TX - Sopra Steria, Data Mining & Machine Learning

Etudiants : Alexandre LACOUR et Paul GOUJON
Suiveur Sopra Steria : Pascal PEZARD
Superviseur UTC : Sébastien DESTERCKE

Printemps 2016

Table des matières

Chapitre 1

Avant Propos

1.1 Introduction

Sopra HR Lab, département de l'entreprise **Sopra Steria**, développe des solutions digitales innovantes ayant pour but de répondre aux besoins des divisions Ressources Humaines (HR en anglais, RH en français, nous utiliserons les deux) des clients de Sopra Steria. Les solutions du Sopra HR Lab se veulent adaptées aux entreprises publiques comme privées, dans tous les secteurs d'activité et tentent de couvrir de manière exhaustive les sujets de Gestion administrative & Paie, Gestion des Temps & des Activités, Gestion des Talents, Core HR International, Pilotage et Performance, Espaces Collaboratifs, etc. Dans ce contexte, le lab a ainsi souhaité étendre sa gamme de produits RH en développant un conceptware permettant d'améliorer les interactions et les tâches de gestion RH des entreprises.

Un logiciel intelligent Ce logiciel se veut "user-oriented" et "user-friendly", afin d'humaniser les rapports des collaborateurs et des managers avec les tâches RH, c'est pourquoi Sopra Steria désire intégrer à ce logiciel des méthodes de Data Mining et Machine Learning, afin d'être en mesure de fournir des interfaces auto-adaptatives (et donc plus "intelligentes") à ses utilisateurs. C'est dans ce cadre que s'inscrit notre projet "Intégration de méthodes de Data Mining et de Machine Learning à des fins de développement d'interfaces auto-adaptatives au sein d'un logiciel HR Sopra Steria"

1.2 Rappel du cas d'usage considéré

Contexte Notre travail s'articule autour du cas d'usage "Système de recommandation à visée d'ordonnancement d'items au sein d'éléments d'interface" (widgets au sein d'une page, applications au sein d'un catalogue...). Nous nous intéressons particulièrement dans notre mise en application à un premier exemple concret, duquel découlera probablement beaucoup d'autres : l'affichage ordonné de widgets au sein d'un catalogue de widgets.

Illustration du cas d'usage Afin de bien situer l'usage, il s'agit de la situation selon laquelle l'utilisateur souhaite ajouter un nouveau widget au sein de son interface utilisateur. Il a donc la possibilité de choisir un nouveau widget parmi un catalogue de widgets.

Objectif Notre objectif est donc dans ce contexte que l'affichage du catalogue affiche en priorité les widget susceptibles de plaire à l'utilisateur, et ainsi recommander une liste de widgets pour cet utilisateur dans l'ordre de leur pertinence supposée.

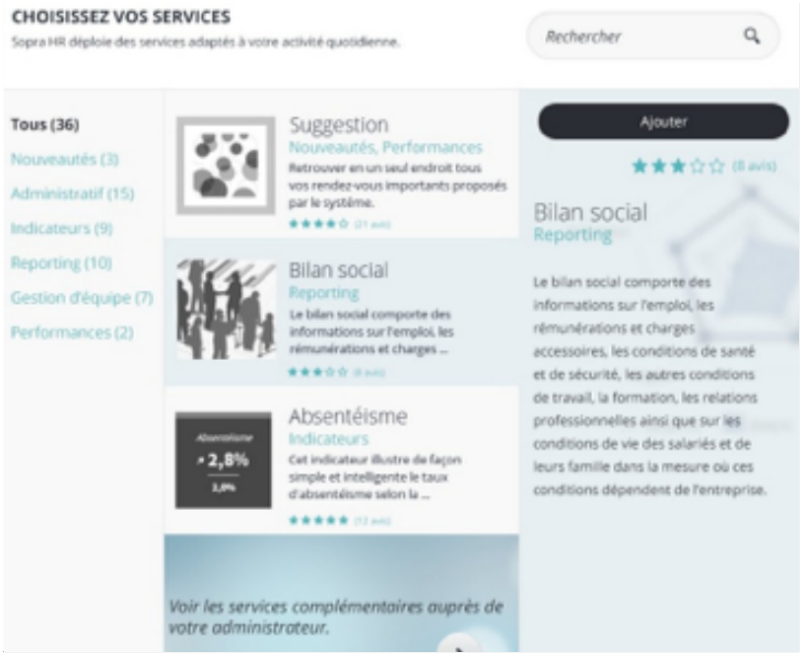


FIGURE 1.1 – Représentation des widgets au sein du catalogue

1.3 Contenu du rapport

1.3.1 Première partie - Architecture technique

Contenu La première partie de ce rapport se concentre principalement sur l'architecture technique de notre solution. Cette partie contient notamment :

1. **Présentation générale** : Une présentation générale de l'architecture du module.
2. **Structures de données** : Les hypothèses initiales concernant les structures de données manipulées desquelles découlent notre implémentation en base de données.
3. **Architecture technique** : Une description plus détaillée du fonctionnement de notre application, orientée "génie logiciel"
4. **Système de recommandation** : Une présentation de la solution utilisée pour l'implémentation de notre système de recommandation.
5. **Fonctionnement global** : L'explicitation du fonctionnement, concrètement, pour l'utilisateur final (Sopra Steria).

1.3.2 Seconde partie - Approche scientifique

Contenu La seconde partie de ce rapport traite avec une approche plus scientifique le fonctionnement intrinsèque du système de recommandation. Elle contient notamment :

1. **Résultats de veille** : La présentation des deux méthodes de prédiction de recommandations que nous avions retenues de notre travail de veille initial.
2. **Choix de la méthode de Machine Learning & motivations** : Des explications concernant notre choix de méthode de Machine Learning et nos motivations au moment d'effectuer ce choix.
3. **UBCF** : Une présentation des concepts théoriques à l'origine de l'User Based Collaborative Filtering, et de sa mise en pratique au sein de notre projet
4. **Matrix Factorization** : Une présentation des concepts mathématiques intrinsèques à la Matrix Factorization et de se mise en pratique au sein de notre projet.

1.3.3 Troisième partie - Evaluation des performances du système de recommandations

Contenu Dans cette section nous exposerons indicateurs de mesures de performances utilisés pour évaluer la qualité de notre système de recommandation, ainsi que nos résultats d'évaluation.

1.3.4 Quatrième partie - Du livrable à l'intégration

Contenu La quatrième partie de ce rapport traite brièvement "l'après projet". Cette partie commencera par une liste des livrables, suivie de détails concernant ces derniers et concernant la manière selon laquelle Sopra Steria pourra intégrer notre module de Data Mining et Machine Learning à leur application.

1.3.5 Cinquième partie - Conclusion & Pistes de réflexion et d'amélioration

Contenu Nous concluons notre rapport par cette cinquième partie, contenant une conclusion, ainsi qu'un certains nombre de propositions d'améliorations, auxquelles nous avons pensé mais que nous ne pensons pas avoir le temps d'implémenter au sein de notre solution technique.

Chapitre 2

Architecture Technique

2.1 Descriptif général de la solution

Introduction Nous allons dans cette section décrire de façon générale les choix qui nous ont guidés au cours du développement de notre solution, et notamment de ses 3 principales "briques" :

1. **Web Service** : Basé sur Django, un framework Python très réputé.
2. **Module de Machine Learning** : Codé en R, étant donné qu'il s'agit d'un des langages les plus adaptés pour les tâches de Data Mining & Machine Learning.
3. **Database** : PostgreSQL, technologie populaire dans le domaine des bases de données.

2.1.1 Web Service

Interface avec SopraHR Le premier choix effectué en terme d'architecture technique de la solution est celui de la développer en tant que Web Service. Cela permettra à Sopra de facilement se "pluguer" à notre module, en communiquant avec ce dernier par requêtes HTTP, en passant des paramètres sous le format de données JSON.

Technology Stack Nous avons choisi d'utiliser Django pour construire cette application Web étant donné que Paul est très familiarisé avec cette *technology stack*. Le module python `Celery` sera utilisé pour la gestion des tâches asynchrones.

2.1.2 Module de Machine Learning

Choix du langage Le module de Machine Learning est développé en `R`, langage très populaire et documenté pour la réalisation de tâches de Data Mining et Machine Learning.

RecommenderLab L'apprentissage du modèle de recommandation, ainsi que la prédiction à partir de ce dernier se font à l'aide de la librairie RecommenderLab de `R`. Nous détaillerons plus loin dans ce rapport tout d'abord le fonctionnement de l'apprentissage et la prédiction d'un point de vue technique, puis dans la partie "approche scientifique" l'aspect scientifique intrinsèque à l'établissement et l'utilisation du modèle.

2.1.3 Base de données

Choix Nous avons choisi PostgreSQL tout d'abord car cette technologie est connue et reconnue, et ensuite parce qu'elle n'est pas payante tout en assurant des performances suffisante pour le cadre de notre étude.

Lien avec la base de données de Sopra Dans le cadre du prototypage de notre application, nous utilisons donc cette base de données (Database en anglais, nous abrégerons "DB") PostgreSQL contenant des données factices que nous avons générées manuellement, mais l'une des hypothèses principales sur lesquelles se fonde notre application est qu'à terme, Sopra "raccordera" notre DB à la DB principale de Sopra, pour que l'application ait accès aux données réelles dont dispose Sopra sur ses utilisateurs afin de pouvoir les utiliser en les "passant" au module de recommandation `R`.

NB Nous détaillerons les étapes de "raccordement" de notre DB à celle de Sopra au sein de la documentation technique qui sera fournie parmi les livrables.

