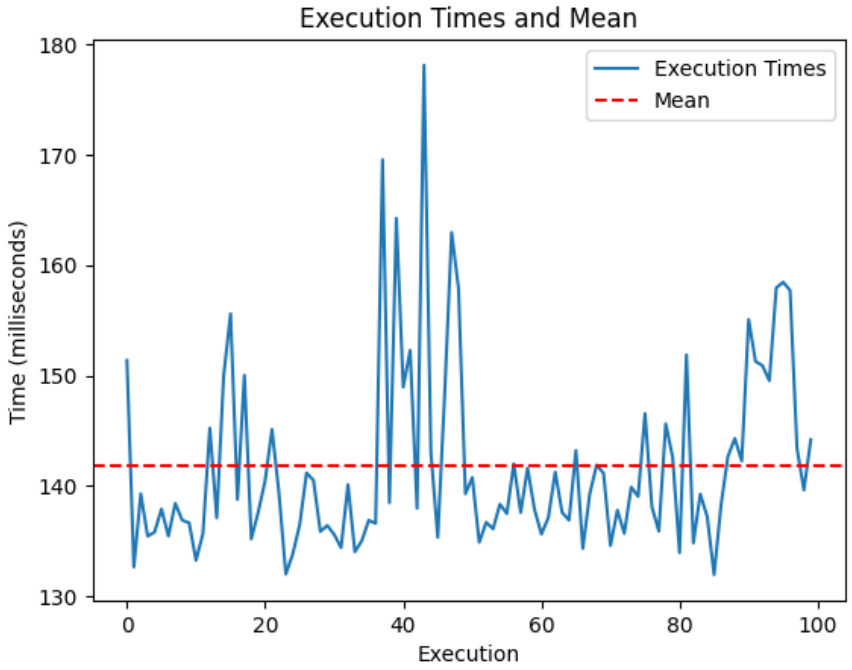


FIGURE 3.9 – Modèle avec une couche rotationelle et sans embeddings

On remarque que le temps de calcul de la similarité cosinus pour l'ensemble du jeu de données est supérieur à l'inférence du modèle. Notre implémentation était plutôt naïve, et peut être grandement optimisé. En effet, en introduisant une parallélisation et une utilisation efficace des instructions spéciales du processeur (SIMD) ou utilisation des bibliothèques qui utilisent les facilités de CUDA (cupy par exemple), il est possible de réduire considérablement cette valeur lors du déploiement de l'application. De plus, avec la clusterification présentée dans la partie suivante, il est possible de rendre la recommandation encore plus rapide.

Précision :

Pour mesurer la précision du système de recommandation, nous nous sommes heurtés à une problématique majeure : les systèmes de recommandations sont subjectifs ! Pour surmonter cette difficulté, chaque membre de l'équipe a testé le modèle en essayant de retrouver des voiture selon des marque et des couleurs précises par exemple. Nous avions réussi à les retrouver facilement au bout de 12 swipes, voire 4 swipes pour une vingtaine de voitures. Cela s'approche d'un cas utilisateur, car celui ci n'a vraiment une idée très précise de la voiture qu'il recherche (toutes les caractéristiques). Globalement, nous avions jugé que le système de recommandation fonctionne de manière satisfaisante.

3.2.3 Livrable

Nous avons fourni au client :

— un Jupyter notebook qui contient le traitement des données
— un Jupyter notebook qui contient l'entraînement du modèle,
— les fichiers Zip contenant les modèles enregistrés (les poids qui résultent de l'entraînement)
— une interface graphique pour tester les swipes.

3.3 Clustering (apprentissage non supervisé)

3.3.1 Déroulement de la tâche

Pour mieux comprendre notre base de données de voitures et analyser l'influence des différentes caractéristiques sur la répartition des véhicules, nous avons choisi d'utiliser une approche de clusterisation. Cette approche nous nous permettra d'identifier des groupes de voitures similaires et de découvrir des structures dans nos données.
Cette tâche se divise en principalement en 3 étapes : choisir les caractéristiques utilisées pour le clustering, déterminer le nombre optimal de cluster, effectuer la clusterefication et enfin interpréter les résultats. Cette tâche s'est étalée sur 3 semaines, en parallèle d'autres tâches, et a été réalisé par Elias et Eliott.

