Sciences des données - TP3

Alexandre Theisse Louis-Vincent Capelli Tom Sartori

December 13, 2023

Contents

Le code source de notre projet est organisé de la manière suivante :

- `src/data.py` : module de gestion des données
- `src/preprocessing.py` : module de prétraitement des données
- `main.ipynb` : notebook contenant les réponses aux questions
- `hp_search.ipynb` : notebook contenant la recherche des hyperparamètres optimaux pour l'arbre de décision

1 Question 1

En utilisant la fonction `load_movies_with_genre` de notre module de gestion des données, nous avons chargé uniquement les films ayant un genre défini.

Voici la répartition des genres dans la base de données :

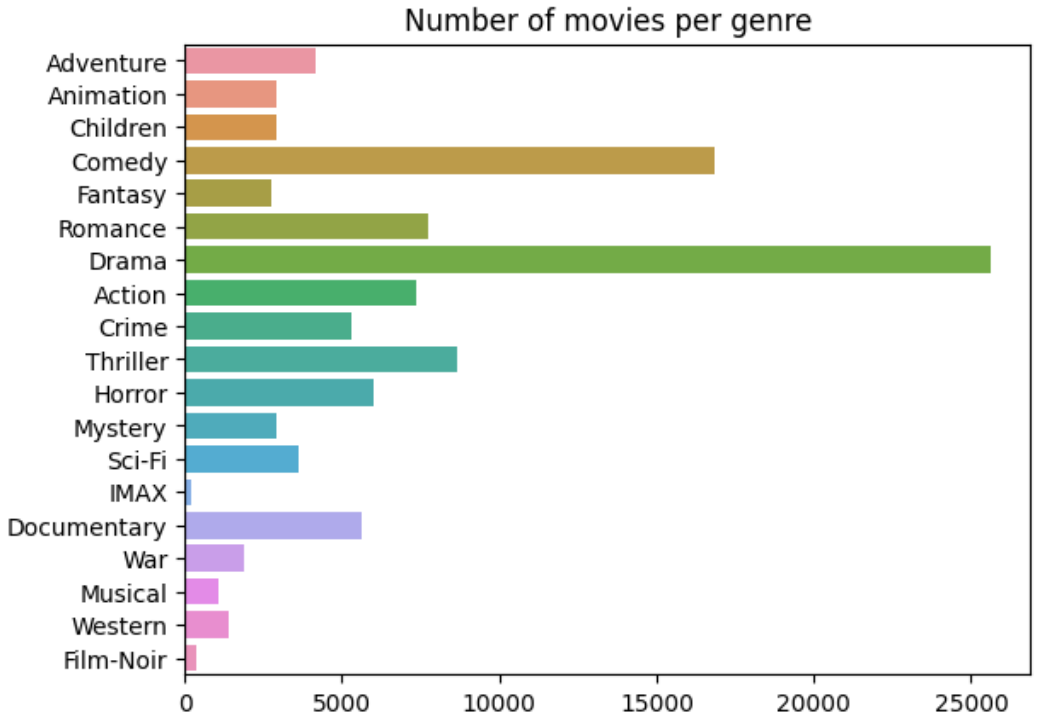

Figure 1: Répartition des genres dans la base de données

2 Question 2

Le fichier `movies1.csv` a été créé en itérant sur les lignes du fichier `movies.csv` et en ne gardant que les lignes dont le genre est défini. Pendant cette itération, nous avons également stocké les moviesId des films avec un genre.

Le fichier `ratings1.csv` a été créé en itérant sur les lignes du fichier `ratings.csv` et en ne gardant que les lignes dont le movieId est présent dans le fichier `movies1.csv`. Nous en avons profité pour arrondir les notes de la manière préconisée par l'énoncé.

Le fichier `movies1.csv` contient ainsi 57361 films sur les 62423 du fichier `movies.csv`, et le fichier `ratings1.csv` contient 1569746 notes sur les 1571101 du fichier `ratings.csv`.

3 Question 3

La construction de la matrice binaire des genres des films est effectuée par la fonction `movies_stats` de notre module de gestion des données.