2.2 Schémas récapitulatif de la solution technique

Introduction Afin d'être en mesure de bien se représenter les interactions entre les différents composants de l'application, nous vous fournissons le schéma suivant :

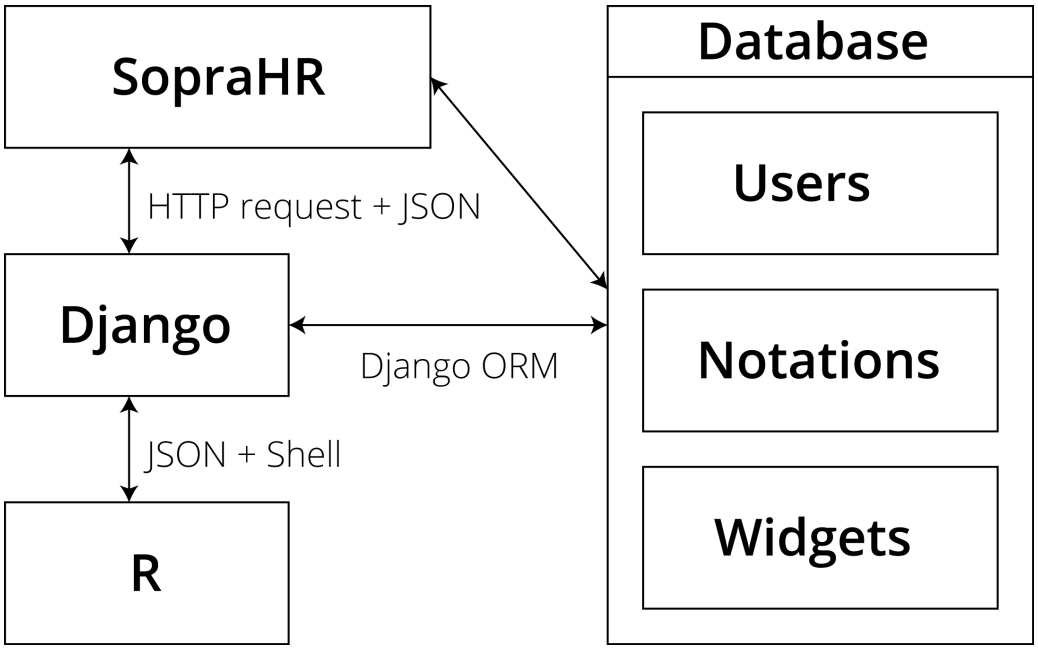


FIGURE 2.1 – Représentation globale des interactions entre les différentes briques du module

Flot apprentissage / mise à jour du modèle Ci-dessous un résumé du flot de mise à jour du modèle de recommandation :

1. Toutes les x heures, Django fait un appel à R pour mettre à jour le modèle de recommandation.
2. Pour cela, Django a au préalable fait une requête la DB pour passer à R les paramètres et données nécessaires à l'apprentissage.
3. R s'exécute et sauvegarde le modèle une fois appris.

Flot recommandation Voici globalement le flot que suivra le système de recommandation :

1. Django reçoit un requête de recommandation venant de SopraHR.
2. Django appelle la DB pour obtenir les informations concernant l'utilisateur sur lequel porte la requête et les transmet à R.
3. R utilise le modèle appris précédemment pour effectuer une prédiction des meilleures recommandations pour cet utilisateur et transmet la réponse à Django.
4. Django sert la réponse à SopraHR.

2.3 Structures de données

Introduction Cette section traite des hypothèses que nous faisons en terme de structures de données. Elle se décompose en deux parties :

1. **Hypothèses initiales** : ressasse les hypothèses que nous avions prises à mi parcours, c'est à dire lorsque nous n'en étions qu'à un stade de formalisation de notre application.
2. **Implémentation technique** : explique comment nous avons implémenté ces structures de données au sein de notre DB une fois le développement de l'application effectivement démarré. L'abandon de la piste "Graded Multilabel Classification" au profit du "Collaborative Filtering" en tant que méthode de Data Mining & Machine Learning a en effet influencé nos choix en terme de DB.

2.3.1 Hypothèses Initiales

Introduction Nous avons dans un premier temps tenté de recenser les données pertinentes dans le cadre de notre projet, sans pour autant passer du temps sur une formalisation "en profondeur" de l'architecture complète de notre DB étant donné que nous attendions la levée d'un certain nombre de zones d'ombre concernant les hypothèse convenant à Sopra. Nous nous sommes alors concentrés sur l'usage que nous souhaitions faire de nos données, afin de mettre en lumière celles qui nous intéressaient. En sont ressortis les usages suivants :

— **Profil utilisateur** : Nous souhaitions que notre ordonnancement de widgets se fasse en fonction de l'utilisateur, donc en quelques sortes à partir d'un "profil utilisateur", c'est à dire en fonction des données que nous aurions eues sur cet utilisateur. Nous souhaitons donc dans ce cas obtenir un maximum d'informations pertinentes sur notre utilisateur.

— **Mesure de l'appréciation d'un widget** Nous souhaitions également que l'ordonnancement des widgets se fasse en fonction de l'appréciation qu'ont les utilisateurs (notamment utilisateurs similaires, c'est à dire ayant un profil similaire à l'utilisateur étudié) des widgets. Nous souhaitions ainsi obtenir une mesure de l'appréciation d'un widget par un utilisateur la plus précise et exacte possible.

Conclusion Nous avions tiré de cela la nécessité de définir les trois structures de données "Profil Utilisateur", "Widget", "Notation" explicitées dans les sous-sections à venir.

Profil utilisateur

Objectif L'un des objectifs principaux de cette structure de données était d'être en capacité de dégager un maximum d'informations caractérisant de manière significative nos utilisateurs.

Sources Deux types de sources de données alimentant un profil utilisateur sont envisageables dans le cas de SopraHR :

— **Profil utilisateur - données RH** : L'utilisateur s'identifiera nécessairement via login / password, et aura un profil d'utilisateur minimum, contenant un certain nombre d'informations personnelles.
— **Weblogs** : L'utilisateur pourra également être caractérisé par certains aspects de son utilisation de l'outil RH.

Représentation Nous avions par la suite défini une représentation potentielle de la structure de données "Profil utilisateur" :

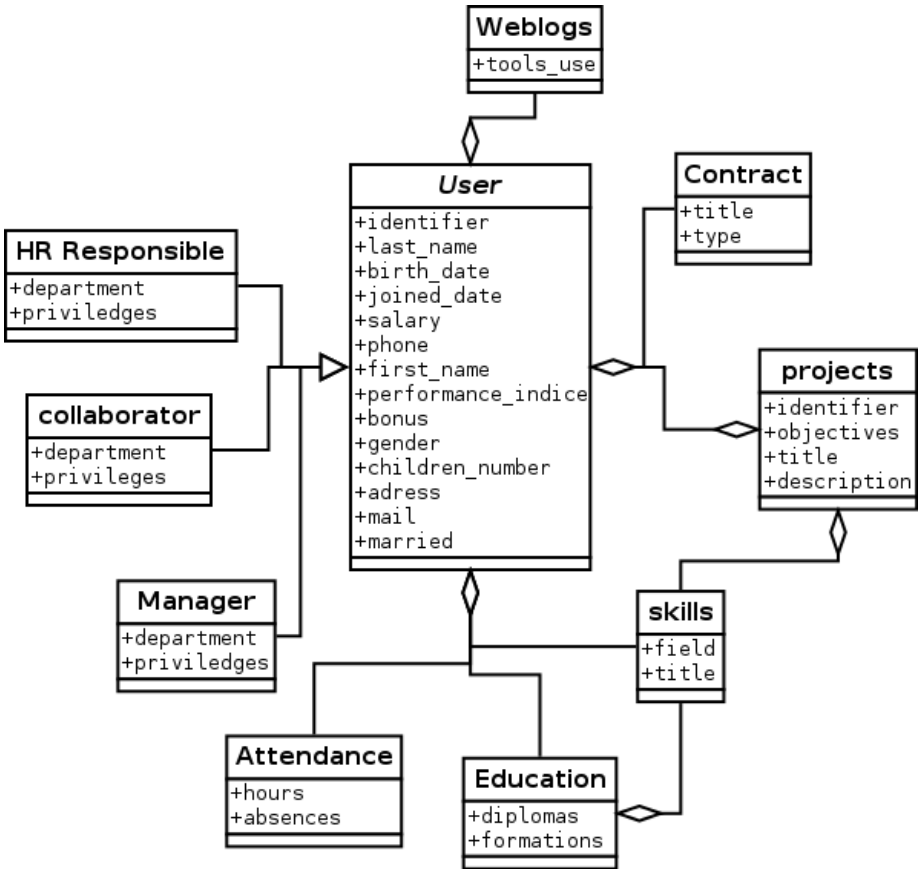


FIGURE 2.2 – Représentation initiale de la structure de données "User"

Précisions Ce modèle est bien évidemment incomplet. Notons notamment le manque de précision des structures de données du côté des weblogs, ou encore au niveau des composants de l'éducation (diplômes et formations).

Appréciation d'un widget

Objectif L'objectif de cette structure de données était de connaitre l'appréciation d'un widget par les utilisateurs afin de déduire de façon la plus précise possible l'appréciation de ce widget par un nouvel utilisateur.

Sources Deux types de sources de données sont envisageables dans le cas de SopraHR :

- **Notation statique par les utilisateurs** : Les visuels de catalogue qui nous ont été fournis laissent penser que les utilisateurs auront la possibilité de noter les différents widgets du catalogue.
- **Weblogs** : Nous jugions nécessaire que l'estimation de l'appréciation d'un widget par un utilisateur ne se limite pas à la notation statique des widgets par les utilisateurs pour les raisons suivantes :
  - Tout d'abord, il est trivial que rien n'empêche à un utilisateur d'attribuer une note ne correspondant pas à son appréciation réelle d'un widget.
  - Ensuite, il est possible que l'appréciation d'un widget par un utilisateur à un moment t ne soit pas la même que son appréciation du même widget à un moment $t + 1$... peu d'utilisateurs dans ce cas prennent le temps de mettre à jour leur note du widget. Ainsi il nous semble que l'appréciation des widget devrait être principalement estimée à partir de l'utilisation qui est faite de ces widget en terme de fréquence d'utilisation, ce à quoi s'ajouterait par la suite, en demi-teinte, la notation des widgets.
  - Enfin, et c'est probablement la raison principale, il est fort probable qu'un grand nombre d'utilisateurs ne prenne pas le temps de noter par eux même les différents widgets qu'ils utilisent. Ainsi nous risquons de nous retrouver avec très peu d'informations explicites concernant l'appréciation des widgets par les utilisateurs.

Représentation Vous trouverez ci-dessous, la représentation potentielle que nous avions émise représentant l'appréciation d'un widget par un utilisateur.

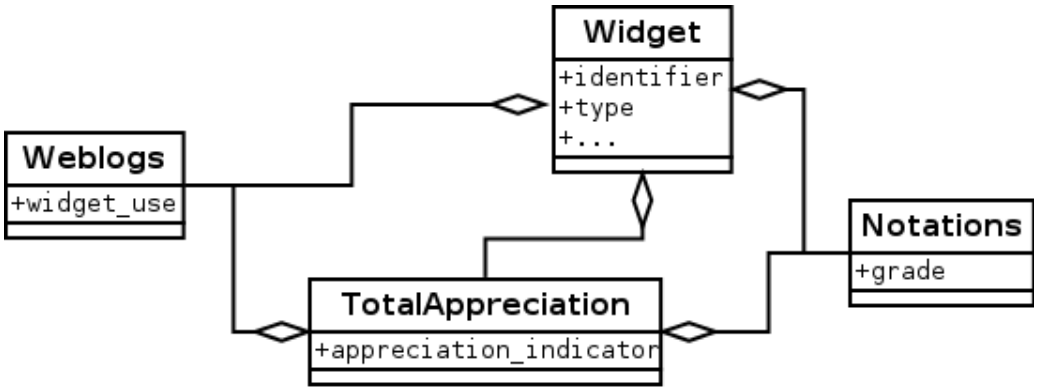


FIGURE 2.3 – Représentation initiale de la structure de données "WidgetAppreciation"

Précisions De nouveau, ce modèle est bien évidemment incomplet. Le concept que nous souhaitions faire resortir ici est le principe d'estimation d'appréciation d'un widget par un utilisateur en prenant en compte à la fois ses statistiques d'usage du widget, ainsi que sa notation statique du widget.