3.3.2 Éléments techniques

Hypothèses : Avant de commencer à travailler sur le clustering, il est judicieux de définir les hypothèses du modèle :

- Nous supposons que les voitures de la base de données partagent des similarités et caractéristiques communes qui peuvent être regroupé en clusters significatif
- La taille du dataset est suffisamment grande pour permettre une identification précise des clusters (clusters linéairement séparables)
- Nous supposons que l'utilisateur cible des catégories de voitures et n'est pas spécifiquement à la recherche d'une voiture bien précise. Cette hypothèse est raisonnable dans notre cas, car sinon, une recherche directe par filtres dans d'autres sites ou application de vente serait plus pertinente que l'utilisation de l'application *Izyleaf*.

Caractéristiques utilisées pour le clustering : Les caractéristiques retenues pour le clustering sont : **le prix, l'année, le kilométrage, les chevaux fiscaux et la puissance du moteur** (DINPower). La prise en compte des caractéristiques catégorielles, comme la couleur par exemple, complexifie significativement le modèle sans qu'il soit vraiment plus pertinent. En prenant en compte les caractéristiques catégorielles (avec encodage), l'elbow méthode prédit un nombre optimal 30 clusters pour notre base de donnée, ce qui est très élevé et difficilement interprétable (comparé aux résultats sans ces catégories dans la suite). On ne tient alors pas compte de ces paramètres catégoriels.

Elbow méthode La méthode du coude (Elbow method) permet de déterminer le nombre optimal de clusters K dans un algorithme de clustering, dans notre cas pour l'algorithme **Kmeans++**. L'idée principale derrière cette méthode est d'évaluer la variation de l'inertie intra-cluster en fonction du nombre de clusters. L'inertie intra-cluster mesure la somme des distances au carré entre chaque point de données et le centre de son cluster correspondant. Plus l'inertie est faible, plus les points d'un cluster sont proches les uns des autres. Pour l'appliquer, on calcule l'inertie pour différents nombres de clusters. Ensuite, on trace un graphique du nombre de clusters en fonction de l'inertie. Le graphique ressemble à une courbe qui présente une forme de "coude". Le point de coude correspond au nombre de clusters où l'inertie cesse de diminuer rapidement et commence à diminuer lentement.
En l'appliquant à notre modèle on trouve **K=3**, comme le montre la figure 3.10 .**Cela signifie que nos données peuvent être regroupées en 3 clusters distincts en fonction de leurs caractéristiques**.