Une matrice de taille $nb\_films \times nb\_genres$ remplie de 0 est créée grâce à la fonction `zeros` de numpy. Ensuite, pour chaque film, on récupère ses genres en itérant sur les lignes du fichier `movies1.csv` et on remplit la matrice avec des 1 aux positions correspondantes aux genres du film.

4 Question 4

La construction de la matrice des profils des utilisateurs est effectuée par la fonction `users_stats` de notre module de gestion des données.

Une matrice de taille $nb\_users \times nb\_genres$ remplie de 0 est créée puis on itère sur les lignes du fichier `ratings1.csv` pour remplir la matrice. Pour chaque ligne, on récupère la ligne correspondant au film dans la matrice créée à la question précédente, on multiplie cette ligne par la note du film et on ajoute le résultat à la ligne correspondant à l'utilisateur dans la matrice des profils des utilisateurs.

5 Question 5

5.1 Normalisation des profils des utilisateurs

Afin de rendre le clustering des centaines de fois plus rapide et d'éviter des erreurs d'overflow notamment lors de produits de matrices, nous avons décidé de normaliser les profils des utilisateurs. Pour cela, nous avons simplement divisé tous les coefficients de la matrice des profils des utilisateurs par le maximum de la matrice afin que tous les coefficients soient compris entre 0 et 1.

5.2 Regroupement spectral des utilisateurs

Le regroupement spectral des utilisateurs est effectué par la classe `SpectralClustering` de `scikit-learn`.

5.2.1 Choix du solveur et de la stratégie

Nous avons utilisé le solver `arpack` et la stratégie `kmeans` qui donnaient les meilleurs résultats.

5.2.2 Choix du nombre de clusters

En faisant varier nombre de clusters `k` de 2 à 5, nous avons pu determiner que le nombre de clusters optimal était 2. Voici les résultats obtenus pour les différentes valeurs de `k` en utilisant la métrique `silhouette_score` de `scikit-learn` :

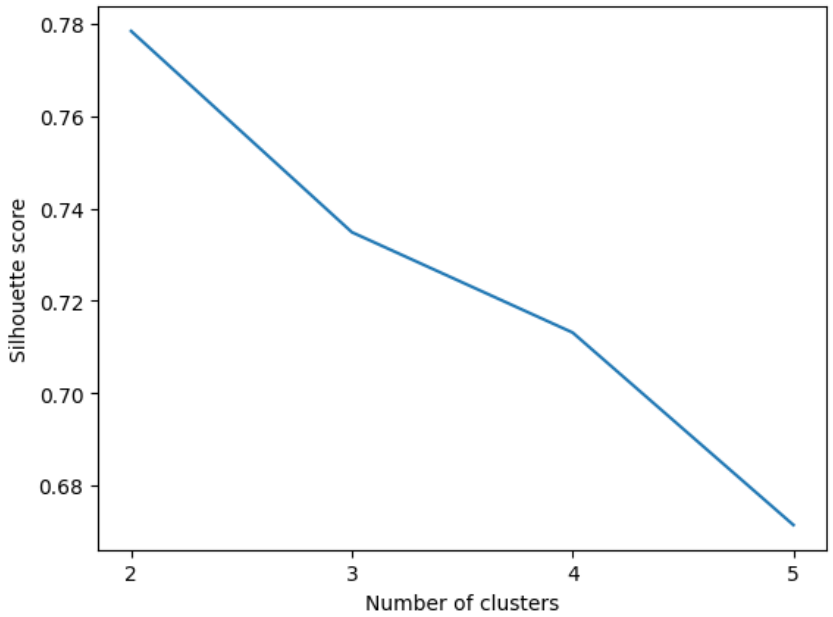

Figure 2: `silhouette_score` pour différentes valeurs de `k`

5.2.3 Choix du kernel

Nous avons essayé différents kernels avec différents paramètres pour le regroupement spectral.