Motivations du choix des structures

Motivations initiales Les deux principales pistes que nous avions trouvées pour la mise en place d'un système de recommandation nous laissaient penser qu'il était nécessaire que nous soyons capables de représenter nos données sous forme d'une matrice similaire à la matrice suivante :

TABLE 2.1 – Matrice de Preference Learning

PL Matrix	X1	X2	...	Xn	P1	P2	...	Pn
U1	14	0	...	12.3	1	5	...	3
U2	11	1	...	11.9	?	0	...	5
...	...	...	...	...	...	...	...	...
Un	12	1	...	13.2	?	5	...	?

Notation Avec Ui les utilisateurs, représentés chacun par un vecteur de caractéristiques $[X1, X2, ...Xn]$; Et Pi les produits (dans notre cas les widgets), pour lesquels nous disposons d'estimations d'appréciation du produit par l'utilisateur, représenté par un "label" numérique allant de 1 à 5, ou " ?" si nous ne disposons pas d'information sur l'appréciation du produit par l'utilisateur.

2.3.2 Implémentation Technique

Introduction Cette section traite notre implémentation effective des structures de données suscitées en base de données PostgreSQL. En effet dans le laps de temps entre la formalisation des structures de données que nous estimions nécessaires et la réalisation de la solution technique, le choix du modèle de recommandation "Collaborative Filtering" nous a permis de savoir quels étaient exactement nos besoins en termes de données.

2.3.3 Influence du choix de modèle de recommandation sur le design de la DB

Choix de modèle de prédiction Nous avons décidé d'implémenter un modèle de prédiction basé sur le "Collaborative Filtering" pour des raisons que nous expliciterons dans la partie suivante. Cette méthode de Machine Learning ne prend pas en input de vecteur de caractéristiques représentant les utilisateurs pour effectuer ses recommandations, ainsi, les données liées aux utilisateurs sont a priori "inutiles". Il en va de même pour les données caractérisant à proprement parler les widgets.

Choix de quantification d'appréciation d'un widget - Notation Nous avons également pris la décision (principalement à cause d'un léger manque de temps), de ne pas complexifier notre évaluation de l'apprécition des widgets, et de nous en tenir à la notation statique attribuée par les utilisateurs comme méthode de base de "rating".

Structures implémentées Notre base de données effectivement implémentée peut donc être représentée de la manière suivante :

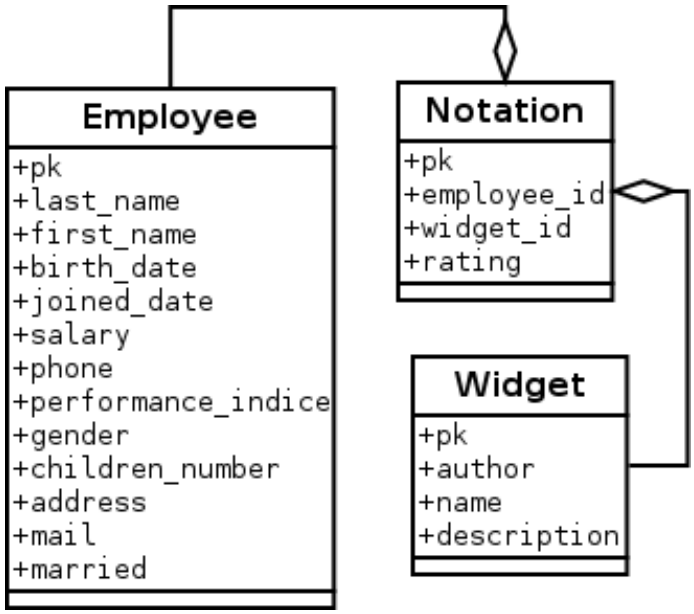


FIGURE 2.4 – Structure de données implémentée en base de données

Commentaires La structure "Notation" en elle même est suffisante à remplir une matrice Utilisateur - Produit - Note (Nous aurions pu limiter la réprésentation des utilisateurs et widgets à leurs *primary keys*), mais nous trouvions dommage de nous borner à cela et ne pas implémenter les structures décrivant un minimum les collaborateurs et les widgets, nous avons donc accompagné "Notation" par "Employee" et "Widget". Notons tout de même que les structures ont été relativement réduites comparées aux structures que nous avions envisagées initialement.

Note `pk` = primary key

2.4 Architecture technique du module

Introduction Cette section s'étend légèrement plus sur plusieurs aspects de l'architecture technique de notre module de recommandation, sans pour autant entrer dans des détails techniques, que nous gardons pour la documentation du module, livrée séparément.

2.4.1 Interactions avec l'existant

Représentation des interactions module - application Voici donc dans un premier temps une représentation de la manière dont notre module de recommandation et l'application SopraHR (ici Soprapp) pré-existante de Sopra Steria peuvent communiquer.

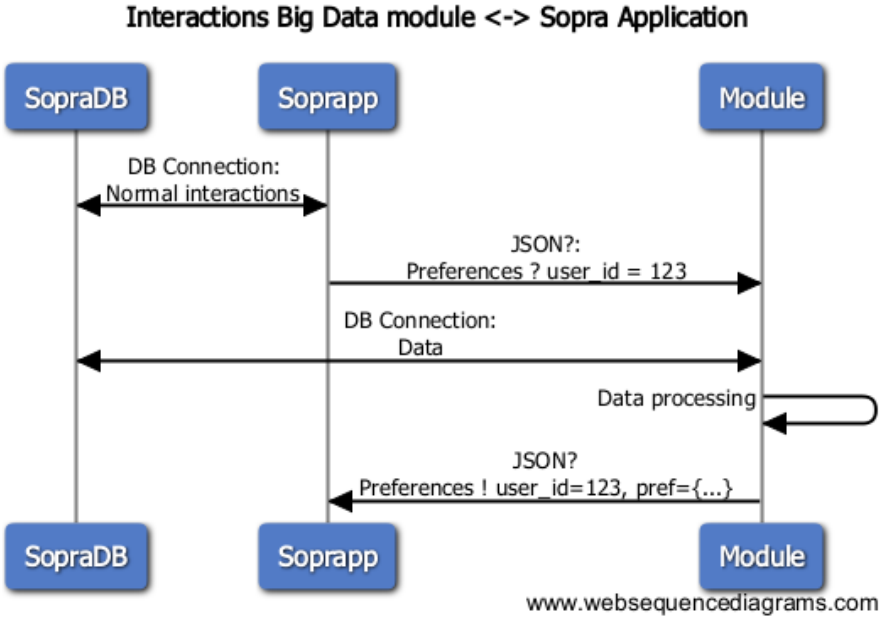


FIGURE 2.5 – Représentation des interactions entre SopraHR et notre module Big Data

Explicitations Le flot est donc, de façon plus détaillée, le suivant :

1. le conceptware de Sopra peut, à tout moment, effectuer une requête HTTP demandant à notre Recommender System une prédiction de recommandations pour l'utilisateur n°123 ;
2. notre module se connecte à la Database afin de déterminer les informations qui lui sont nécessaires concernant l'utilisateur ;
3. notre module traite ces informations afin d'en déduire la recommandation adéquate ;
4. le module répond à la requête HTTP en indiquant au logiciel de Sopra Steria quelles sont les meilleures recommandations pour l'utilisateur considéré.

2.4.2 Interactions internes R <-> Python

Introduction Nous avons décidé d'utiliser `python` pour tout ce qui relève du développement de la partie "web service" de l'application, et `R` pour tout ce qui touche au Data Mining et Machine Learning. En effet, nous avons tout ces deux des connaissances dans les deux langages, et le duo se marie plutôt bien selon la littérature que nous avons pu consulter.

Interactions Python - R Toute phase relevant du Data Mining et du Machine Learning est gérée par `R`, au sein de notre application. Ainsi, il nous a été nécessaire de mettre en place des "passerelles" entre Django (donc python) et R. Après quelques recherches, nous avons donc décidé de "pipeliner" les interactions entre `python` et `R`. Ce type d'interaction peut être représenté comme suit :

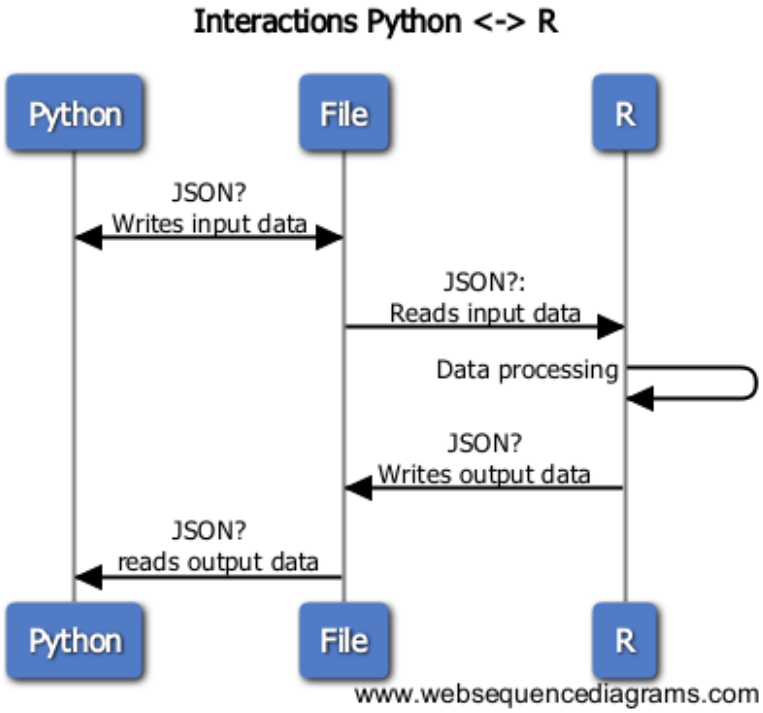


FIGURE 2.6 – Représentation des interactions python et R au sein du module Big Data

Explicitation Ce flot se lit de la manière suivante :

1. `python` "dépose" toutes les données qui vont être nécessaires à `R` au sein d'un fichier d'échange `json` ;
2. `python` déclenche l'execution du script `R` ;
3. le script lit le fichier d'échange, effectue les opérations de Data Mining & Machile Learning nécessaires, puis écrit le résultat de son execution dans un autre fichier d'échange ;
4. une fois le script executé, `python` lit le fichier d'output du script `R` afin de récupérer les données qu'il servira par la suite.