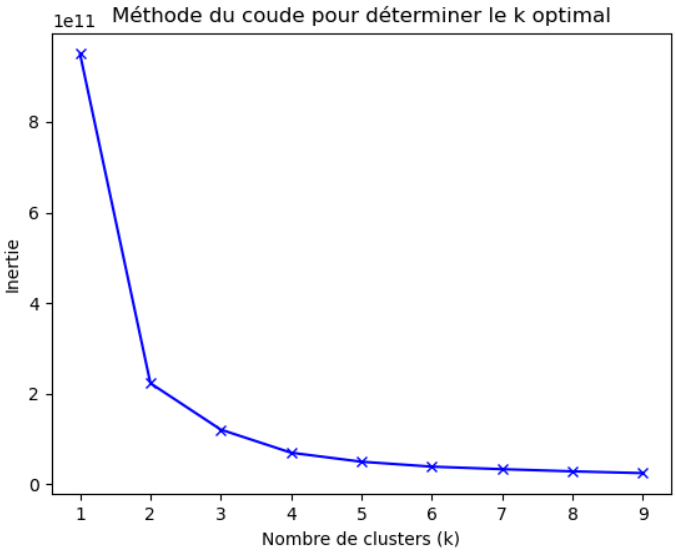


FIGURE 3.10 – Elbow method

Clusterification Nous nous sommes appuyé sur l'algorithme de clustering **K-means**++. La sélection initiale des centres de cluster peut avoir un impact significatif sur les performances de l'algorithme K-means. Nous avons donc privilégié l'algorithme K-means++ qui améliore la sélection des centres initiaux en les choisissant de manière plus intelligente. En effet, K-means++ ne les sélectionne pas de manière aléatoire mais suit une approche de sélection pondérée, où les centres initiaux sont choisis de manière qu'ils soient à une distance plus éloignée les uns des autres. Cela permet d'obtenir une meilleure répartition initiale des clusters et conduit généralement à une convergence plus rapide et à une meilleure solution finale. Nous proposons une des représentation possible dans la figure 3.11.

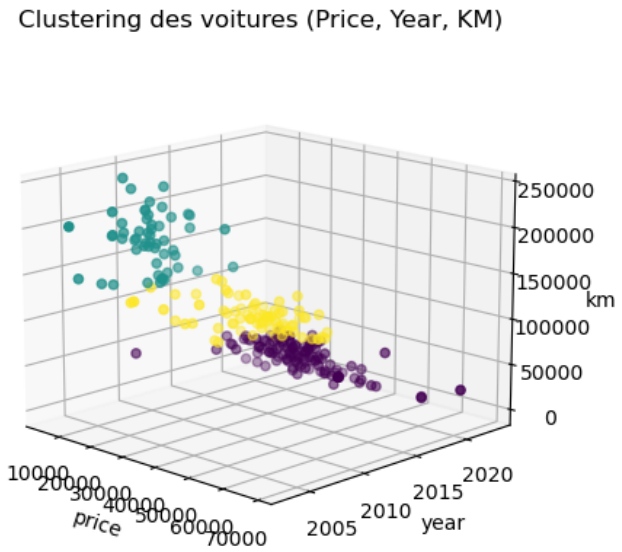


FIGURE 3.11 – (Price,Year,KM)

Interprétation des résultats : Les résultats sont cohérents. Dans la figure 3.11, on retrouve les trois types de véhicules généralement listés dans les sites de vente de voitures d'occasion :

- SUV (Sport Utility Véhicule) : Les SUV sont des véhicules polyvalents qui offre à la fois un confort et des capacités tout-terrain. **Elles sont principalement caractérisées par un prix élevé et un kilométrage modéré. (en mauve dans la figure)**.
- Berline : Les berlines sont des voitures de taille moyenne à grande. Elles sont généralement conçues pour offrir un bon équilibre entre confort, espace intérieur et performances routières. **Elles sont principalement caractérisées par un prix modéré et un kilométrage modéré. (en jaune dans la figure)**.
- Citadine : Les citadines, sont des véhicules de petite taille conçus pour une conduite urbaine. Elles sont appréciées pour leur maniabilité, leur économie de carburant et leur facilité de stationnement dans les espaces restreints. **Elles sont principalement caractérisées par un prix faible et un kilométrage élevé. (en vert dans la figure)**.

Conclusion : La clusterisation des voitures présentes dans la base de donnée est satisfaisante et cohérente. Elle nous a permis de mettre en évidence des structures dans nos données et de mieux comprendre la répartition des voitures. Ces résultats peuvent devenir très utiles voir essentiels lorsque la base de donnée des voitures devient volumineuse. En effet, cela nous permettra de restreindre les utilisateurs dans certains clusters lorsqu'ils auront un historique suffisamment conséquent, réduisant ainsi les calculs aux voitures présentes uniquement dans ces clusters. Cette approche a donc un impact significatif sur le temps de recommandation et la consommation de ressources GPU, améliorant ainsi l'efficacité et la rapidité du processus de recommandation.

