# Projet de recherche :

# Moteur de recommandations

# de séries par analyse de sous-titres

Équipe : Taleb Rida, Jday Achraf, Serraf Dan

Encadrants : Baskiotis Nicolas, Guigue Vincent

# Sommaire

# 1. Introduction

## 1.1 Contexte

Au cours des dernières décennies, avec l'essor de Youtube, Amazon, Netflix et de nombreux autres services web de ce type, les systèmes de recommandation ont pris de plus en plus de place dans nos vies. Du commerce électronique (suggérer aux acheteurs des articles susceptibles de les intéresser) à la publicité en ligne (suggérer aux utilisateurs les bons contenus, correspondant à leurs préférences), les systèmes de recommandation sont aujourd'hui incontournables dans nos parcours quotidiens en ligne. D'une manière très générale, les systèmes de recommandation sont des algorithmes visant à suggérer des éléments pertinents aux utilisateurs (éléments pouvant être des films à regarder, des textes à lire, des produits à acheter ou tout autre élément selon les secteurs). Les systèmes de recommandation sont très importants dans certains secteurs, car ils peuvent générer des revenus considérables lorsqu'ils sont efficaces ou constituer un moyen de se démarquer de manière significative de la concurrence. Comme preuve de l'importance des systèmes de recommandation, nous pouvons mentionner qu'il y a quelques années, Netflix a organisé un défi (le "prix Netflix") dont l'objectif était de produire un système de recommandation plus performant que son propre algorithme avec un prix d'un million de dollars à gagner. Les algorithmes de recommandation se divisent en deux grandes familles : les algorithmes de filtrage collaboratif, basés sur la recommandation de produits appréciés par les personnes qui aiment les mêmes choses que la personne cible et les algorithmes basés sur le contenu, qui recommandent des choses proches des items déjà visités.

## 1.2 Objectif

Parmi l'ensemble des secteurs auxquels on peut appliquer des systèmes de recommandations nous nous intéresserons à l'analyse de sous-titre. L'objectif de ce projet est de réaliser un moteur de recommandation pour séries à partir d'analyse de corpus de sous-titres et de notes utilisateurs. Pour y parvenir, nous organiserons le projet en plusieurs étapes distinctes. On commencera par récupérer l'ensemble des informations nécessaires à l'apprentissage de nos futurs modèles. La première étape consiste à compléter nos données. En effet la base de données de sous titres nous a été fourni. Toutefois les notes d'utilisateurs sur les différentes séries ne nous ont pas été donné. Il faudrait que nous réalisions un scrapping sur internet afin de pouvoir extraire ces données pour la suite. En second lieu il faut nettoyer l'ensemble de nos données. Les sous titres que nous utiliserons tout au long de ce projet sont au format texte. Afin de pouvoir faire fonctionner nos différents algorithmes d'apprentissage il va falloir réaliser un prétraitement sur ces textes afin de pouvoir nettoyer ces derniers. En ce qui concerne les notes d'utilisateurs, nous récupérerons nous même ces informations ils nécessiteront alors aucun pré-traitement. Une fois notre base d'apprentissage prête à l'emploi, nous allons explorer nos données afin de mieux les comprendre.

A cette étape du projet nous allons appliquer des fonctions sur l'ensemble dans le but de trouver un sens et un lien entre les différentes données. Et faire différents affichages graphiques pour les visualiser et mettre en évidence leur répartition et supprimer notamment des informations qui nous semblent inutiles pour la suite. Ensuite nous allons construire différents algorithmes de recommandation pour pouvoir analyser nos données. Dans le cadre de ce projet nous avons décidé de nous attarder sur les algorithmes de filtrage collaboratif. Il est important à ce niveau de mettre en place différentes fonctions qui vont nous permettre d'une part d'évaluer nos données et d'autre part de pouvoir comparer les résultats et pouvoir sélectionner les stratégies de recommandation les plus pertinentes. La dernière étape consiste à faire une synthèse des meilleurs résultats de nos algorithmes et de les appliquer afin de mettre en place l'implémentation du moteur de recommandation qu'on appliquera aux corpus de sous-titres des séries.

## 1.3 Problématique.

Maintenant que nous avons vu l'immense valeur que les entreprises peuvent tirer des systèmes de recommandations, examinons le type de problème qu'elles peuvent résoudre. De manière générale, les entreprises technologiques essaient de recommander le contenu le plus pertinent à leurs utilisateurs. La formulation du problème est ici critique. La plupart du temps, les entreprises souhaitent recommander du contenu que les utilisateurs sont les plus susceptibles d'apprécier à l'avenir.

Nous allons explorer ces différentes stratégies et leur hybridation sur la recommandation de séries en exploitant une base de sous-titres en guise de contenu et un ensemble d'avis d'utilisateurs.

# 2. Architecture générale

## 2.1 Descriptions des technologies

**Pandas** est un paquetage Python qui fournit des structures de données rapides, flexibles et expressives conçues pour rendre le travail avec des données structurées (tabulaires, multidimensionnelles, potentiellement hétérogènes) et des séries temporelles à la fois facile et intuitif. Il a pour but d'être la brique de base de haut niveau pour effectuer des analyses de données pratiques et réelles en Python. En outre, il a pour objectif plus large de devenir l'outil d'analyse et de manipulation de données open source le plus puissant et le plus flexible disponible dans n'importe quel langage. Il est déjà en bonne voie pour atteindre cet objectif.

Les deux principales structures de données de pandas, Series (1-dimensionnelle) et DataFrame (2-dimensionnelle), gèrent la grande majorité des cas d'utilisation typiques en finance, statistiques, sciences sociales et de nombreux domaines de l'ingénierie. Pour les utilisateurs de R, DataFrame fournit tout ce que le data.frame de R fournit et bien plus encore. pandas est construit au-dessus de NumPy et est destiné à bien s'intégrer dans un environnement de calcul scientifique avec de nombreuses autres bibliothèques tierces.

**NumPy** est le paquetage fondamental pour le calcul scientifique en Python. Il s'agit d'une bibliothèque Python qui fournit un objet tableau multidimensionnel, divers objets dérivés (tels que les tableaux masqués et les matrices) et un assortiment de routines permettant d'effectuer des opérations rapides sur les tableaux, notamment des opérations mathématiques, logiques, de manipulation de formes, de tri, de sélection, d'E/S, de transformées de Fourier discrètes, d'algèbre linéaire de base, d'opérations statistiques de base, de simulation aléatoire et bien plus encore. Au cœur du paquetage NumPy se trouve l'objet ndarray. Il encapsule des tableaux à n dimensions de types de données homogènes, de nombreuses opérations étant effectuées en code compilé pour des raisons de performances. Il existe plusieurs différences importantes entre les tableaux NumPy et les tableaux Python standard :

- Les tableaux NumPy ont une taille fixe à la création, contrairement aux listes Python (qui peuvent croître dynamiquement). Changer la taille d'un tableau NumPy va créer un nouveau tableau et supprimer l'original.

- Les éléments d'un tableau NumPy doivent tous être du même type de données, et auront donc la même taille en mémoire. L'exception : on peut avoir des tableaux d'objets (Python, y compris NumPy), permettant ainsi des tableaux d'éléments de tailles différentes.

- Les tableaux NumPy facilitent les opérations mathématiques avancées et d'autres types d'opérations sur de grands nombres de données. En général, ces opérations sont exécutées plus efficacement et avec moins de code que ne le permettent les séquences intégrées de Python.