2.4.3 Apprentissage et mise à jour du modèle - Tâches asynchrones

Introduction Notre système de recommandation repose sur l'apprentissage d'un modèle sur lequel se fonde nos prédictions. Il ne nous est pas possible de recalculer le modèle à chaque fois qu'un individu *lambda* ouvre le catalogue de widgets (l'apprentissage d'un modèle est une opération généralement relativement complexe, actualiser notre modèle impliquerait donc un temps de réponse extrêmement long pour l'utilisateur). Nous avons donc décidé de stocker notre modèle et de l'actualiser à intervalle régulier.

Solution technique Le stockage du modèle courant (c'est à dire celui utilisé pour la prédiction à un instant t) se fait, assez facilement via export du modèle courant depuis `R` (sauvegarde .RData). Concernant la réalisation de tâches asynchrones, nous avons utilisé le module `celery` de `python`, que Paul a déjà utilisé au cours de son stage, permettant de programmer des tâches asynchrones d'actualisation du modèle réalisées à intervalle régulier.

2.4.4 Modèle de recommandation et prédiction

RecommenderLab Comme nous l'avions expliqué précédemment, nous basons le calcul de notre modèle de prédiction, ainsi que la réalisation des prédictions elles-mêmes sur une librairie de `R` nommée `RecommenderLab`. Cette librairie permet, en quelques commandes d'effectuer de nombreuses tâches en lien avec le Collaborative Filtering, méthode de Machine Learning à la base de notre système de recommandation. Ainsi, vous trouverez au sein de notre code source (répertoire `RLib` de notre projet) toutes les commandes qui nous permettent d'établir un modèle de prédiction et de le sauvegarder, ainsi que celles qui nous permettent d'effectuer une prédiction, à la base de nos recommandations.

Documentation technique Pour plus de détails techniques, nous vous invitons à lire la documentation technique livrée avec notre solution. Vous trouverez également au sein de cette documentation toute les informations nécessaires à l'utilisation de notre API.

Chapitre 3

Approche scientifique

3.1 Choix de la méthode de Data Mining & Machine Learning

3.1.1 Veille

Introduction Deux méthodes principales de Machine Learning ont initialement émergé de notre travail de veille : la "Graded Multilabel Classification" (abrégée GMC dans le suite de ce rapport), et le Collaborative Filtering (CF). Nous exposons brièvement dans la suite le principe de chacune d'entre elles, puis analysons les avantages de chacune d'elles en fonction de certaines hypothèses.

3.1.2 Graded Multilabel Classification (GMC) - Présentation

Principe La méthode de GMC permet de déterminer l'appréciation par un utilisateur d'un produit, en fonction de l'appréciation du même produit par d'autres utilisateurs relativement similaires à l'utilisateur observé. Les données sont structurées comme suit :

TABLE 3.1 – Matrice de Preference Learning

PL Matrix	X1	X2	...	Xn	P1	P2	...	Pn
U1	14	0	...	12.3	1	5	...	3
U2	11	1	...	11.9	?	0	...	5
...	...	...	...	...	...	...	...	...
Un	12	1	...	13.2	?	5	...	?

Notation Avec Ui les utilisateurs, représentés chacun par un vecteur de caractéristiques $[X1, X2, ...Xn]$; et Pi les produits (dans notre cas les widgets), pour lesquels nous disposons d'estimations d'appréciation du produit par l'utilisateur représenté par un "label" numérique allant de 1 à 5, ou " ?" si nous ne disposons pas d'information sur l'appréciation du produit par l'utilisateur.

Objectif L'objectif de cette méthode est d'apprendre une fonction f, associant à un vecteur représentant un utilisateur $U = \{X1, ..., Xn\}$ un vecteur d'appréciation $Y = \{Y1, ..., Yn\} : f(U) -> Y$. Pour cela, l'apprentissage de f est décomposé en l'apprentissage de fonctions fi associant au vecteur

représentant l'utilisateur les valeurs de chacune des colonnes, individuellement, d'appréciation de widgets pour l'utilisateur en question. L'opération peut être schématisée comme suit :

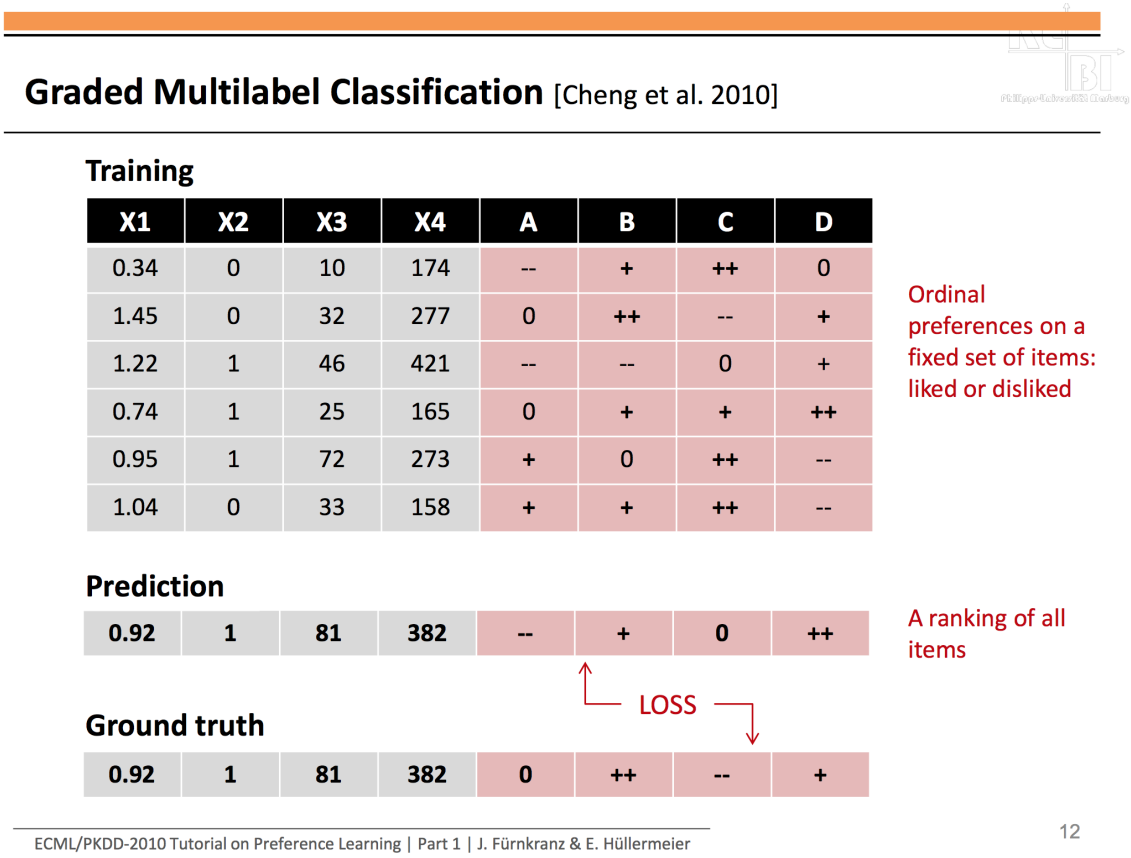


FIGURE 3.1 – La "Graded Multilabel Classification" selon Hullermeier & Furnkranz

Pré-suppositions de la méthode La méthode de GMC est adaptée de préférence à certaines conditions d'entrainement. Généralement, les critères suivants doivent être respectés pour des résultats optimums.

1. **Nombre de produits** : le nombre de produits (partie droite de la matrice de PL) doit être suffisamment restreint (plusieurs dizaines au maximum).
2. **Densité de la matrice de notation** : la matrice des notations (droite) est supposée quasi-pleine.
3. **Densité des caractéristiques utilisateur** : la matrice des caractéristiques utilisateurs (partie gauche de la matrice de PL) est supposée complète.

3.1.3 Collaborative Filtering (CF) - Présentation

Principe La méthode de CF permet elle aussi de déterminer l'appréciation par un utilisateur d'un produit, en fonction de l'appréciation du même produit par d'autres utilisateurs. Les données sont cette fois structurées comme suit :

Table 3.2 – Matrice de Preference Learning

PL Matrix	P1	P2	...	Pn
U1	1	5	...	3
U2	?	0	...	5
...	...	...	...	...
Un	?	5	...	?

Notation Avec Ui les utilisateurs, cette fois ci n'étant pas représentés chacun par un vecteur de caractéristiques ; et Pi les produits (dans notre cas les widgets), pour lesquels nous disposons d'estimations d'appréciation du produit par l'utilisateur représenté par un "label" numérique allant de 1 à 5, ou " ?" si nous ne disposons pas d'information sur l'appréciation du produit par l'utilisateur.

Objectif Différentes méthodes de CF permettent d'atteindre l'objectif final de prédiction d'un vecteur de notations, fonctionnant de manière plus ou moins différentes. Dans la suite nous développerons tout d'abord le "User Based Collaborative Filtering", dont le principe de base consiste à chercher des utilisateurs similaires à l'utilisateur courant afin d'affecter à ce dernier une estimation de son vecteur de notation. La seconde, "Matrix Factorization", va chercher à décomposer et approximer la matrice de notation en matrices moins complexes, c'est à dire appartenant à un espace de dimension plus petites et ainsi la simplifier tout en l'approchant au mieux. Les prédictions seront alors réalisées en multipliant ces matrices simplifiées par notre vecteur représentant les notations de l'utilisateur actif, afin de déduire les valeurs des notations manquantes.

Pré-suppositions de la méthode La méthode de CF est elle aussi adaptée de préférence à certaines conditions d'entrainement. Généralement, les critères suivants doivent être respectés pour des résultats optimums.

1. **Nouvel utilisateur** : il est supposé que les premiers choix proposés à un nouvel utilisateur ne sont pas déterminants : en effet lorsqu'un nouvel utilisateur est ajouté au système, nous ne disposons pas encore d'informations le concernant, ainsi les suggestions qui lui seront proposées risquent d'être, selon la méthode de CF choisie, d'une pertinence limitée.
2. **Nombre de produits** : cette fois ci, le nombre de produits proposés à l'utilisateur peut s'avérer beaucoup plus grand, allant de plusieurs milliers à plusieurs centaines de milliers.
3. **Compléxite d'implémentation** : cette méthode est moins complexe à implémenter (car plus documentée) que la précédente.