Voici les résultats obtenus pour les différentes valeurs de `gamma` du kernel `rbf` avec `k = 2` :

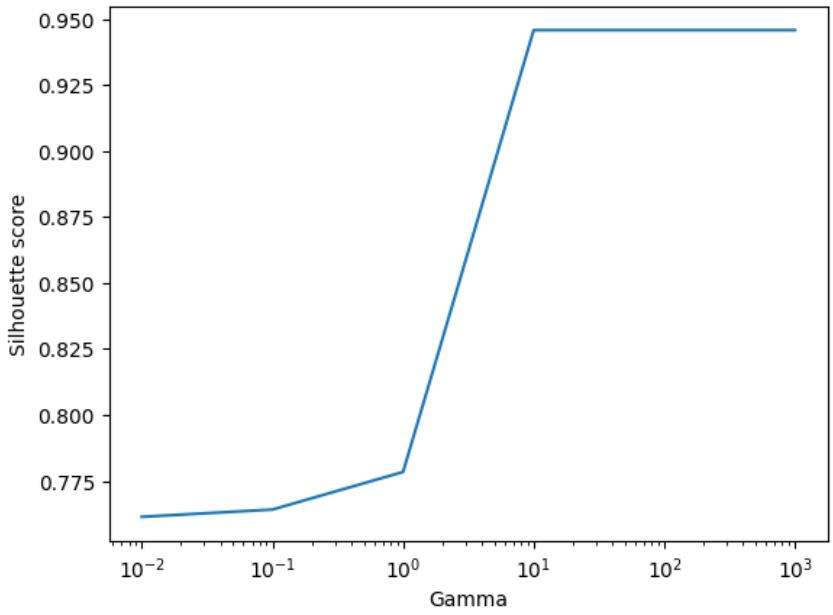

Figure 3: `silhouette_score` pour différentes valeurs de `gamma` du kernel `rbf`

On constate qu'à partir de 10, le `silhouette_score` est très élevé. Cependant, dans ce cas le silhouette score n'est pas un bon indicateur car il ne prend pas en compte le fait que les clusters obtenus sont très déséquilibrés. En effet, en utilisant un `gamma` de 10, on obtient un cluster de 10512 utilisateurs et un cluster de 2 utilisateurs.

Nous avons également essayé le kernel `nearest_neighbors` avec différentes valeurs de `n_neighbors` (`k = 2`) :

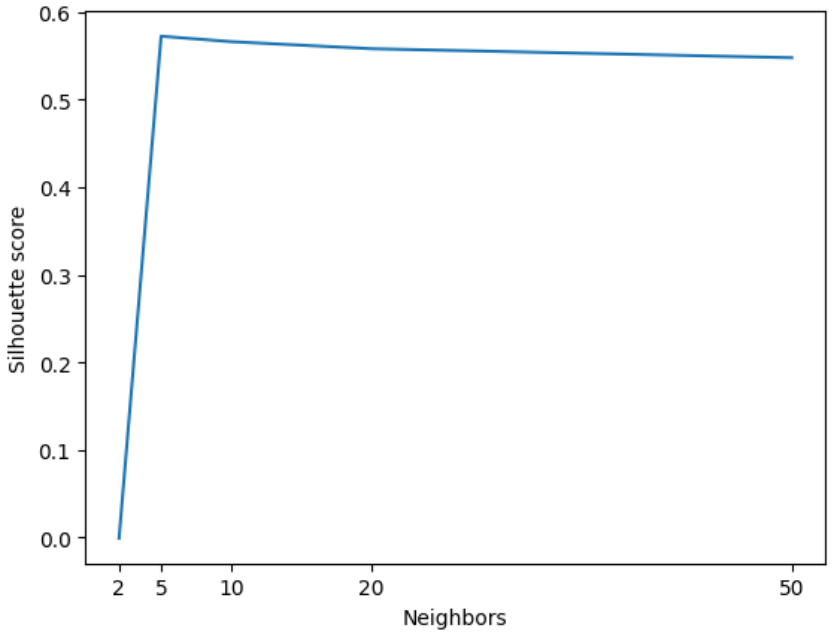

Figure 4: `silhouette_score` pour différentes valeurs de `n_neighbors`

Nous avons donc décidé de conserver le kernel `rbf` car il semble meilleur que le kernel `nearest_neighbors` sur notre jeu de données. Nous utiliserons un `gamma` de 1 car il donne des clusters un peu plus équilibrés malgré un `silhouette_score` plus faible et que les résulats finaux du classifieur sont identiques.