**Surprise** est un scikit Python pour construire et analyser des systèmes de recommandation qui traitent des données de notation explicites. Surprise a été conçu avec les objectifs suivants :

- Donner aux utilisateurs un contrôle parfait sur leurs expériences. A cette fin, un fort accent est mis sur la documentation, que nous avons essayé de rendre aussi claire et précise que possible en pointant chaque détail des algorithmes.

- Alléger la douleur de la manipulation des jeux de données. Les utilisateurs peuvent utiliser à la fois des jeux de données intégrés (Movielens, Jester) et leurs propres jeux de données personnalisés.

- Fournir divers algorithmes de prédiction prêts à l'emploi, tels que les algorithmes de base, les méthodes de voisinage, les algorithmes basés sur la factorisation matricielle (SVD, PMF, SVD++, NMF), et bien d'autres. De plus, diverses mesures de similarité (cosinus, MSD, pearson...) sont intégrées.

- Fournir des outils pour évaluer, analyser et comparer les performances des algorithmes. Les procédures de validation croisée peuvent être exécutées très facilement en utilisant de puissants itérateurs CV (inspirés des excellents outils de Scikit-learn), ainsi que la recherche exhaustive sur un ensemble de paramètres.

- **Matplotlib** est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy. Plusieurs points rendent cette bibliothèque intéressante :

- Export possible en de nombreux formats matriciels (PNG, JPEG...) et vectoriels (PDF, SVG...)

- Interface pylab : reproduit fidèlement la syntaxe MATLAB

- Bibliothèque haut niveau : idéale pour le calcul interactif

## 2.2 Nature des données

Nous allons présenter les différents types de données que nous rencontrerons dans ce projet. Les fichiers textes correspondants aux sous titres et les notes d'utilisateurs par rapport aux séries.

**Analyse sous titres**

Tous d'abord nous commencerons par l'analyse des sous titres. Les sous titres que nous utiliserons tout au long de ce projet sont au format texte. Afin de pouvoir faire fonctionner nos différents algorithmes d'apprentissage il va falloir réaliser un prétraitement sur ces textes. Il faudrait toutefois que l'ensemble de notre processus de pré-traitement soit modulable afin de pouvoir sélectionner les différentes étapes du processus sans pour autant devoir modifier notre projet. Les fichiers qui contiennent les sous-titres recueillis ne sont pas formater et nécessite un pré-traitement afin de supprimer toutes les informations qui nous semblent inutiles à l'apprentissage. Il y a 5 étapes principales à appliquer dans le prétraitement d'un texte.

Ce processus de pré-traitement commence par la tokenisation du texte. La tokenisation est une étape indispensable. En effet pour que nos algorithmes comprennent nos textes il est nécessaire de le décomposer de manière anatomique en ne conservant que des mots. Pour résumer la tokenisations consiste à diviser des chaines de texte en morceaux plus petits. Ainsi les paragraphes peuvent être tokenisés en phrases et les phrases peuvent être tokenisés en mots. Ensuite il faut normaliser les données. La normalisation consiste à mettre tout le texte sur un pied d'égalité. Par exemple on convertira tous les caractères en minuscules, on convertira les caractères accentués en caractère ASCII, on remplacera la contraction des mots et enfin on supprimera tous les caractères spéciaux. En effet si nous ne réalisons pas ces différentes modifications nos algorithmes considérerons chaque variante de mot comme des mots différents.

Suite à cela nous pouvons essayer d'affiner nos données en les lemmatisant. Lemmatiser consiste à garder la forme canonique d'un mot. Cela permettra de garder notamment le radical d'un mot en supprimant notamment les préfixes et suffixes ou bien encore les formes de grammaire tel que le pluriels ou le féminin. La dernière étape consiste à nettoyer le texte en supprimant les espaces blancs ainsi qu'une liste de mot courant qui n'apportera aucune information à l'apprentissage. Cette liste de mot s'appelle « Stopwords » ce sont des mots communs nous permettant pas d'indexer ou de les utiliser dans une recherche afin d'extraire des informations d'un texte.

**Notes utilisateurs**

Les seconds types de données que nous traiterons sont les notes que des utilisateurs ont fait sur différentes séries. A la différence des sous titres, les notes d'utilisateurs sur les différentes séries n'ont pas été fourni. Il a donc fallu les recueillir nous-même sur internet.La première étape consiste à créer un automatisme de recherche sur internet (un Bot) qui trouve la série demandé (son identifiant) à partir de son nom sur le site de IMDB en envoyant une requête Get sur web vers leur site. Ensuite il faut acquérir les données de la page de la série sur IMDB sans besoin d'API en utilisant une technique d'analyse syntaxique (Parsing), un analyseur syntaxique est un composant logiciel qui prend des données d'entrée (souvent du texte) et construit une structure de données - souvent une sorte d'arbre d'analyse, un arbre syntaxique abstrait ou une autre structure hiérarchique, donnant une représentation structurelle de l'entrée tout en vérifiant la syntaxe correcte. Dans notre cas, les données sont du code HTML. Puis on doit fouiller les données HTML pour trouver les notes de chaque épisode et les notes de chaque saison en utilisant une technique de recherche similaire au Parsing que nous avons utilisé tout au début qui consiste à trouver ces données à partir du contenu de la page en se basant sur le fait que avant chaque avis d'utilisateur dans le HTML il y a une balise 'div' qui nous indique que c'est une nouvelle section (/ division) du code et juste après cette division il y a le titre 'rating' c'est-à-dire les avis/les notes des utilisateurs sur IMDB pour cette série. Enfin on rend ces données utilisables en modifiant leurs types du simple code HTML chaines de caractères vers des flottants pour les insérer dans des listes qui vont servir comme notre base de données sur les avis des utilisateurs IMDB.

Pour terminer, on réalise des statistiques sur le contenu de la base de données (moyenne des avis, pire(/meilleure) épisode/saison, ligne de tendance, comparaisons entre les épisodes les saisons et les séries, trouver des Trends sur l'ensemble des séries en se basant sur les avis des utilisateurs et les dates de diffusion, etc..), et finalement dessiner nos données sur un graphe pour bien visualiser les tendances.

# 3. Etude des algorithmes de recommandations

## 3.1 Difficultés liées à la construction d'un système de recommandation

Construire un système de recommandations est une tâche qui peut s'avérer particulièrement complexe pour les entreprises qui n'ont ni les mêmes besoins, ni les mêmes objectifs. Malgré ces différences, certains problèmes sont récurrents à la majorité des systèmes de recommandations et c'est ce que nous allons voir dans cette partie.

**Sérendipité**

La sérendipité désigne la faculté à faire des découvertes heureuses par accident. S'il peut être plus facile de rester dans sa zone de confort idéologique, il y a des avantages à embrasser le hasard de cette manière. Que cela nous plaise ou non, nous nous baignons réellement dans le hasard tous les jours. Par exemple, à la maison lorsque vous recherchez un objet spécifique pour se retrouver face à face avec un objet précédemment perdu. Ou le soir quand on cherche un ami, mais qu'on finit par en trouver un autre avec qui la discussion s'avère ennuyeuse. Le hasard n'est pas synonyme de découvertes heureuses mais peut produire des résultats à la fois négatifs et positifs. Par conséquent, un algorithme de recommandations efficace devrait non seulement recommander ce que nous sommes susceptibles d'apprécier, mais aussi suggérer des éléments aléatoires tout en étant objectifs pour nous aider à garder une fenêtre ouverte sur d'autres mondes et de nouvelles découvertes.