3.1.4 Comparaison des méthodes et enjeux du choix

Introduction De ce travail de veille était donc ressorti les deux méthodes exposées ci-dessus, que nous avions par la suite comparées de la manière suivante.

Comparaison Globalement, il apparait que le choix de la méthode est grandement lié à la densité de la matrice de notation Users/Items, la GMC pré-supposant une matrice de notations très dense, tandis que le CF peut très bien fonctionner sur une matrice beaucoup moins fournie. Il apparait également que le choix entre l'une ou l'autre des méthodes dépend grandement du nombre de produits proposés aux utilisateurs, ainsi que du nombre d'utilisateurs. La méthode de GMC a tendance à plus correspondre au cas où le nombre de produits (ici les widgets) proposés aux utilisateur ne dépasse pas plusieurs dizaines, et le nombre d'utilisateurs est restreint. Cette méthode produit également des suggestions plus pertinentes dans le cas de l'arrivée d'un nouvel utilisateur. La seconde méthode, le CF, semble plus adaptée dans le cas d'un "scaling" rapide de l'utilisation de la solution, elle est en effet plus adaptée dans le cas d'un grand nombre d'utilisateurs et de produits. Le nombre de widget étant dans notre cas élevé, suggère potentiellement une matrice assez creuse (manque de notes), ce qui poserait problème pour la méthode de GMC, qui pourrait être en revanche utilisée pour la prédiction des indicateurs et démarches (moins nombreux).

Nouveauté, périodes, campagnes Il a également été évoqué par Sopra Steria la volonté d'intégrer les aspects nouveauté, période, campagne, ... etc au système de recommandation. Ces notions sont difficilement intégrables aux modèles envisagés et nous suggérons que cela ne soit plutôt inscrit en "dur", au sein du post-traitement des résultats de prédiction renvoyés par notre futur système de recommandation. Ces données de nature assez différentes de l'utilisateur, et dont l'effet ne demande pas à être appris par l'observation de données, mais plutôt ajouté aux inférences du modèle n'ont potentiellement pas leur place au sein d'un modèle mathématique de prédiction.

Note Il semble qu'il est également possible d'enrichir la méthode de CF en caractérisant les utilisateurs à l'aide de vecteurs de caractéristiques, comme cela est fait pour la GMC. Cela serait notamment utile lors de l'ajout d'un nouvel utilisateur, car sans notations apparente, on pourrait ainsi comparer les caractéristiques utilisateurs pour suggérer des widgets. Une des autres pistes possible lors de l'ajout d'un nouvel utilisateur serait de lui suggérer les meilleurs widgets du moment, en attendant qu'il n'ajoute et note des widgets.

Litterature Il semble que la méthode de CF soit plus anciennes, et qu'ainsi il existe une plus grosse quantité de litterature à son propos, ce qui peut faciliter notre compréhension de cette dernière et son application. Dans le cas de la GMC, nous avons également trouvé de bonnes bases d'un point de vue littérature, notamment via des présentations et papiers de Eyke Hullermeier et Johannes Furnkranz, références en la matière.

3.1.5 Choix & Motivations

Explications Suite à notre travail de veille, nous avons consulté Sopra Steria ainsi que notre suiveur UTC, M. Destercke, afin d'obtenir certaines informations susceptibles de nous aider dans notre choix de méthode de Data Mining & Machine Learning. Nous avons finalement, au vu des informations qui suivent, opté pour le "Collaborative Filtering". Certains arguments ont en effet retenu notre attention :

1. **Nombre de produits** : Le nombre de produits que compte proposer Sopra Steria dépasse la centaine, ce qui est plus ou moins déjà trop pour la GMC.
2. **Nombre d'utilisateurs** : Idem, le nombre d'utilisateur auquel veut être prêt à faire face Sopra est supérieur à 100 000, ce qui commence à être réellement conséquent pour une méthode aussi complexe que la GMC.
3. **Matrice de Notations quasi-vide** : A part dans le cas où nous aurions interprêté les weblogs pour en ressortir des notations non statiques, la matrice des notations a de grandes chances d'être quasi-vide. Il s'agit là du principal argument nous ayant poussé à éliminer la GMC.
4. **Litterature** Le CF étant moins récent que la GMC, c'est un sujet plus étudié. Il existe par conséquent plus de litterature à son sujet.

Choix final Notre choix d'implémentation s'est donc porté sur le **Collaborative Filtering** (CF).

3.2 Collaborative Filtering - Détails

Introduction Nous allons maintenant nous intéresser plus en détail à deux méthodes de Collaborative Filtering : le "User Based Collaborative Filtering" et la "Matrix Factorization".

3.2.1 User-based Collaborative Filtering

Principe Le User-Based Collaborative Filtering (UBCF), cherche à prédire les valeurs manquantes au sein d'un vecteur de notation d'un utilisateur en particulier (que l'on appellera "utilisateur actif") en déduisant les valeurs de ses composantes à partir des valeurs prises par les composantes des vecteurs de notations d'utilisateurs que l'on estime similaires. L'hypothèse sur laquelle repose le Collaborative Filtering est que des utilisateurs avec des préférences similaires noterons les applications de manière similaire. Ainsi, les notations manquantes pour l'utilisateur actif peuvent être prédites en trouvant tout d'abord un voisinage d'utilisateurs similaires à ce dernier, puis en agrégeant les notations de ces utilisateurs pour former une prédiction.

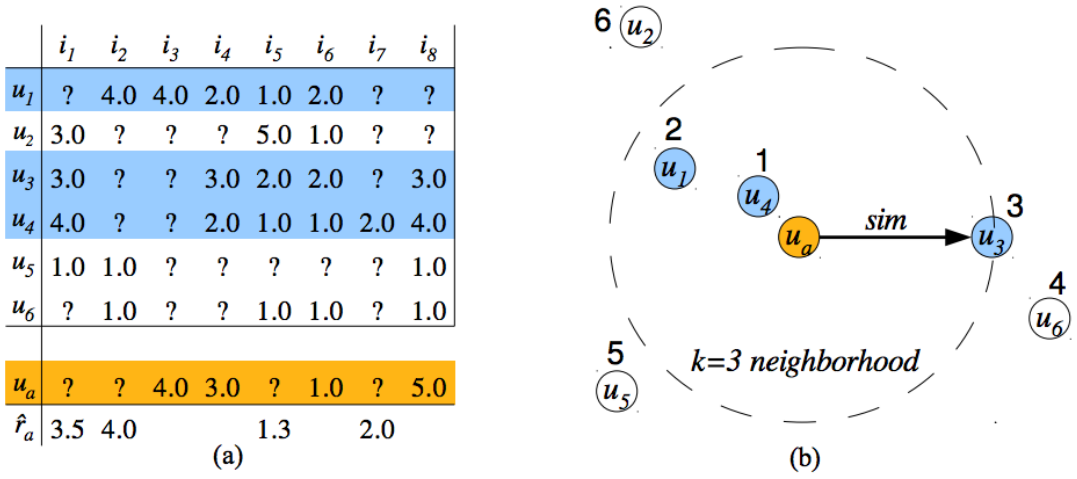


FIGURE 3.2 – Un exemple d'UBCF avec (a) la matrice de notation utilisateurs/items, et (b) la détermination d'un voisinage de l'utilisateur courant

Mesure de similarité Le voisinage est défini en termes de similarité entre utilisateurs, soit en prenant en compte un nombre donné d'utilisateurs similaires (Algorithme des K Plus Proches Voisins (KPPV) ou K Nearest Neighbours en anglais (KNN)), soit en fixant un seuil minimum de similarité. Il est donc nécessaire de déterminer la mesure de similarité que nous utiliserons pour déterminer la similarité entre nos différents vecteurs utilisateurs. Dans notre cas, nous utilisons la *mesure cosinus* permettant de calculer la similarité entre deux vecteurs à n dimensions en déterminant le cosinus de l'angle entre eux. Cette métrique s'obtient par le produit scalaire et la norme des vecteurs :

$$\text{sim}_{Cosine}(\vec{x}, \vec{y}) = \cos(\theta) = \frac{\vec{x}.\vec{y}}{\|\vec{x}\|\|\vec{y}\|}$$

où $\vec{x}$ et $\vec{y}$ représentent deux vecteurs utilisateurs. La mesure de similarité, équivalente à la valeur $\cos(\theta)$ de l'angle entre les deux vecteurs est alors comprise dans l'intervalle [-1,1]. La valeur -1 indiquera des vecteurs résolument opposés, 0 des vecteurs indépendants (orthogonaux) et 1 des vecteurs similaires (colinéaires de coefficient positif). Les valeurs intermédiaires permettent d'évaluer le degré de similarité.

Note Une autre mesure de similarité populaire, que nous ne développerons pas ici est la mesure de similarité de Pearson.

Voisinage Une fois évaluée la similarité entre l'utilisateur actif et les individus d'apprentissage, il est nécessaire de choisir ce que nous allons considérer comme "voisinage" $\mathcal{N}(a)$ de notre utilisateur. Dans le cadre de notre étude, le voisinage de notre utilisateur actif est déterminé par la méthode de k plus proches voisins. Nous avons choisi d'utiliser la valeur par défaut de la librairie `Recommenderlab` $k = 25$ comme paramètre des notre méthode KNN. Cependant, si nous avions eu l'occasion de travailler sur des données réelles, il aurait été nécessaire d'effectuer des tests afin de déterminer la valeur optimale de ce paramètre.

Prédiction Finalement, la prédiction des valeurs manquantes dans le vecteur de notation de l'utilisateur actif se fait en agrégeant la valeur des composantes correspondantes venant des vecteurs de notation des différents utilisateurs appartenant au voisinage de l'utilisateur actif. L'une des méthodes les plus simples pour y parvenir est de simplement en calculer la moyenne :

$$\hat{r}_{aj} = \frac{1}{|\mathcal{N}(a)|} \sum_{i \in \mathcal{N}(a)} r_{ij}$$

avec r_{ij} la notation de l'utilisateur u_i pour l'item i_j. La figure 3.2 présentée ci-dessus nous permet donc de comprendre l'intégralité du processus de prédiction d'une notation manquante au sein du vecteur de notations de u_a, notre utilisateur actif. La notion de voisinage est représentée en (b) sur le plan 2D, et les individus séléctionnés au sein de ce voisinage sont surlignés dans la matrice de notation User / Item en (a). Finalement, les valeurs du vecteur $\hat{r}_a$, en bas de la partie (a) sont prédites en moyennant les valeurs venant des utilisateurs sélectionnés dans le voisinage.