5.3 Visualisation des clusters

Nous avons utilisé plusieurs méthodes pour réduire la dimension des données afin de pouvoir visualiser en 2 dimensions les clusters obtenus pour $k \in \{2, 3, 4, 5\}$.

D'abord la classe `UMAP` de `umap-learn` dont voici les représentations :

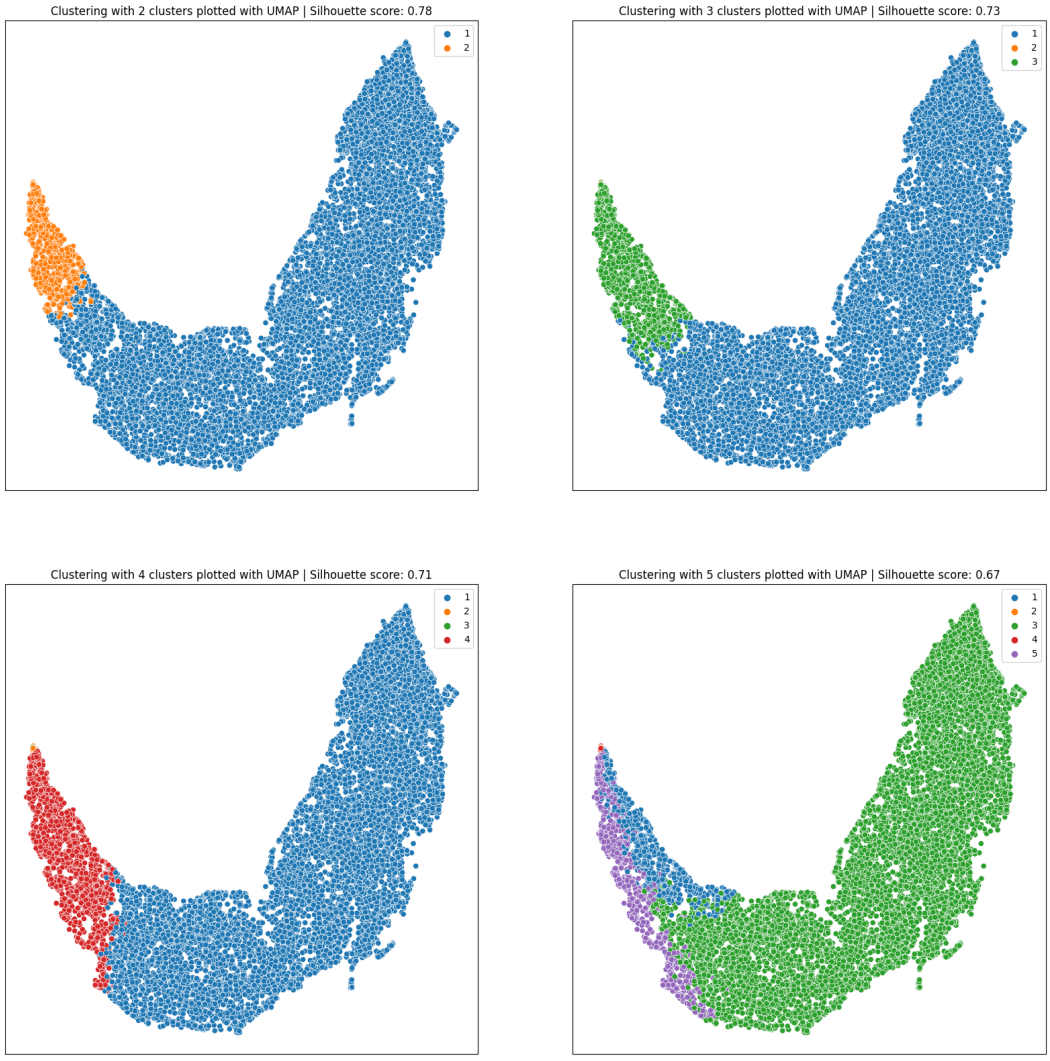

Figure 5: Visualisation des clusters pour différentes valeurs de `k` avec `UMAP`

Ensuite la classe `PCA` de `scikit-learn` dont voici les représentations :

Figure 6: Visualisation des clusters pour différentes valeurs de `k` avec `PCA`

Enfin la classe `TSNE` de `scikit-learn` dont voici les représentations :