Bien qu'étant une qualité importante, le manque de sérendipité est un syndrome courant dans les systèmes de recommandations actuels qui privilégient souvent l'engagement à tout prix et préfèrent ne prendre aucun risque de décevoir l'utilisateur.

**Sparsité**

Les utilisateurs d'une application ne vont généralement interagir qu'avec un pourcentage très faible du catalogue d'items proposé. Ces données une fois représentées sous forme de matrices ont un très fort degré d'interactions nulles. Ce genre de matrices est appelé matrice creuse et les algorithmes de recommandations collaboratives réalisent des calculs sur base de celles-ci. Or quand on veut manipuler ou stocker des matrices creuses à l'aide de programmes informatiques, il est avantageux, voire souvent nécessaire d'utiliser des algorithmes et des structures de données qui prennent en compte la structure éparse de ces matrices. Les opérations sur les matrices sont lentes et utilisent beaucoup de mémoire.

Les matrices creuses ont néanmoins l'avantage de pouvoir être facilement compressibles de par leur sparsité. La structure de données naïve utilisée pour stocker une matrice est un tableau bidimensionnel où chaque entrée du tableau représente un élément de la matrice. Pour une matrice m x n il faut au moins m x n espaces mémoires de taille fixe pour la représenter. Beaucoup, si ce n'est la majorité des entrées d'une matrice creuse, sont des zéros ou des valeurs nulles. L'idée de base est alors de ne stocker que les entrées non nulles de la matrice, plutôt que d'en stocker l'intégralité. En fonction du nombre et de la répartition des entrées non nulles, des structures de données différentes peuvent être utilisées et amènent de grandes économies dans la taille utilisée en mémoire par rapport à la structure naïve. Un exemple d'une telle représentation est le format "Yale Sparse Matrix" qui stocke une matrice de taille m x n sous la forme de trois tableaux unidimensionnels.

**Démarrage à froid**

**Utilisateurs :**

Le problème du démarrage à froid est souvent observé dans les systèmes de recommandations où les méthodes telles que le filtrage collaboratif reposent fortement sur les interactions utilisateur-item antérieures. Les entreprises sont confrontées au problème du démarrage à froid de deux manières selon les plateformes : démarrage à froid par l'utilisateur et par item. Lorsqu'un nouveau membre s'inscrit par exemple sur Netflix l'entreprise ne sait rien des préférences de ce nouveau membre.

De nombreuses plateformes telles que Netflix, Medium ou Pinterest vous demandent de passer par une étape de sélection de vos intérêts avant de pouvoir utiliser l'application. Cet effort actif demandé à l'utilisateur permet de résoudre en partie le problème du démarrage à froid. Les applications font néanmoins attention à ce genre d'effort qui peut augmenter le taux d'abandon lors du processus d'inscription, ce qui explique pourquoi cette étape vient généralement en dernier lieu.

**Items :**

Les entreprises sont confrontées à un défi similaire lorsque de nouveaux articles ou contenus sont ajoutés au catalogue. Des plateformes comme Netflix ou Prime Video contiennent un catalogue qui change "peu" fréquemment (il faut du temps pour créer des films ou des séries). Par conséquent, elles ont moins de difficultés comparées à Airbnb ou Zillow où de nouvelles annonces sont créées chaque jour. A ce moment-là, elles n'ont pas de possibilité d'apparaître car elles n'étaient pas présentes pendant le processus d'entraînement du système de recommandations. Airbnb résout ce problème de la manière suivante : "Pour créer des incorporations pour une nouvelle liste, nous trouvons 3 listes géographiquement les plus proches qui ont des incorporations, et sont du même type de liste et de la même gamme de prix que la nouvelle liste, et calculons leur vecteur moyen." (Listing Embeddings in Search Ranking, Airbnb mars 2018)

## 3.2 Traitement des données

Pour qu'un algorithme puisse utiliser des données, celles-ci doivent subir quelques transformations préalables. La liste de ces transformations possibles est large mais nous retrouvons en général le retrait des mots vides de sens suivi d'une transformation en valeurs numériques selon la technique choisie.

**Retrait des mots vides**

Lorsque l'on traite les données, on commence souvent par retirer les mots considérés comme vide de sens ("stopwords" en anglais). Les mots vides sont tellement communs qu'il est inutile de les indexer dans le vocabulaire du corpus étant donné leur distribution statistique uniforme dans les textes de la collection. Chaque langue a sa liste de mots vides, nous retrouvons par exemple en français les mots : "le", "la", "de", "du", "ce", etc.

**Transformation des données en vecteurs**

| | La | caméra | est | très | bien | mauvaise |
|---|---|---|---|---|---|---|
| La caméra est bien | 1 | 1 | 1 | 0 | 1 | 0 |
| La mauvaise caméra est très mauvaise | 1 | 1 | 1 | 1 | 0 | 2 |
| La caméra est très bien | 1 | 1 | 1 | 1 | 1 | 0 |

Représentation de valeurs textuelles en valeurs numériques. Chaque case indique le nombre d'occurrences d'un mot dans une phrase. Chaque ligne représente une phrase et chaque colonne représente un mot du vocabulaire du corpus.

Afin de pouvoir être comprises par les algorithmes, les descriptions textuelles des items doivent être transformées en valeurs numériques. La méthode la plus simple consiste à reprendre la fréquence brute des termes et à les placer dans une matrice où chaque ligne représente un item et chaque colonne un mot du vocabulaire du corpus. Cela signifie que si le vocabulaire fait 10.000 mots, il faudra autant de colonnes pour chaque item. Chaque case de cette matrice indique le nombre d'occurrences d'un mot pour un item. Cette simple représentation est suffisante pour que les algorithmes puissent manipuler ces données et effectuer des calculs de similarités, voir d'entraîner des algorithmes d'apprentissage automatique.

**Pondération des mots via la méthode TF-IDF**

Pour le moment, nous comptons les mots avec la même importance, or un mot n'a pas nécessairement la même pertinence face aux autres. Afin d'ajouter cette nuance, nous pouvons pondérer les mots via la mesure TF-IDF. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document relativement à une collection ou un corpus. Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document et varie également en fonction de la fréquence du mot dans le corpus. Cette méthode vise à donner un poids plus important aux termes les moins fréquents, considérés comme plus discriminants.

$$w_{ij} = tf_{i,j} * log(\frac{N}{df_i})$$

Formule TF-IDF où i représente un item, j un mot et N le nombre d'items

➢ Le premier terme de la formule représente la fréquence d'apparition d'un terme dans un item. Ceci se calcule par la division du nombre d'apparitions du terme dans l'item par le nombre total des termes dans l'item.

➢ Le deuxième terme de la formule représente la fréquence inverse d'un terme calculé par le logarithme de l'inverse de la proportion des items du corpus qui contiennent le terme.

**Calcul de similarité**

Avant de pouvoir étudier les algorithmes de recommandations, nous devons savoir comment nous allons calculer la similarité entre les utilisateurs et les items. Plusieurs mesures de similarité ont été proposées dans la littérature. Parmi celles-ci, nous retiendrons deux des plus utilisées, le coefficient de corrélation de Pearson et la similarité cosinus. Il existe d'autres mesures de similarité qui n'ont pas connu d'adoption significative et que par conséquent je n'aborderai pas.

**Coefficient de corrélation de Pearson :**

Le coefficient de Pearson est un indice situé entre -1 et 1, reflétant une relation linéaire entre deux variables continues. S'il s'agit de calculer la similarité entre deux utilisateurs, leur corrélation est mesurée à l'aide de deux vecteurs représentants les interactions antérieures. Dans le cas de données explicites, seuls les items co-évalués sont incorporés dans les deux vecteurs. Un coefficient proche de -1 signifie que les utilisateurs a une similarité inverse et inversement, un coefficient proche de 1 signifie une similarité entière entre les deux utilisateurs. Un coefficient proche de 0 signifie que les utilisateurs partagent une similarité moyenne.