Pondération Le fait que certains utilisateurs du voisinage de l'utilisateur actif sont plus similaires à ce dernier que d'autres peut être incorporé à la prédiction des notations en ajoutant une pondération à la moyenne tel que suit :

$$\hat{r}_{aj} = \frac{1}{\sum_{i \in \mathcal{N}(a)} s_{ai}} \sum_{i \in \mathcal{N}(a)} s_{ai} r_{ij}$$

avec s_{ai} la similarité entre l'utilisateur actif u_a et l'utilisateur u_i du voisinage.

Recommandation finale - TopN Ainsi, ayant prédit les notations manquantes au sein du vecteur de notations de l'utilisateur actif, il est possible d'émettre une recommandation sous la forme d'un "top n items" pour l'utilisateur, en choisissant les n items les mieux notés au sein des notations prédites. C'est ce que nous faisons au sein de notre application, toujours à l'aide de la librairie `RecommenderLab` de R, qui fonctionne exactement de la manière décrite ci-dessus.

3.2.2 Précisions

Normalisation Afin d'améliorer les performances de l'algorithme, il sera utile de normaliser les notes auparavant. Cela permettra de retirer les utilisateurs qui notent toujours les widgets plus bas ou plus haut que les autres utilisateurs. Une des méthodes souvent utilisée consiste à centrer les lignes de la matrice ainsi :

$$h(r_{ui}) = r_{ui} - \bar{r}_u$$

où $\bar{r}_u$ représente la moyenne de toutes les notes disponibles sur la ligne concernée.

Apprentissage vs Prédiction Dans le cas de l'UBCF, l'apprentissage consiste simplement en une mémorisation au sein de notre modèle de `Recommender` de la matrice d'apprentissage. Le réel traitement a lieu au moment de la prédiction, moment auquel va avoir lieu l'application de l'algorithme KNN. Ainsi le gros des calculs a lieu au moment de la prédiction.

3.2.3 Limites & Décision de changement de stratégie

Implémentation initiale Par méconnaissance des possibilités imprésionnantes du réel framework qu'est le `RecommenderLab` et surement une certaine précipitation à l'approche des deadlines, nous avions initialement implémenté la méthode UBCF pour notre système de recommandation.

Limites Cependant, au cours de notre soutenance de TX, notre suiveur Sébastien DESTERCKE a soulevé le problème évoqué ci-dessus : l'algorithme des KNN est complexe, et sa complexité augmente avec le nombre d'utilisateurs. La répartition des temps de calculs penche du côté de la prédiction : le gros des calculs se fait au moment de prédire une recommandation, et non au moment d'un "apprentissage" de modèle.

Changement de stratégie Cette méthode n'était donc pas la méthode la plus adaptée dans le cadre de notre application. C'est pourquoi nous avons étudié plus en détail les possibilités offertes par la librairie `RecommenderLab` et nous sommes aperçus que celle-ci proposait également une implémentation de la méthode de **Collaborative Filtering par Matrix Factorization**. Méthode que nous explicitons donc par la suite.

3.2.4 Collaborative Filtering - Singular Value Décomposition (SVD)

Introduction & Motivations Comme expliqué auparavant, la méthode UBCF nécessite l'application de l'algorithme des KNN à chaque demande de recommandation sur l'ensemble du dataset stocké au sein du modèle de `Recommender UBCF`, ce qui n'est pas sans coût, d'un point de vue complexité. Au sein d'un web service, supposé répondre à des requêtes en un temps raisonnable afin d'assurer une bonne expérience utilisateur, cette méthode n'était pas forcément viable, du fait de son potentiel temps de réponse élevé, lorsque le nombre d'utilisateurs d'apprentissage à examiner est élevé. D'où notre intérêt grandissant pour la méthode de CF par SVD, qui suppose un temps d'apprentissage potentiellement plus long, pour un temps de réponse au moment des demandes de prédictions plus court.

Principe Face à la matrice de notations Users / Items dont les nombres d'utilisateurs et le nombre d'items sont relativement élevés, nous allons chercher à "résumer" cette matrice, c'est à dire tenter de la réduire, la rendre moins complexe, la simplifier, tout en prenant garde à perdre le moins d'information possible.

SVD L'une des méthodes les plus utilisées pour simplifier une matrice est la "Singular Value Decomposition". Toute matrice A peut être décomposée de la manière suivante :

$$A = USV^T$$

avec A de taille `nb_users` x `nb_items`, U de taille `nb_users` x `nb_users`, S matrice rectangulaire diagonale de taille `nb_users` x `nb_items`, et V de taille `nb_items` x `nb_items`.

Intérêt Cela est intéressant car la matrice S contient les valeurs décroissantes de la variance. Ainsi la première valeur diagonale de S capture le plus de variance, la seconde valeur un petit peu moins, et ainsi de suite jusqu'à la dernière valeur diagonale de S (de manière décroissante). Il devient donc possible de "compresser" A de telle sorte à ne garder qu'un certain pourcentage de la variance initiale.

Vecteur propres L'autre intérêt de cette méthode vient du fait que les vecteurs propres obtenus sont orthogonaux. Nous pouvons réécrire l'équation précédente de la façon suivante :

$$A = U\sqrt{S}\sqrt{S}V^T$$

soit :

$$Y = X\Theta^T$$

Obtention de la "SVD tronquée" si X est de taille `nb_users` x `nb_catégories` et Θ est de taille `nb_items` x `nb_catégories`, alors nous obtenons un modèle décent qui approxime A en Y. Y est alors appelé "approximation de bas rang de A" ou "SVD tronquée". X est alors une représentation des utilisateurs dans un espace restreint (dont les dimensions correspondent aux catégories citées précédemment, auxquelles nous ne pouvons donner de signification explicite), et Θ est une représentation des items dans ce même espace de faible dimensions.

Matrices de faible densité Nous pourrions penser que notre simplification de matrice utilisant la SVD est suffisante pour l'obtention de notre modèle, cependant ce n'est pas le cas, car la SVD est destinée aux matrices à fortes densités, or nous avons pris pour hypothèse que la densité de notre matrice est très faible, du fait du faible taux de notation des widgets par les utilisateurs. Il est donc nécessaire d'adapter la formule obtenue précédemment à notre problème. Nous allons ignorer toutes les valeurs non renseignées au moment de résoudre notre modèle, et ainsi poser la formule de coût suivante :

$$J = R(Y - X\Theta^T)^2 + \lambda(\|X\|^2 + \|\Theta\|^2)$$

Ici $Y - X\Theta^T$ est la différence entre les données observées et notre prédiction. R est une matrice de 1 dans le cas où nous avons une valeur et de 0 là où nous n'en avons pas. En effectuant la multiplication de ces deux derniers membres, nous ne considérons donc la fonction de coût que dans le cas où nous avons des valeurs. Nous utilisons une régularisation $L2$ de magnitude λ (cf le concept de régularisation en mathématiques). Nous divisons ensuite par deux afin de faciliter les opérations mathématiques :

$$J = \frac{1}{2}R(Y - X\Theta^T)^2 + \frac{1}{2}\lambda(\|X\|^2 + \|\Theta\|^2)$$

par suite, nous obtenons nos gradients :

$$\frac{\delta}{\delta X} = R(Y - X\Theta^T)\Theta + \lambda\|X\|$$

$$\frac{\delta}{\delta \Theta} = R(Y - X\Theta^T)X + \lambda\|\Theta\|$$

Optimisation L'objectif devient par la suite de minimiser le coût, donc de minimiser la valeur de cette fonction, en appliquant un algorithme d'optimisation. Nous obtenons ainsi un X et un Θ optimaux.

Implémentation Nous n'avons bien entendu pas eu le temps d'implémenter de A à Z cette méthode mathématique d'élaboration d'un modèle de système de recommandation. Heureusement, nous avons découvert que `RecommenderLab` proposait une implémentation du CF par SVD, que nous nous sommes empressés d'implémenter.

Système de recommendation final Notre système de recommendation final utilise donc la méthode de **Collaborative Filtering par Matrix Factorization**, plus particulièrement en appliquant les principes de la SVD pour apprendre un modèle, et effectuant ses prédictions à partir de ce dernier, utilisant une approximation de la matrice initiale pour effectuer les prédictions de notations manquantes.

Chapitre 4

Evaluation des modèles de Recommender

Introduction Nous l'avons vu au cours de notre cursus, le choix d'un bon modèle dans le domaine du Machine Learning passe toujours par l'évaluation de ses performances. Nous avons donc décidé d'évaluer les performances de nos modèles au moyen de différents indicateurs de performances.

4.1 Estimateurs & Indicateurs de performances

Introduction Nous allons au sein de cette section détailler les différents estimateurs et indicateurs de performances utilisés au cours de notre évaluation.

4.1.1 Evaluations des notations prédites

Mean Average Error (MAE) Une façon typique d'évaluer une prédiction est tout d'abord d'évaluer sa déviation par rapport à sa vraie valeur. Pour cela nous calculerons la MAE :

$$\text{MAE} = \frac{1}{|\mathcal{K}|} \sum_{i,j \in \mathcal{K}} |r_{ij} - \hat{r}_{ij}|$$

avec $\mathcal{K}$ l'ensemble de toutes les paires utilisateur - item (i, j), pour lesquelles nous avons prédit une notation $\hat{r}_{ij}$ et dont nous disposons d'une notation connue r_{ij}, n'ayant pas été utilisée pour l'apprentissage du modèle de recommandation.

Root Mean Square Error (RMSE) Une autre mesure populaire est la RMSE, se calculant de la manière suivante :

$$\text{RMSE} = \sqrt{\frac{\sum_{(i,j) \in \mathcal{K}} (r_i j - \hat{r}_{ij})^2}{|\mathcal{K}|}}$$

Cette dernière pénalise de manière plus importante les erreurs importantes que la MAE, et est ainsi très appropriée dans le cas où de petites erreurs de prédiction ne sont pas très importantes (comparées à de grossières erreurs)

4.1.2 Evaluation des recommandations de type "topN"

Dans le cas des recommandations de type "topN", il est possible d'agréger les items retenus par l'utilisateur dans une "matrice de confusion" représentée ci-dessous :

TABLE 4.1 – Matrice de confusion

Réalité / Prédiction	négatif	positif
négatif	a	b
positif	c	d

Explications Cette matrice montre combien des items recommandés dans la liste des topN (colonne "positif", b et d) ont été retenus et ainsi les recommandations correctes (d), et par conséquent les potentielles prédictions erronées (b). Elle montre également les items non recommandés (colonne "négatif", a et c), qu'ils soient retenus (c) ou non (a).