3.3.3 Livrable :

Nous avons fourni au client un jupyter notebook commenté qui permet de :

- choisir les caractéristiques pris en compte
- appliquer l'elbow méthode et visualiser les résultats
- appliquer l'algorithme Kmeans++
- visualiser les clusters

3.4 kNN (apprentissage supervisé)

3.4.1 Déroulement de la tâche

Le but de cette tâche est de fournir un algorithme de recommandation basé sur le principe de fonctionnement de l'algorithme kNN (k Nearest Neighbor). Elle se divise principalement en 3 étapes. Tout d'abord choisir les données pertinentes et les mettre en forme pour quelles soient exploitables pour kNN. Ensuite il s'agit d'optimiser au mieux les hyperparamètres (ici k le nombre de voisins). La dernière partie de la tâche est de compiler les différents morceaux de l'algorithme (traitement des données, entraînement, optimisation) dans une fonction exploitable par le client. L'ensemble du processus s'est étalé sur 2 semaines et a été traité par Laure.

3.4.2 Eléments techniques

Le premier point technique, avant même d'essayer d'utiliser kNN, est la sélection des données utiles pour décrire l'espace des voisins. Garder toutes les caractéristiques des voitures dans cet espace ne s'avère pas pertinent car l'utilisateur n'accorde pas la même importance aux différentes informations du véhicule. On peut facilement concevoir que le prix est souvent bien plus important que le nom du vendeur. Il faut donc déterminer quelles variables sont réellement importantes. Une première sélection a été faite de manière arbitraire en demandant aux membres de l'équipe quelles étaient les variables qu'ils regarderaient en priorité lors de l'achat d'une voiture d'occasion. Le prix, la marque, le kilométrage, le type de carburant, la boite de vitesse et la couleur sont les variables qui sont ressortie. Ensuite la qualité des prédictions du modèle a été testée en changeant les variables d'entrée pour voir l'impact sur l'algorithme. Il est ressorti de ces tests que toutes les variables étaient pertinentes, elles ont donc toutes été conservées dans le modèle.

Une fois les variables choisies, il s'agit de les transformer pour qu'elles puissent être interprétées par kNN. En effet les variables doivent être du type int ou float, ce qui n'est pas le cas de la marque par exemple. S'il n'est pas compliqué d'attribuer un entier à chaque marque, le problème de la proximité se pose. En effet, comme kNN s'intéresse à des voisinages, la marque 1 sera plus proche de la marque 2 que de la marque 25 ce qui influe fortement sur le résultat de l'algorithme. Pour la couleur il est possible de se baser sur le codage RGB pour chiffrer la variable, mais dans le cas des voitures il a été décidé de plutôt classer les couleurs par ordre croissant de part du marché automobile. Dans le cas des marques et des modèles le choix de l'ordre a été fait par test aléatoire. Une centaine de permutation de la liste des marques ont été générées pour définirs une centaine d'ordres différents puis les performance de kNN ont été calculé pour tous les utilisateurs en utilisant les différents ordres. Ensuite le score moyen de chaque ordre a été calculé et l'ordre ayant le meilleur score moyen a été retenu. Cela ne garanti pas que l'ordre choisi soit le meilleur possible mais permet d'améliorer le modèle par rapport à un choix totalement arbitraire.
Une fois toutes les variables numérisées, elles sont normalisées, c'est à dire converties en un nombre entre 0 et 1 en utilisant la méthode minmax.

$$minmax(x) = \frac{x - xmin}{xmax - xmin} \tag{3.1}$$

Cela permet de ne pas trop éloigner certains points juste à cause de la disproportion entre les échelles de mesure.