La similarité sim(u, v) entre les utilisateurs u et v est donnée par l'équation suivante:

$$sim(u,v) = \frac{\sum_{i \in I} (r_{ui} - \bar{r}_u) * (r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{ui} - \bar{r}_u)^2 * \sum_{i \in I} (r_{vi} * \bar{r}_v)^2}}$$

Similarité par la corrélation Pearson entre deux utilisateurs u et v. I est l'ensemble des items qui ont été co-évalués par les utilisateurs u et v.

La similarité sim(i, j) entre les items i et j est donnée par l'équation suivante:

$$sim(i,j) = \frac{\sum_{u \in U} (r_{ui} - \bar{r}_i) * (r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{ui} - \bar{r}_i)^2 * \sum_{u \in U} (r_{uj} * \bar{r}_j)^2}}$$

Similarité par la corrélation Pearson entre deux items i et j. U est l'ensemble des utilisateurs qui ont co-évalué les items i et j.

**Similarité basée sur le cosinus :**

La similarité cosinus permet de calculer la similarité en déterminant le cosinus de l'angle entre deux vecteurs de même dimension. Cette similarité se situe entre 0 et 1 où le 0 signifie aucune similarité et 1 une similarité parfaite.

La similarité sim(u, v) entre les utilisateurs u et v est donnée par l'équation suivante:

$$sim(u,v) = cos(r_u, r_v) = \frac{\sum_{i \in I} r_{ui} * r_{vi}}{\sqrt{\sum_{i \in I} r_{ui}^2 * \sum_{i \in I} r_{vi}^2}}$$

Similarité cosinus entre deux utilisateurs u et v. I est l'ensemble des items qui ont été co-évalués par les utilisateurs u et v.

La similarité sim(i, j) entre les items i et j est donnée par l'équation suivante:

$$sim(i,j) = cos(r_i, r_j) = \frac{\sum_{u \in U} r_{ui} * r_{uj}}{\sqrt{\sum_{u \in U} r_{ui}^2 * \sum_{u \in U} r_{uj}^2}}$$

Similarité cosinus entre deux items i et j. U est l'ensemble des utilisateurs qui ont co-évalué les items i et j.

## 3.3 Approche basée sur le contenu

Les algorithmes de recommandation se divisent en deux grandes familles : les algorithmes de filtrage collaboratif, basés sur la recommandation de produits appréciés par les personnes qui aiment les mêmes choses que la personne cible et les algorithmes basés sur le contenu, qui recommandent des choses proches des items déjà visités.

Contrairement aux méthodes collaboratives qui ne s'appuient que sur les interactions utilisateur-article, les approches basées sur le contenu utilisent des informations supplémentaires sur les utilisateurs et/ou les articles. Si nous prenons l'exemple d'un système de recommandation de films, ces informations supplémentaires peuvent être, par exemple, l'âge, le sexe, l'emploi ou toute autre information personnelle des utilisateurs, ainsi que la catégorie, les acteurs principaux, la durée ou d'autres caractéristiques des films.

Ensuite, l'idée des méthodes basées sur le contenu est d'essayer de construire un modèle, basé sur les "caractéristiques" disponibles, qui explique les interactions observées entre l'utilisateur et l'objet. Toujours en considérant les utilisateurs et les films, nous essaierons, par exemple, de modéliser le fait que les jeunes femmes ont tendance à mieux évaluer certains films, que les jeunes hommes ont tendance à mieux évaluer d'autres films et ainsi de suite. Si nous parvenons à obtenir un tel modèle, il est alors assez facile de faire de nouvelles prédictions pour un utilisateur : il suffit de regarder le profil (âge, sexe, ...) de cet utilisateur et, sur la base de ces informations, de déterminer les films pertinents à suggérer.

Les méthodes basées sur le contenu souffrent beaucoup moins du problème du "démarrage à froid" que les approches collaboratives : les nouveaux utilisateurs ou éléments peuvent être décrits par leurs caractéristiques (contenu) et des suggestions pertinentes peuvent donc être faites pour ces nouvelles entités. Seuls les nouveaux utilisateurs ou éléments présentant des caractéristiques inédites souffriront logiquement de cet inconvénient, mais une fois que le système est suffisamment ancien, cela n'a que peu ou pas de chances de se produire.

## 3.4 Approche basée sur le filtrage collaboratif

Les méthodes collaboratives pour les systèmes de recommandation sont des méthodes qui se basent uniquement sur les interactions passées enregistrées entre les utilisateurs et les articles afin de produire de nouvelles recommandations. Ces interactions sont stockées dans ce que l'on appelle la "matrice des interactions utilisateur-article".

| | Item 1 | Item 2 | Item 3 | Item 4 | Item 5 | Item 6 |
|---|---|---|---|---|---|---|
| User 1 | X | | X | | X | |
| User 2 | | X | X | | | |
| User 3 | | | | X | | X |
| User 4 | | | | | X | |
| User 5 | X | X | | X | | X |
| User 6 | | | X | X | | |
| User 7 | X | X | X | | X | X |
| User 8 | | X | | X | | |
| User 9 | | | X | | | |

$$R$$

Ensuite, l'idée principale qui régit les méthodes collaboratives est que ces interactions passées entre utilisateurs et articles sont suffisantes pour détecter les utilisateurs similaires et/ou les articles similaires et faire des prédictions basées sur ces proximités estimées.

La classe des algorithmes de filtrage collaboratif est divisée en deux sous-catégories, généralement appelées approches basées sur la mémoire et approches basées sur un modèle.

- Les approches basées sur la mémoire travaillent directement avec les valeurs des interactions enregistrées, sans supposer de modèle, et sont essentiellement basées sur la recherche des plus proches voisins (par exemple, trouver les utilisateurs les plus proches d'un utilisateur d'intérêt et suggérer les articles les plus populaires parmi ces voisins).
- Les approches basées sur un modèle supposent un modèle "génératif" sous-jacent qui explique les interactions utilisateur-article et tentent de le découvrir afin de faire de nouvelles prédictions.

Le principal avantage des approches collaboratives est qu'elles ne nécessitent aucune information sur les utilisateurs ou les articles et qu'elles peuvent donc être utilisées dans de nombreuses situations. Par ailleurs, plus les utilisateurs interagissent avec les articles, plus les nouvelles recommandations deviennent précises : pour un ensemble fixe d'utilisateurs et d'articles, les nouvelles interactions enregistrées au fil du temps apportent de nouvelles informations et rendent le système de plus en plus efficace.

Cependant, comme il ne prend en compte que les interactions passées pour formuler des recommandations, le filtrage collaboratif souffre du "problème du démarrage à froid" : il est impossible de recommander quoi que ce soit aux nouveaux utilisateurs ou de recommander un nouvel élément à n'importe quel utilisateur, et de nombreux utilisateurs ou éléments ont trop peu d'interactions pour être traités efficacement. Cet inconvénient peut être résolu de différentes manières :

- Recommander des éléments aléatoires aux nouveaux utilisateurs ou de nouveaux éléments à des utilisateurs aléatoires (stratégie aléatoire).
- Recommander des éléments populaires aux nouveaux utilisateurs ou de nouveaux éléments aux utilisateurs les plus actifs (stratégie d'espérance maximale).
- Recommander un ensemble de divers éléments aux nouveaux utilisateurs ou un nouvel élément à un ensemble de divers utilisateurs (stratégie exploratoire).
- Utiliser une méthode non collaborative au début.