Exploitation De cette matrice, différentes mesures de performances très utilisées peuvent être tirées. Explicitons les.

Exactitude (Accuracy) La première d'entre elle représente la proportion de recommandation juste sur le total de recommandations, il s'agit de l'exactitude (connue sous le nom "accuracy" en anglais) se calculant comme suit :

$$\text{Accuracy} = \frac{a+d}{a+b+c+d}$$

MAE Nous retrouvons également la MAE traitée précédemment, qui se calcule cette fois selon l'expression :

$$\text{MAE} = \frac{b+c}{a+b+c+d}$$

Précision Exposons également la Précision, représentant la proportion de vrais positifs, par rapport au total d'items recommandés :

$$\text{Precision} = \frac{d}{b+d}$$

Rappel Et terminons par la mesure du Rappel ("Recall" en anglais), représentant la proportion de vrais positifs, par rapport au total d'items retenus par l'utilisateur :

$$\text{Recall} = \frac{d}{c+d}$$

4.2 Dataset d'évaluation

Dataset Ne disposant pas de données réelles venant de Sopra Steria, il était nécessaire que nous nous procurions un jeu de données réelles ressemblant un maximum aux structures de données définies dans le cadre de notre étude. Nous avons choisi le jeu de données `MovieLense` du package `RecommenderLab` comportant 100 000 notes allant de 0 à 5 provenant de 943 utilisateurs sur 1664 films. Nous retrouvons ainsi une structure similaire à celle énoncée auparavant dans ce rapport : les films faisant office de widgets, les utilisateurs d'employés, et les notes étant nos instances de notations.

4.3 Résultats

4.3.1 Méthodes & Implémentation

Introduction Nous avons organisé nos évaluations de la manière suivante :

1. **Evaluation des prédictions de notations** : mesure des métriques énoncées précédemment pour les prédictions de notations via nos deux modèles (UBCF, SVD).
2. **Evaluation des recommandations topN** : mesure des métriques énoncées précédemment pour les recommandations de type "topN" via nos deux modèles (UBCF, SVD).
3. **Comparaison des performances en variant modèles et paramètres** : pour finir, nous comparons les performances de nos différents modèles, et des modèles lorsque l'on fait varier certains paramètres.

Partitionnement des données : cross-validation Afin d'évaluer notre modèle, nous allons instancier un `Recommender` à partir d'un ensemble d'individus d'apprentissage et allons le tester sur un ensemble disjoint d'individus de test (`training set` / `testing set`). Plusieurs approches existent pour séparer les données, nous allons utiliser la méthode de `cross-validation` afin de valider notre modèle. La méthode consiste à séparer notre jeu de données en k partitions, puis d'utiliser $k-1$ partitions pour instancier notre modèle que l'on évalue sur la dernière partition. Nous avons décidé (arbitrairement) de fixer la valeur de k à $k = 5$

Implémentation Une nouvelle fois, le module `RecommenderLab` contient de nombreuses fonctions natives nous permettant de calculer et représenter très facilement les différentes métriques d'évaluation de performances !

4.3.2 Résultats

Evaluation des prédictions de notations

Résultats Nous obtenons après calcul les résultats suivants :

TABLE 4.2 – Résultats d'évaluation des performances de prédiction des notations des méthodes UBCF et SVD

Méthode	RMSE	MSE	MAE
UBCF	0.9162346	0.8737574	0.7293243
SVD	0.9461154	0.9312675	0.7622858

Interprétation La méthode UBCF nous donne un taux d'erreur plus faible mais reste relativement similaire à la méthode SVD.

Evaluation des recommandations de type "topN"

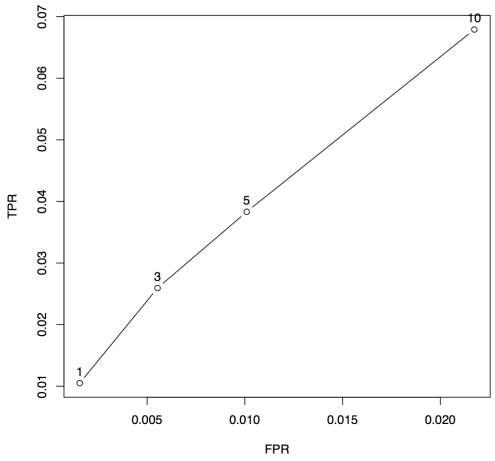
Temps de calcul Intéressons tout d'abord aux temps de calculs moyens d'apprentissage des modèles et de prédictions des deux modèles étudiés (UBCF, SVD) :

TABLE 4.3 – Temps moyen de calcul au cours de l'application de la méthode des 10-folds

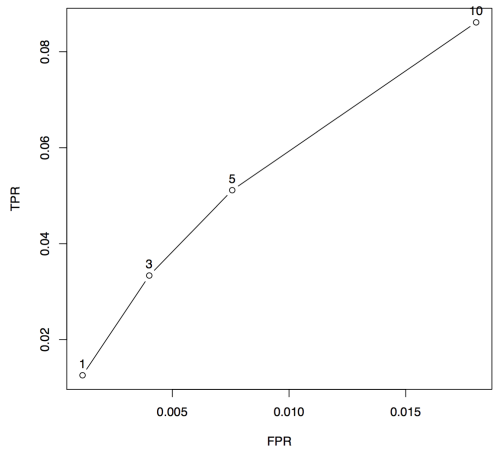
Méthode	Model time	Prediction Time
UBCF	0.0074	0.1342
SVD	0.1039	0.0286

Interprétation Comme cela pouvait se prévoir, le temps d'apprentissage du modèle UBCF est très faible, comparé à celui du modèle SVD. En revanche, l'algorithme KNN utilisé pour la prédiction via le modèle UBCF rend cette dernière très lente (environ dix fois plus lente que la prédiction via le modèle SVD). Le modèle SVD semble ainsi beaucoup plus adapté dans le cadre de notre web service, si nous souhaitons pouvoir être en mesure de délivrer une prédiction quasi instantanée après requête. D'autant plus que le dataset utilisé pour cette cross-validation contient potentiellement beaucoup moins d'utilisateurs que pourront en compter les systèmes de Sopra Steria.

Performances Intéressons nous maintenant aux performances de nos deux `Recommenders` :



(a) Courbe COR, topN, SVD



(b) Courbe COR, topN, UBCF

Interprétation Du point de vue performance, le modèle UBCF obtient des performances légèrement meilleures que celles du modèle SVD, ce qui n'est pas étonnant étant donné que le modèle SVD repose sur une approximation de la matrice de notations. Cependant la différence de performance est tellement faible entre les deux modèles, et le gain en temps de calcul des prédictions est tellement important en choisissant le modèle SVD que tout semble jusque là nous pousser à choisir d'implémenter ce dernier dans notre version de production.

Variation des modèles et paramètres Pour finir, nous souhaitons comparer les performances des deux modèles, en faisant varier leurs paramètres. Voici un tableau récapitulatif des paramètres utilisés :

Table 4.4 – Résultats d'évaluation des performances de prédiction des méthodes UBCF et SVD

UBCF	Method	NN
UBCF_cos_30	Cosine	30
UBCF_cos_50	Cosine	50
UBCF_pear_30	Pearson	30
UBCF_pear_50	Pearson	50
SVD	k	maxiter
SVD_3	3	100
SVD_10	10	100
SVD_10	10	300
SVD_20	20	100
SVD_40	40	100

Résultats En faisant varier les paramètres de la sorte, nous obtenons :