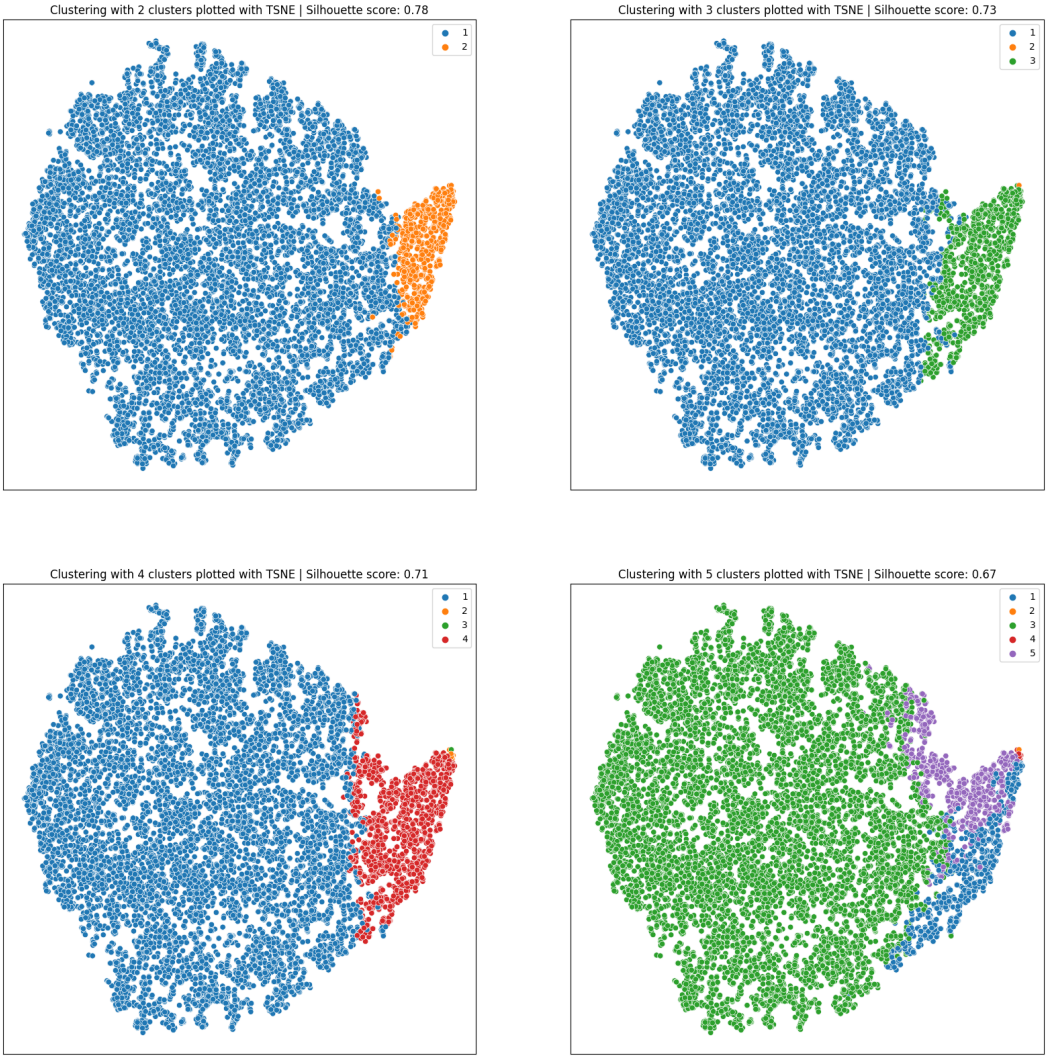

Figure 7: Visualisation des clusters pour différentes valeurs de `k` avec `TSNE`

Il semble que les clusters obtenus soient très déséquilibrés, nous avons donc affiché la répartition des utilisateurs dans les clusters afin de confirmer cette hypothèse :