## 3.5 Approche hybride

La plupart des systèmes de recommandations utilise une approche hybride combinant le filtrage collaboratif et le filtrage basé sur le contenu. Il n'y a aucune raison pour que plusieurs techniques différentes du même type ne puissent pas être hybridées. Plusieurs études comparant empiriquement les performances de l'hybride aux méthodes purement collaboratives et basées sur le contenu ont démontré que les méthodes hybrides peuvent fournir des recommandations plus précises que les approches pures. Ces méthodes peuvent également être utilisées pour surmonter certains des problèmes tels que le démarrage à froid. Ces techniques d'hybridation comprennent les techniques :

- Pondéré : combinaison numérique du score de différents éléments de recommandation.
- Commutation : choisir parmi les composants de recommandation et appliquer celui sélectionné.
- Mixte : les recommandations de différentes recommandations sont présentées ensemble pour donner la recommandation.

## 3.6 Métrique d'évaluation

Les systèmes de recommandations basés sur le filtrage collaboratif font des prédictions sur la matrice des interactions des utilisateurs. Pour mesurer la précision de celles-ci, les interactions estimées sont comparées avec les interactions réelles, c'est-à-dire celles qui ont été créées par l'utilisateur. La précision d'un système de recommandations est généralement évaluée par deux mesures principales : l'erreur quadratique moyenne (RMSE) et l'erreur absolue moyenne (MAE). Ces deux métriques permettent une interprétation facile de par leur même échelle que les notes d'origine. Cependant, il peut être préférable d'en utiliser une ou l'autre selon le contexte.

**MAE**

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Erreur moyenne absolue - Mean absolute error

L'erreur moyenne absolue a pour caractéristique qu'elle ne donne aucun biais aux erreurs extrêmes. S'il y a des valeurs aberrantes ou des termes d'erreur importants, leur erreur pèsera de la même manière que les autres erreurs de prédictions. Par conséquent, la métrique MAE doit être privilégiée lorsque l'on recherche davantage l'exactitude des notes plutôt que de donner de l'importance aux valeurs aberrantes. Pour obtenir une vue ou une représentation holistique du système de recommandation, on utilisera donc la métrique MAE.

**RMSE**

$$RMSE = \sqrt{\sum_{i=1}^{n}\frac{(\hat{y}_i - y_i)^2}{n}}$$

Erreur quadratique moyenne - Root mean squared error

L'erreur quadratique moyenne a pour caractéristique principale qu'elle a tendance à pénaliser davantage les erreurs importantes puisqu'elles sont mises au carré. Cela signifie que la métrique RMSE est plus susceptible d'être affecté par des valeurs aberrantes ou de mauvaises prédictions. Par définition, la métrique RMSE ne sera jamais aussi petit que la métrique MAE. De plus, la métrique RSME n'utilise pas de valeurs absolues, ce qui est beaucoup plus pratique mathématiquement lors du calcul de la distance, du gradient ou d'autres mesures. C'est pourquoi la plupart des fonctions de coût de l'apprentissage automatique évitent d'utiliser la métrique MAE.

# 4. Expérimentation

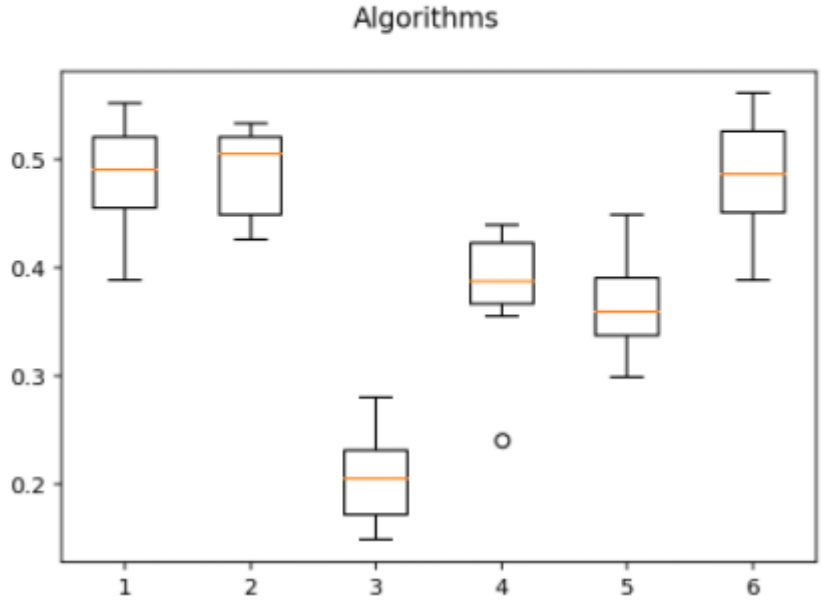
## 4.1 Expérimentation basée sur le contenu

**Prédiction des genres :**

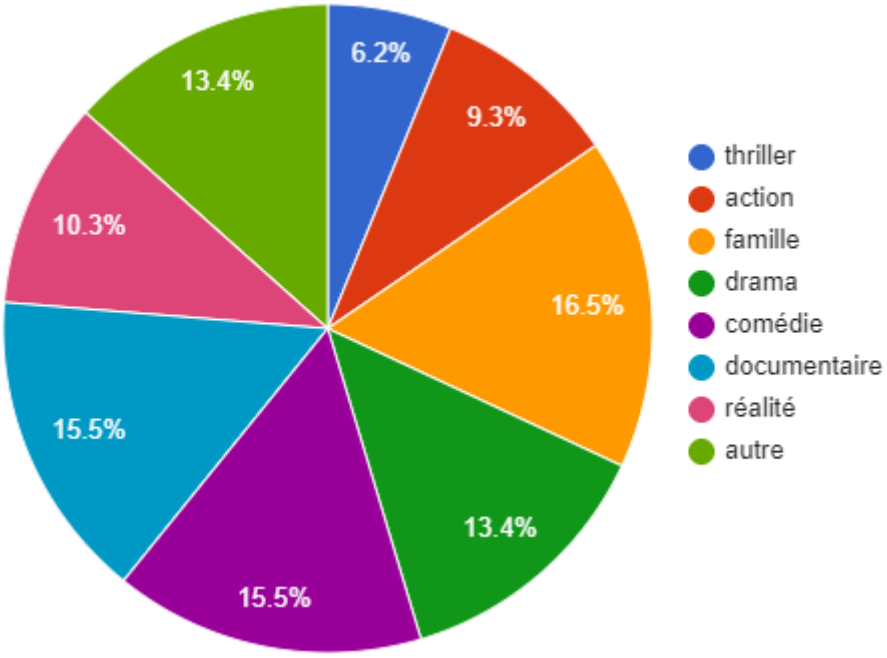
Nous avons utilisé différents classifier pour la prédiction des genres des séries à partir de leurs sous-titres afin de voir lequel nous fournira la meilleure précision. Nous pouvons remarquer que le Random Forest Classifier nous donne les meilleurs résultats. Le Random Forest Classifier est une méthode d'apprentissage d'ensemble pour la classification, la régression et d'autres tâches qui fonctionne en construisant une multitude d'arbres de décision au moment de la formation et en produisant la classe qui est le mode des classes (classification) ou la prédiction moyenne (régression) des arbres individuels. Les forêts de décision aléatoires corrigent l'habitude des arbres de décision de s'adapter de manière excessive à leur ensemble d'apprentissage. Les forêts aléatoires sont généralement plus performantes que les arbres de décision, mais leur précision est inférieure à celle des arbres boostés par le gradient.