Les variables sont maintenant choisies et adaptées à l'algorithme. Nous pouvons passer à l'amélioration de l'algorithme en lui même. Pour cela 2 hyperparamètres doivent être optimisés. Tout d'abord le poids des voisins. Celui-ci peut être uniforme (tous les voisins ont le même poids) ou inversement proportionnel à la distance (plus un voisin est loin, plus son poids est faible). En testant les 2 le poids uniforme s'est révélé plus adapté. Ensuite il faut déterminer l'hyperparamètre k qui

indique le nombre de voisins pris en compte par le modèle. Ce choix se fait à l'aide d'une fonction d'optimisation adaptée incluse dans le module sklearn. Il faut ensuite intégrer l'algorithme dans l'application. Pour cela, comme la classification en like et dislike est propre à chaque utilisateur, le modèle et ses hyperparamètres doivent être réajustés pour chacun. Le fonctionnement est donc le suivant : la liste des voitures "swipées" est fournie, les données sont transformées puis entrées dans le modèle pour calculer k et l'entraîner. De là on utilise le modèle pour prédire les notes attribuées à une liste de voitures non-jugées fournie.

3.4.3 Livrable

Le client a demandé une fonction prenant en argument :

— une liste de voiture dont on veut prédire la note (une voiture étant représentée par un dictionnaire de ses informations)

— la liste des voitures déja "likées" sous forme de couple. Le premier élément est une voiture et le deuxième la note que l'utilisateur lui a attribuée.

Les deux critères principaux d'évaluation du modèle sont sa précision et son temps d'exécution. Le modèle a été testé sur les likes effectués par l'utilisateur 1014 lors de la deuxième session de la bêta d'*IzyLeaf*. Les 170 premiers likes ont été utilisés pour prédire les 55 derniers. Le score obtenus par l'algorithme sur ses 55 prédictions est de 0.77, et le temps d'exécution est de 0.1s. Le score est le nombre moyen de prédictions justes, soit dans le cas de ce test :

$$score = \frac{nb_{accurate}}{55} \tag{3.2}$$

ou $nb_{accurate}$ est le nombre de prédictions correctes.

Chapitre 4

Conclusion

Travailler sur ce projet a été une expérience enrichissante pour chacun d'entre nous, tant sur le plan technique que humain, à travers la gestion d'un projet de groupe sur une durée significative, la gestion d'un client et des échéances collectives vis à vis de l'école et des encadrants.

Nous avons, dans une première partie, acquis des compétences sur les algorithmes de recommandations de façon générale, grâce à l'état de l'art. Pour beaucoup d'entre nous, nous partions de zéro dans le domaine, et même dans l'IA de façon générale. Nous avons acquis la base des systèmes de recommandations de cette façon.

Il a fallu, dans un second temps, trouver un modèle de recommandation adapté au client. Ce projet nous a vraiment permis de monter en compétence en ce qui concerne l'intelligence artificielle. Nous avions exploré différentes facette de l'IA, en nous concentrant sur trois aspects : le DeepLearning avec l'implémentation du réseau de neurones, l'apprentissage non supervisé avec la clusterification par l'algorithme des Kmeans, et finalement l'apprentissage supervisé avec les KNN.

En terme de création de valeur, nous avons essayé de créer des systèmes pertinents, performants et rapides pour le client. Celui-ci pourra continuer de les améliorer par la suite, et particulièrement de continuer d'entraîner les modèles avec encore plus de données s'il en a les ressources. L'application sera ainsi plus performante et pourra convaincre plus de client, c'est-à-dire améliorer sa réputation etc... De plus, en ayant plus de clients, elle aura plus de données et pourra encore améliorer nos modèles, qui dépendent majoritairement des données pour être performant.

Concernant l'avancée du projet au cours du temps, nous avons peut-être trop attendu l'arrivée des données du clients, et donc pris un léger retard sur l'implémentation des algorithmes. Nous aurions pu commencer plus tôt, quitte à travailler avec des données aléatoires dans un premier temps, pour au moins construire la base en code de nos algorithmes.

Nous tirons ainsi de ce projet de nombreuses compétences techniques d'intelligence artificielle et d'algorithmes de recommandations. Nous avons aussi beaucoup appris en terme de gestion d'un projet de groupe sur 6 mois, et nous attendons le prochain projet pour pouvoir appliquer les plus grandes leçons apprises cette année.