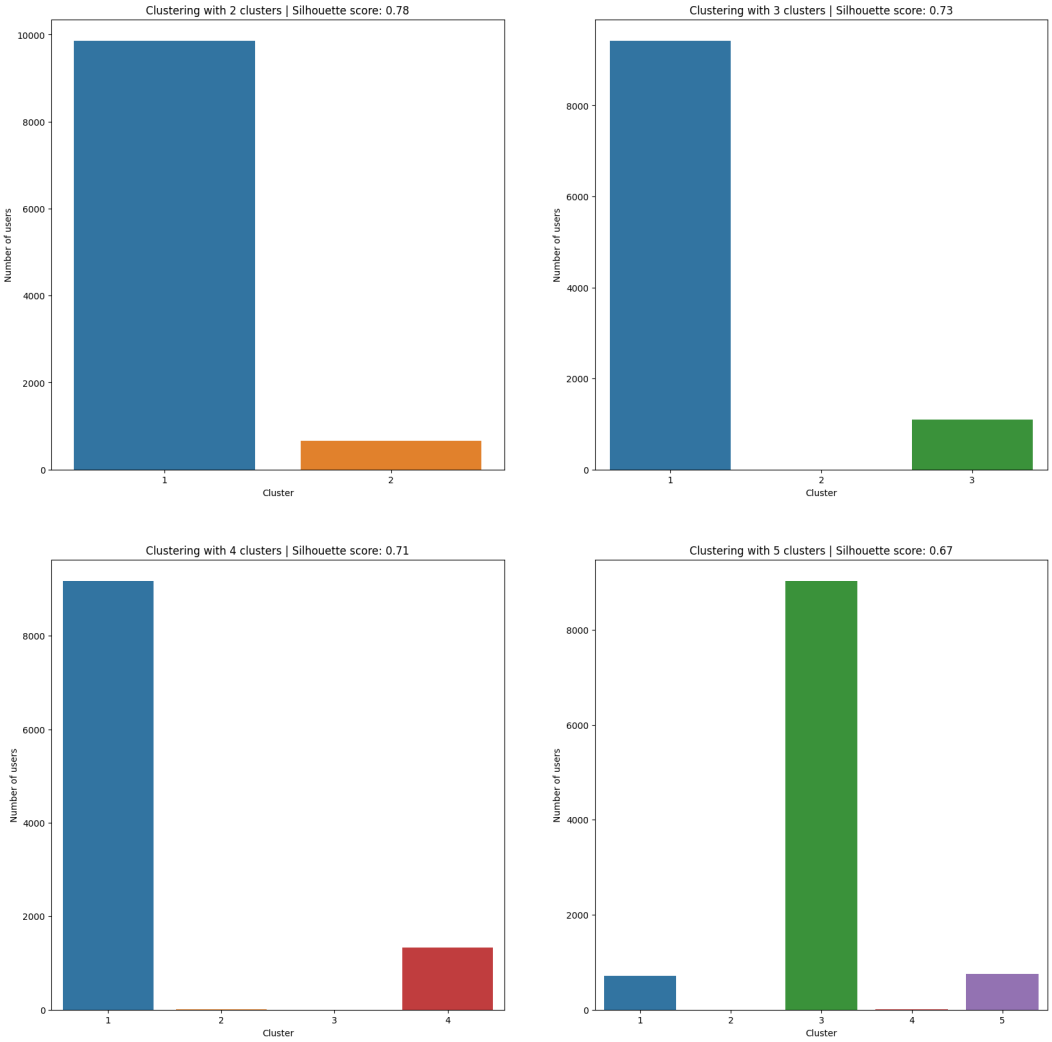

Figure 8: Répartition des utilisateurs dans les clusters pour différentes valeurs de `k`

6 Question 6

6.1 Préparation des ensembles

Nous avons utilisé la fonction `StratifiedShuffleSplit` de `scikit-learn` pour séparer les données en 3 ensembles : un ensemble d'entraînement, un ensemble de validation et un ensemble de test avec des proportions respectives de 60%, 20% et 20%.

Cette fonction nous permet de conserver la répartition des notes dans les différents ensembles.

On obtient ainsi 3 fichiers `ratings_train.csv`, `ratings_evaluation.csv` et `ratings_test.csv` contenant respectivement 941847, 313950 et 313950 notes.

6.2 Prétraitement des données

Afin de pouvoir entraîner l'arbre de décision, il faut d'abord trouver pour chacune des notes laissées par les utilisateurs, les notes des 5 utilisateurs les plus proches ayant noté le même film.