Les précisions des différents classifier pour la prédiction des genres des séries à partir des sous-titres

```
Logistic Regression: 0.483749 (0.046856)
Random Forest Classifier: 0.490256 (0.041607)
K Neighbors Classifier: 0.208238 (0.040413)
Decision Tree Classifier(): 0.382018 (0.055142)
Ada Boost Classifier: 0.366970 (0.040928)
SVM: 0.482814 (0.050136)
```



Cependant, les caractéristiques des données affectent leurs performances. Dans notre cas nous manquons de données ce qui explique que nos résultats ne sont pas très représentatifs. En effet on peut observer une mauvaise distribution des genres ce qui influe sur les accuracy.



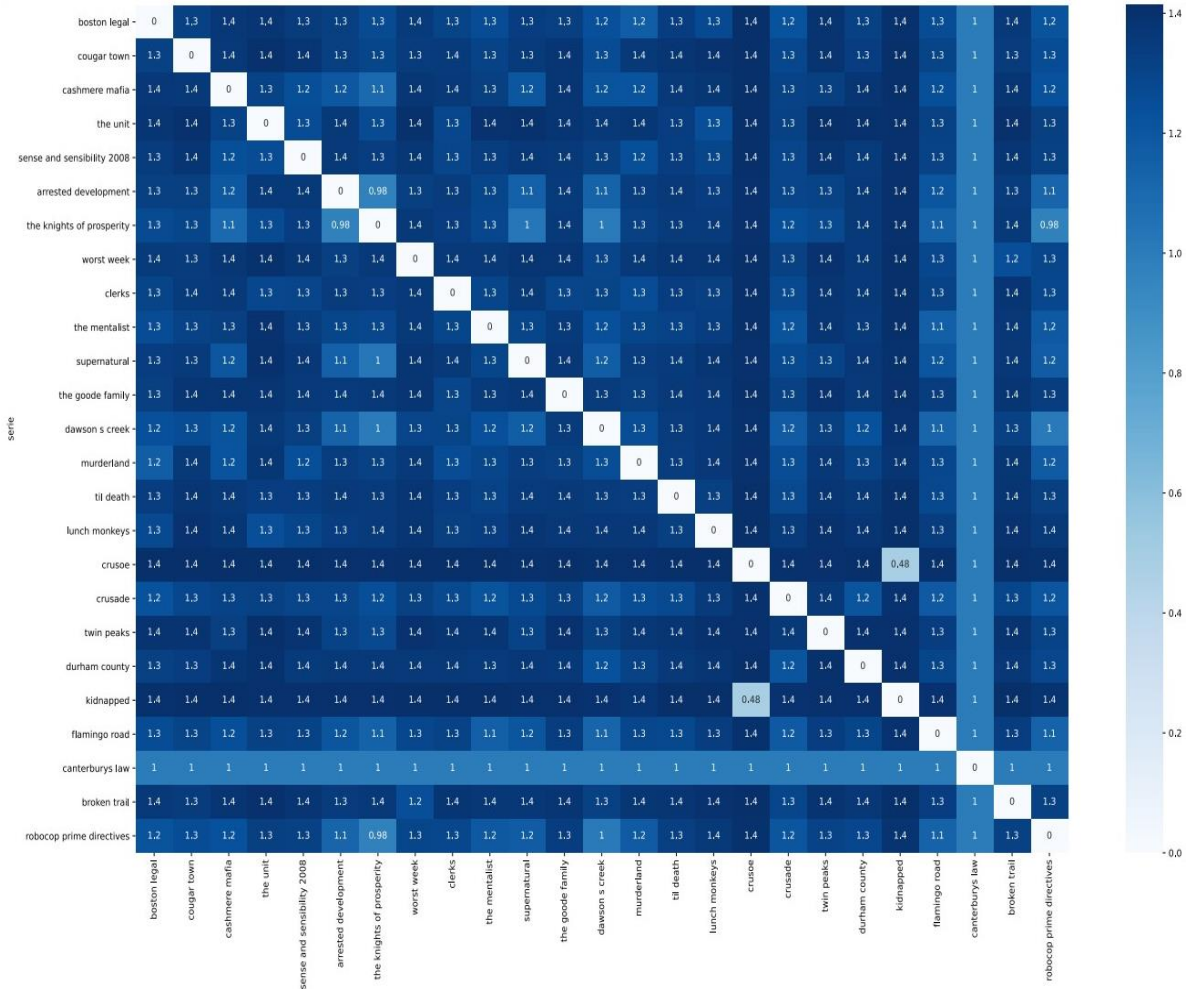

La distribution des genres de la base de données de séries

**Mesure des distances :**

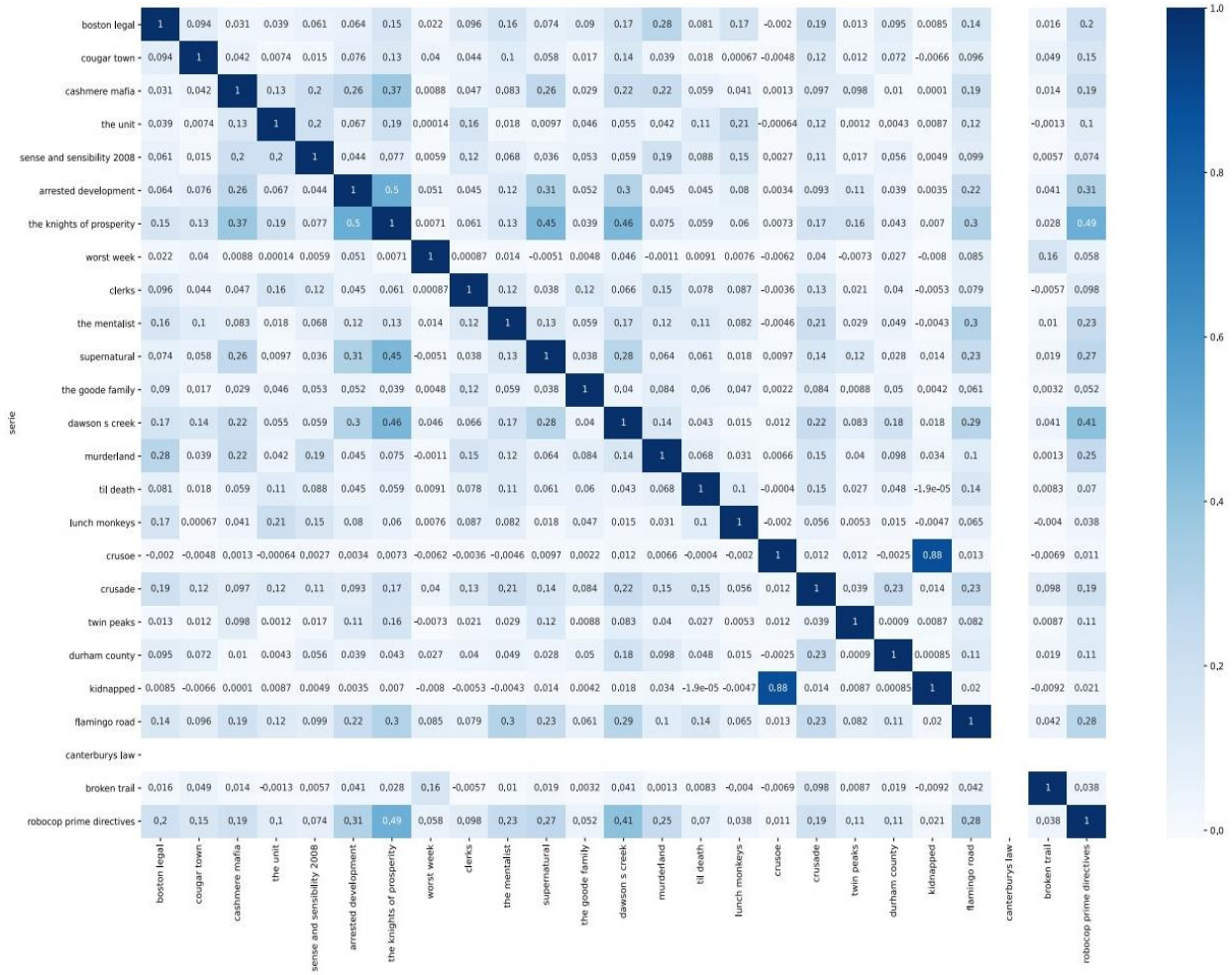
Nous avons comparé plusieurs types de mesures d'évaluation pour voir laquelle sera la plus intéressante à utiliser. Nous pouvons remarquer que la heatmap des matrices proposées par la distance cosine et pearson sont très intéressantes car elles nous permettent de mettre plus clairement en évidence les séries similaires par leurs sous titres. Nous avons décidé d'utiliser dans notre cas la distance cosine qui est légère mieux sur un ensemble de donnés plus conséquent.