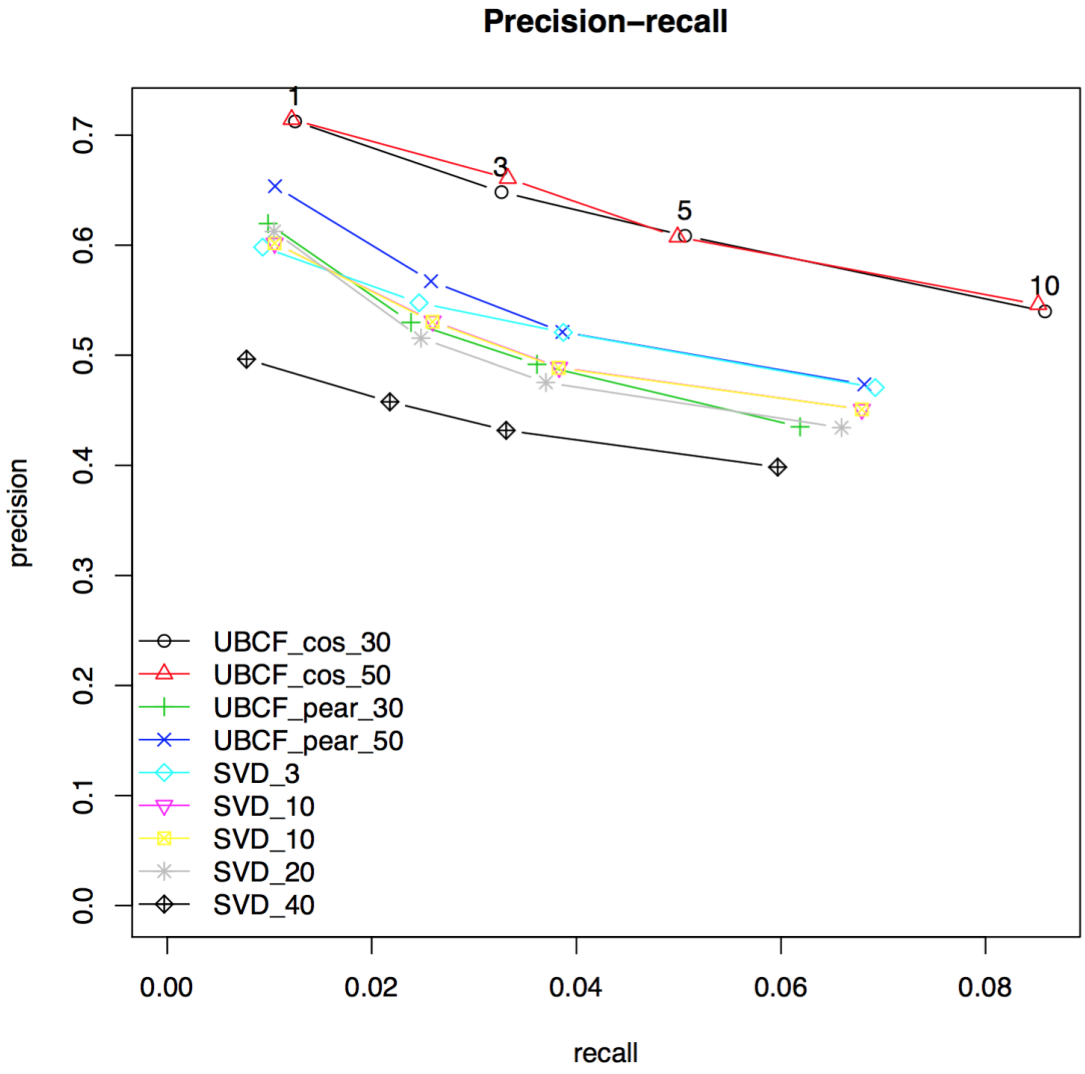


FIGURE 4.2 – Précision vs Recall, variations modèles et paramètres

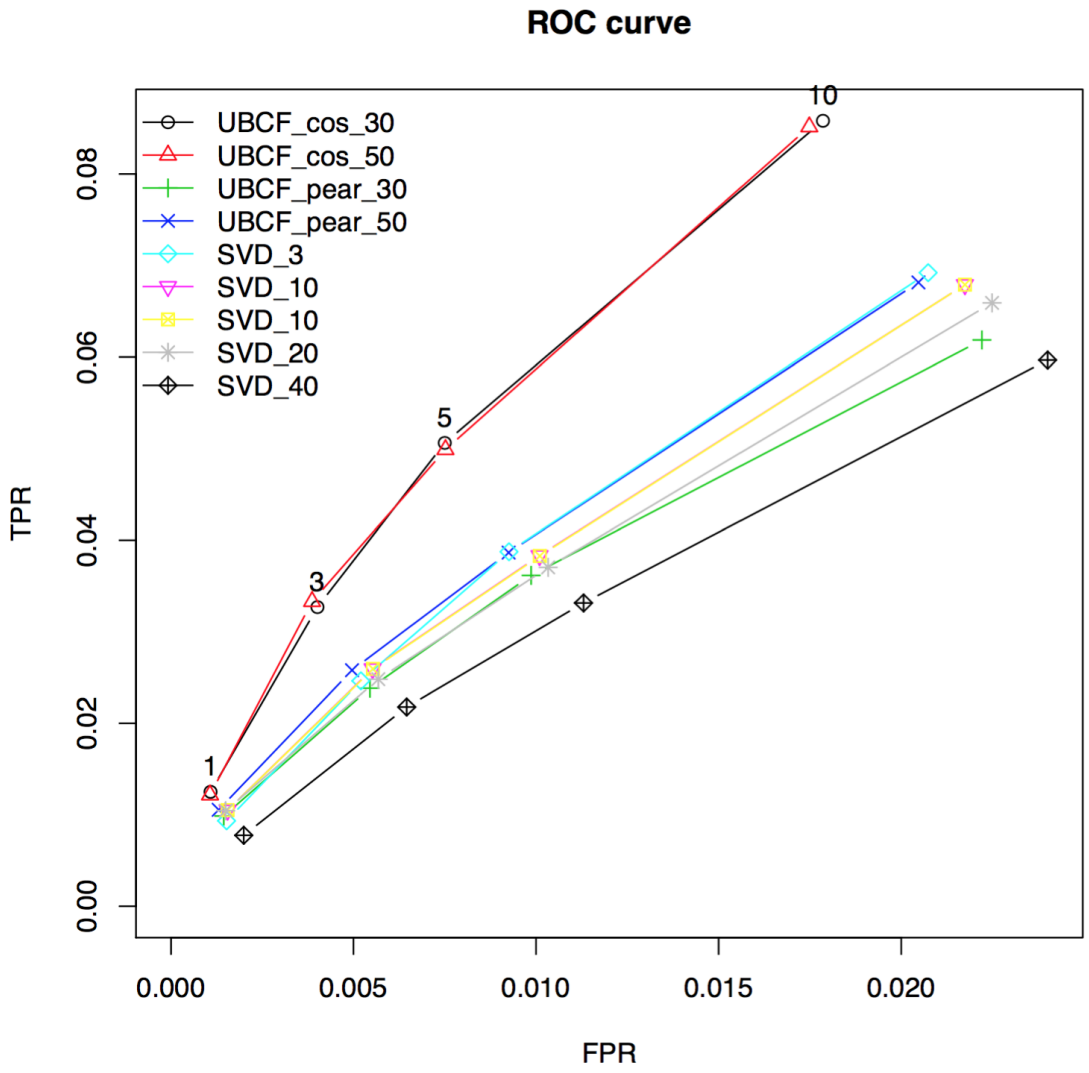


FIGURE 4.3 – Courbe ROC, variations modèles et paramètres

Interprétation Cette évaluation confirme le fait que les performances que nous obtenons via le modèle UBCF sont globalement meilleures que les performances de notre modèle SVD. Cependant, la différence est faible, et le gain en temps de calcul par l'utilisation du modèle SVD est non négligeable. Notons également que les performances de notre modèle SVD semble se dégrader lorsque l'on augmente la valeur du paramètre k. Pour finir, il semblerait que le paramètre `maxiter` du modèle SVD influe assez peu sur ses performances, mais nous pouvons soupçonner que cela est le cas lorsque nous l'augmentons mais pas forcément le cas si justement nous le diminuons.

Conclusion Au vu du gain en temps de calcul que cela apporte, et sachant que les différences de performances sont faibles, nous ferons le choix du modèle SVD pour notre version de production.

Optimisation des paramètres L'évaluation des performances du modèle en faisant varier les paramètres sera à refaire de façon beaucoup plus rigoureuse lors de l'intégration du module au logiciel de `Sopra Steria`. En effet le jeu de données ne sera pas le même, et il est fort probable que les paramètres optimums pour le jeu de données `MovieLense` ne le soient pas pour les données de `Sopra`.

Chapitre 5

Du livrable à l'intégration

Introduction Ce chapitre aborde le futur, détaillant dans un premier temps les livrables, et dans un second temps la marche à suivre pour le cas où `Sopra Steria` souhaiterait intégrer notre travail à son logiciel `SopraHR`

5.1 Livrable

Détail des produits Nous comptons rendre, en fin de projet :

- **Rapport de projet** : Le compte rendu de notre projet, destiné à Sopra Steria ainsi qu'à l'équipe encadrante de l'UTC. Ce rapport est avant tout un bilan de ce qui a été réalisé au cours du semestre, mettant en avant l'évolution de notre travail ainsi que le fonctionnement global de la solution finale livrée.
- **Code source** : Le code source de notre solution technique, hébergé sur un repo github.
- **Documentation technique** : Une documentation technique de notre code, principalement à destination de `Sopra Steria`, permettant à l'entreprise de prendre possession de notre code et de l'adapter à sa guise au software existant. Cette documentation technique fera aussi office de guide d'intégration, étant donné que la marche à suivre pour l'adaptation de notre code y sera détaillée.
- **Vidéo de démonstration de la solution** : Une vidéo de démonstration de la solution livrée, permettant de se faire une bonne idée d'ensemble du fonctionnement de notre solution.

5.2 Intégration

Guide d'intégration Comme précisé auparavant, un guide d'installation et d'intégration guidera `Sopra Steria` dans les étapes d'adaptation du code source de notre solution à l'existant et l'intégration de la solution à leur software `SopraHR`. Ce guide sera intégré à la documentation technique livrée en fin de projet.

Chapitre 6

Conclusion & Pistes d'amélioration

6.1 Conclusion

Réalisations Ce projet a été l'occasion pour nous de mettre en application les connaissances acquises au cours de notre cursus en génie informatique et plus particulièrement en filière "Fouille de données" de l'UTC afin de réaliser ce projet visant l'intégration des méthodes de Data Mining et de Machine Learning à une interface auto-adaptative d'une application HR de `Sopra Steria`. Nous avons ainsi réalisé une solution informatique complète, touchant différent domaines de compétence de l'ingénieur informatique :

- **Applications web et web services** : via la réalisation technique complète d'un web service sous forme d'API, à l'aide de frameworks de référence.
- **Gestion de base de données** : via le design et l'implémentation de la base de données de l'application web sus-citée.
- **Data Mining & Machine Learning** : via un travail de veille technologique, l'étude de différentes techniques de Machine Learning suivi du choix et de l'implémentation de l'une d'entre elles au sein de notre projet
- **Gestion de projet** : via l'inévitable nécessité de gérer correctement un projet qui s'étale sur un semestre entier, en lien avec différents interlocuteurs, et pour lequel il est nécessaire de faire face aux différents aléas potentiels

Ainsi, par la largeur du spectre d'activités réalisées, nous avons eu la satisfaction de travailler sur un projet complet, et relativement exhaustif, si l'on le met en perspective avec ce que nous pourrons rencontrer plus tard, au cours de notre vie professionnelle.

6.2 Pistes d'améliorations

Continuité Nous n'avons bien évidemment pas eu le temps, en un semestre, de traiter autant d'aspects du projet que nous ne l'aurions souhaité, et avons du faire le choix d'abandonner certaines des idées que nous avions eu initialement, faute de temps. Nous exposerons brièvement dans cette section quelques pistes d'amélioration de notre projet, dans l'hypothèse où celui-ci serait continué au cours des semestres à venir.

Exploitation des données de Sopra Steria La première des choses sur lesquelles nous aurions aimé mettre l'accent est l'exploitation de données venant de Sopra Steria. Nous sommes certains

que lorsque le conceptware aura mûri du côté de Sopra, l'entreprise sera en mesure de fournir une quantité plus importante de données qui pourront être utilisées dans le futur afin d'apporter de nombreuses nouvelles fonctionnalités "intelligentes" à leur software.

Optimisation du modèle de système de recommandation Nous nous sommes concentrés sur notre objectif de livrer en fin de semestre une solution en état de marche, fonctionnelle, à `Sopra Steria`. Nous n'avons finalement eu qu'assez peu de temps pour optimiser la manière dont nous paramétrons et utilisons notre modèle de prédiction. Nous sommes persuadés qu'il sera possible, à l'avenir, de l'améliorer, non seulement en effectuant des tests approfondis d'évaluation des performances des `Recommenders`, mais aussi en effectuant un réel travail d'optimisation à l'aide des données réelles.

Un système de recommandation générique Nous nous sommes concentrés au cours de notre projet sur une solution traitant un unique cas d'usage. En réalité, cette solution permettrait de traiter de nombreux cas d'usage dans lesquels un système de recommandation est requis. Notre dernière piste d'amélioration concerne donc cet aspect du problème : notre système de recommandation pourrait être élargi et adapté à de nombreuses situations en une quantité assez limitée d'opérations. Cela ouvre un certain nombre de perspective pour l'avenir de notre projet.

En conclusion Beaucoup peut être encore fait, et nous serions fier que notre projet puisse être repris et amélioré à l'avenir.