Nous avons utilisé l'attribut `affinity_matrix_` de la classe `SpectralClustering` pour obtenir la matrice des similarités entre les utilisateurs qui découle de leur regroupement spectral. Cette matrice de taille $nb\_users \times nb\_users$ contient au coefficient (i, j) la similarité entre les utilisateurs i et j vis-à-vis de leur profil.

Le prétraitement des données est effectué par la fonction `five_closest_users` de notre module de prétraitement des données. Pour chaque note, on trie les utilisateurs selon leur similarité avec l'utilisateur qui a laissé la note et ne garde que les 5 utilisateurs les plus proches ayant noté le même film. Si moins de 5 autres utilisateurs ont noté le même film, la note est ignorée.

On obtient finalement un dictionnaire avec des entrées de la forme `(userId, movieId, rating) : [r1, r2, r3, r4, r5]` où `r1, r2, r3, r4, r5` sont les notes des 5 utilisateurs les plus proches ayant noté le même film.

6.3 Recherche des hyperparamètres optimaux

Afin de trouver les hyperparamètres optimaux pour l'arbre de décision, nous avons utilisé la classe `GridSearchCV` de `scikit-learn` qui permet de faire une recherche exhaustive à travers une grille de paramètres qui contient pour chaque hyperparamètre une liste de valeurs à tester.

Nous avons testé les valeurs suivantes des hyperparamètres suivants :

- `criterion` : `['gini', 'entropy']`
- `splitter` : `['best', 'random']`
- `max_depth` : `[None, 5, 10, 50, 100, 200, 500]`
- `min_samples_split` : `[2, 5, 10, 20, 50, 100]`
- `min_samples_leaf` : `[1, 2, 5, 10, 20, 50, 100]`

- `max_features` : `['sqrt', 'log2']`

Voici les paramètres optimaux obtenus :

- `criterion` : `'gini'`
- `splitter` : `'best'`
- `max_depth` : `50`
- `min_samples_split` : `20`
- `min_samples_leaf` : `100`
- `max_features` : `'sqrt'`

Nous avons remarqué qu'en réalité, la plupart des combinaisons d'hyperparamètres donnent des résultats similaires, et que les hyperparamètres optimaux donnent des performances à peine meilleures que les paramètres par défaut.

6.4 Entraînement et prédictions de l'arbre de décision

L'arbre de décision est entraîné par la classe `DecisionTreeClassifier` de `scikit-learn` avec les hyperparamètres optimaux trouvés précédemment sur l'ensemble d'entraînement et évalué sur l'ensemble d'évaluation.

Les entrées sont les valeurs du dictionnaire créé en 6.3 et les notes prédites sont comparées avec la note qui apparaît dans la clé du dictionnaire et qui est la note réelle laissée par l'utilisateur.

La métrique utilisée pour évaluer l'arbre de décision est le F1-score qui est la moyenne harmonique de la précision et du rappel. Nous avons donc utilisé la fonction `f1_score` de `scikit-learn` pour calculer le `f1-score` des prédictions de l'arbre de décision sur l'ensemble de test.

6.5 Résultats

Le F1-score obtenu sur l'ensemble de test pour les hyperparamètres optimaux trouvés précédemment est de 0.32.

Trouvant ce score assez faible, nous avons essayé de comprendre pourquoi l'arbre de décision ne fonctionnait pas très bien. Nous avons alors décidé de calculer le F1-score en considérant comme un Vrai Positif une prédiction de l'arbre de décision qui était à 1 point ou moins de la note réelle.

Nous avons alors obtenu un F1-score de 0.82 ce qui est beaucoup plus satisfaisant. Cela signifie que même si l'arbre de décision ne prédit pas exactement la note réelle, il prédit une note très proche de la note réelle.

7 Conclusion

Finalement, malgré un F1-score assez faible, notre système de recommandation basé sur un arbre de décision semble fonctionner assez bien. On peut en effet considérer qu'une prédiction à 1 point près est une bonne prédiction puisque le but est de recommander à l'utilisateur des films qui lui plairont et non de prédire exactement la note qu'il donnera au film.