Ces matrices permettent aussi de mettre en évidence certaines anomalies qui peuvent être expliquer par le fait que certaines séries sont dans des langues différentes. Par exemple on remarque que la série canterburys law possède la même valeur avec l'ensemble des différentes séries. On explique cette anomalie du fait que l'ensemble des sous titres des différentes séries sont en anglais tandis que celle-ci est en espagnol.

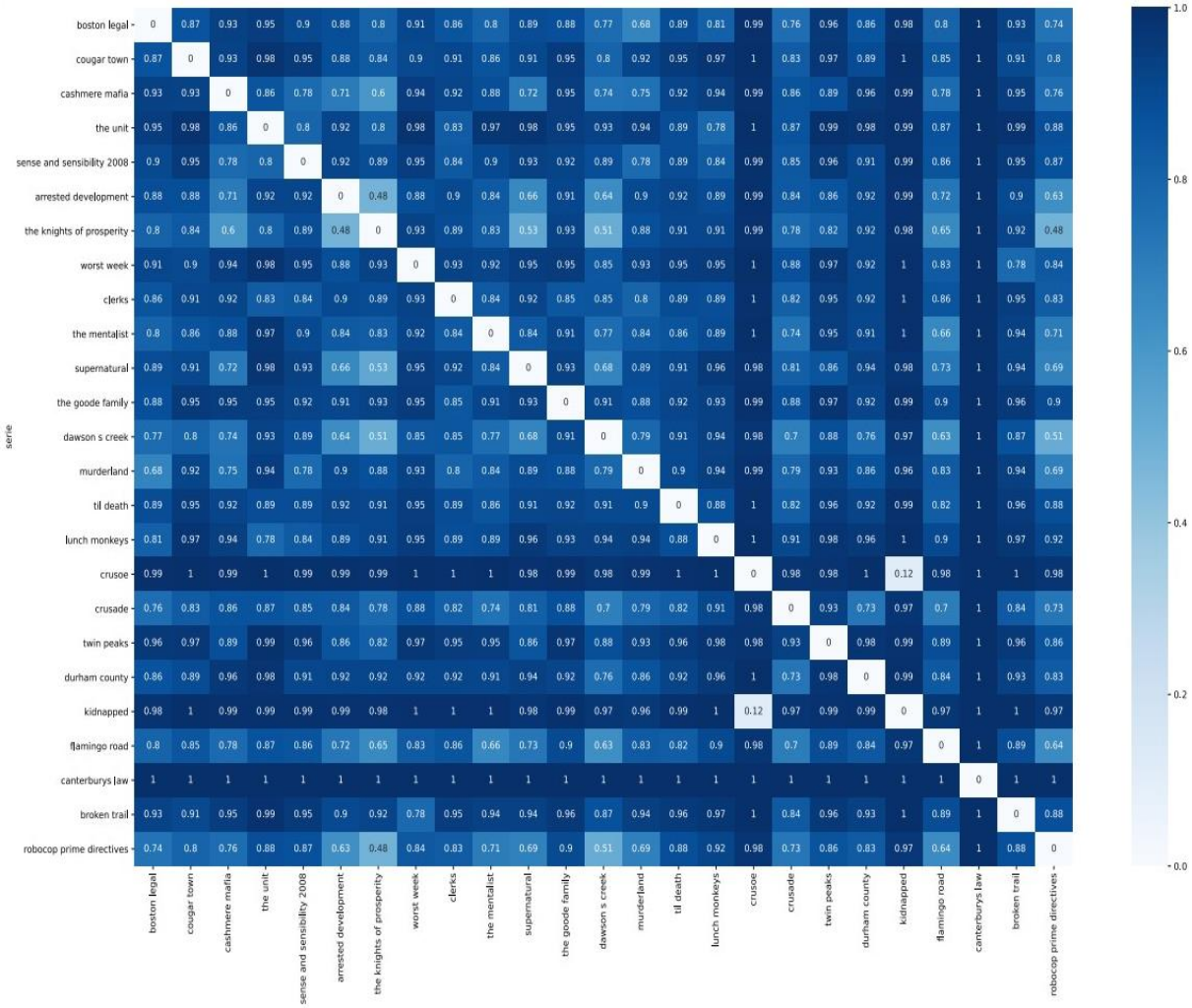
Distance euclidienne



# Corrélation de Pearson



Distance cosine



## 4.2 Expérimentation basée sur le filtrage collaboratif

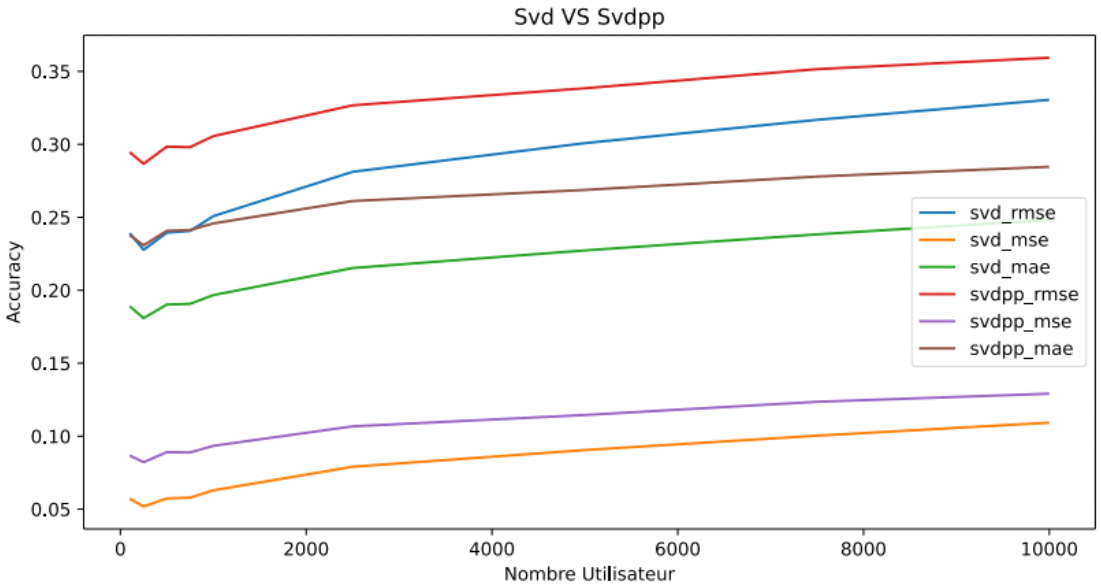
Les données qu'on utilise dans cette expérience sont les données de filtrage collaboratif c'est à dire une base de données comportant une colonne d'utilisateur à laquelle on associe le film qu'il a regardé ainsi que la note qu'il lui a été attribué par cet utilisateur. Notons que nous avons réalisé une transformation au niveau de la note pour de meilleurs résultats avec nos algorithmes. Initialement les notes étaient notées entre 0 et 10. Nous avons ramené ces notes sur 5.

A travers cette expérience nous avons confronté 2 variantes du SVD. Le SVD et le SVD++ (ou SVDpp). La décomposition en valeurs singulières, ou SVD en abrégé, est une méthode de décomposition matricielle permettant de réduire une matrice à ses éléments constitutifs afin de simplifier certains calculs matriciels ultérieurs. L'algorithme SVD++ est une variante de SVD prenant en compte les notations implicites.

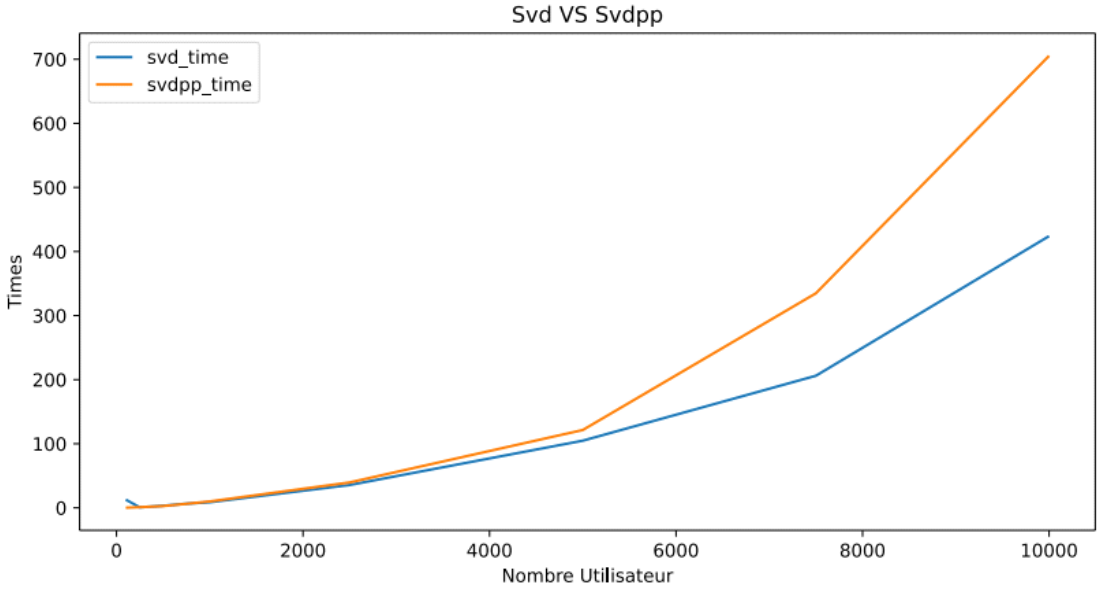
Nous avons réalisé différents apprentissages avec un nombre d'utilisateurs qui varie de 100 à 100000. Afin de mesurer le taux d'erreur, nous avons évalué à travers différentes métriques RMSE, MSE et le MAE. Nous remarquons que nous obtenons de meilleur accuracy et une exécution plus rapide avec le SVD et non la variante SVD++. C'est pourquoi nous avons retenues pour nos recommandations par filtrage collaboratif le SVD.

Nous avons décidé de mesurer plusieurs accuracy avec différentes métriques pour pouvoir relever différents types d'erreurs. Avec la MSE on accorde une grande importance aux grandes erreurs. Notamment les cas où l'utilisateur aurait voté 4 ce qui signifie que l'utilisateur a apprécié la série tandis que le système de recommandation durant sa phase d'apprentissage prédit que cet utilisateur donnera une note de 1 à ladite série ce qui signifie que la recommandation ne le suggérera pas alors que c'est une potentielle série que l'utilisateur pourrait apprécier.

Nous pouvons dire qu'avec la MSE nous mesurons la variance des résidus tandis qu'avec le RMSE mesurons l'écart type des résidus. Avec la MAE on mesure le nombre d'erreur par rapport aux valeurs aberrantes. Par exemple si un utilisateur met des notes comprises entre 0 et 1 à de nombreuses séries globalement très bien notées. On peut dire que les valeurs données par cet utilisateur sont aberrantes.




Comparaison des précisions obtenus pour les différentes metriques




Comparaison des performances obtenus par les 2 variantes du SVD

# 5. Conclusion

Nous avons conclu nos recherches sur un prototype de système de recommandation qui donne la possibilité d'ajouter et mettre à jour les notes globales des séries ou la note d'un utilisateur. Cela nous permet d'une part d'enrichir nos bases de données en ajoutant des séries dans celle-ci et d'autre part de réaliser des recommandations en fonctions des différents profils d'utilisateurs.

Nous distinguons 2 cas :

- Les utilisateurs qui ne possèdent pas de comptes

- Les utilisateurs qui possèdent un compte et ont déjà noté un ensemble de séries.

Si l'utilisateur n'a pas de compte nous lui proposons s'il le souhaite de choisir un genre de séries en particulier ou bien de recevoir une recommandation globale en fonction des tops séries les mieux notés. Cette partie du système de recommandation est essentielle car elle permet de faire face au problème du démarrage à froid vu précédemment. Nous mettons à disposition des recommandations pertinentes selon l'envie de l'utilisateur. Si celui-ci ne souhaite pas perdre de temps il aura immédiatement des recommandations globales sur les meilleures séries dans le cas contraire il aura la possibilité de sélectionner les genres qu'il préfère et ainsi avoir une recommandation plus centrée sur ceux-ci.

Si l'utilisateur possède un compte dans ce cas nous utiliserons notre système de recommandation de filtrage collaboratif qui s'appuie sur l'algorithme de SVD qui permettra de faire la meilleure suggestion en fonction des séries qu il aura déjà vu et noter. Ce système de recommandation a été hybridé avec notre système de recommandation basé sur le contenu afin de suggérer des séries pouvant surprendre l'utilisateur tout en restant pertinent. Enfin pour maximiser la pertinence de nos recommandations nous nous basons aussi sur les utilisateurs ayant des profils ressemblants c'est-à-dire des utilisateurs qui regarde globalement des séries du mêmes genres.

# 6. Bibliographie


– Xiwang Yang, Yang Guo, Yong Liu, Harald Steck, A survey of collaborative filtering based social recommender systems (2013)

– J. Bobadilla, F. Ortega, A. Hernando, A. Gutiérrez, Recommender systems survey (2013)

– Xiaoyuan Su and Taghi M. Khoshgoftaar, A Survey of Collaborative Filtering Techniques (2009)

– Mohd Abdul Hameed, Omar Al Jadaan, S. Ramachandram, Collaborative Filtering Based Recommendation System : A survey (2012)