



Projet de Recherche

Système de recommandation explicable pour les jeux de société

Ekaterina Bogush
Amelie Chu
Lyna Combo

Responsable : Nicolas Baskiotis

Mai 2025

Table des matières

1	Introduction	2
2	Traitement des données	3
2.1	Choix des données	3
2.2	Filtrage des données	4
2.3	Matrice utilisateur-jeu pour filtrage collaboratif	4
3	Estimation des notes par voisinage (k-NN)	5
3.1	Distance euclidienne	5
3.2	Distance cosinus	5
3.3	Variantes étudiées	5
3.3.1	Biais des notes des utilisateurs	5
3.3.2	Pondération par distance	5
3.4	Évaluation des différents modèles	6
3.5	Limitations de k-NN	7
4	Estimation des notes par factorisation de matrices	8
4.1	SVD	8
4.2	NNMF	8
4.2.1	Nombre de facteurs latents	9
4.2.2	K-means	9
4.2.3	Visualisation à l'aide de t-SNE	10
5	NLP : explication à l'aide des commentaires et des descriptions	11
5.1	Pré-traitement des commentaires	11
5.2	Modèles-transformeurs	11
5.2.1	LLM	11
5.2.2	Modèle-encodeur	11
5.3	Explication des clusters	11
5.4	Génération des commentaires	12
5.4.1	Limitation de LLM	12
5.4.2	Prédiction par fréquence lexicale des voisins	13
5.4.3	Prédiction par phrases des voisins	13
5.5	Évaluation quantitative : score BLEU-ROUGE	14
5.5.1	Prédiction par bigrammes et unigrammes	15
5.5.2	Prédiction par phrases	16
5.6	Évaluation qualitative : LLM	17
5.6.1	Prédiction par bigrammes, unigrammes	17
5.6.2	Prédiction par phrases	18
6	Conclusion	19
	Bibliographie	20
	Annexe	21

1 Introduction

Les plateformes proposant une variété importante de contenu et essayant de séduire un nombre important d'utilisateurs, comme les plateformes de streaming ou les réseaux sociaux, ont besoin de pouvoir accéder aux données qui leur sont les plus pertinentes. Celles-ci doivent être personnalisées et se baser sur les préférences de chaque profil afin de pouvoir proposer à chaque utilisateur les contenus qu'il est le plus susceptible d'apprécier. Il s'agit là du but des systèmes de recommandation.

Les techniques de recommandation peuvent être principalement divisées en deux catégories : le filtrage collaboratif (*collaborative filtering*) et le filtrage basé sur le contenu (*content-based filtering*).

Le **filtrage basé sur le contenu** repose sur la création d'un profil pour chaque utilisateur en fonction du contenu avec lequel il a déjà interagi sur la plateforme. Nous essayons alors de lui recommander des objets similaires à ceux qu'il a déjà appréciés.

Le **filtrage collaboratif** se base sur l'identification des utilisateurs ayant des préférences similaires à un utilisateur-cible. Nous essayons alors de lui recommander des jeux que ces utilisateurs ont aimés.

Dans le cadre de ce projet de recherche, nous allons étudier ces deux méthodes de recommandation. Notre but principal est de pouvoir utiliser les notes et commentaires déjà renseignés par un utilisateur pour expliquer les notes prédites et les recommandations faites à celui-ci selon différents algorithmes de recommandation. De manière générale, d'autres critères tels que la diversité ou la nouveauté du contenu recommandé peuvent également être pris en compte.

La base de données sur laquelle nous travaillons provient de données récupérées au début de l'année 2023 sur le site francophone de jeux de société TricTrac.

Nous commencerons dans un premier temps par analyser le **filtrage collaboratif**. Dans ce domaine, deux approches sont prédominantes : le modèle basé sur le voisinage et celui basé sur les variables latentes.

Ensuite, dans un second temps, nous passerons à l'analyse et à l'explication des recommandations faites par nos modèles, en appliquant un clustering et en étudiant les commentaires des utilisateurs sous deux angles différents : l'analyse des fréquences des mots pour déterminer ceux qui portent le plus d'information ainsi que la représentation vectorielle du texte (*embedding*) qui permettra de prendre en compte le contexte de chaque mot et de regarder la similarité entre les commentaires.

2 Traitement des données

Une fois la base de données nettoyée et pré-traitée, nous pouvons en faire une analyse statistique descriptive. Notre base de données contient au total 11 201 utilisateurs, 10 549 jeux ainsi que 142 489 avis. Elle regroupe les informations suivantes :

- **Utilisateurs** : nom d'utilisateur
- **Jeux** : nom, catégorie(s), description, durée, nombre de joueurs minimum et maximum, tranche d'âge recommandée
- **Avis** : note du jeu (*entre 0 et 10*), date, titre et contenu du commentaire

En observant le nombre d'avis par année, nous pouvons remarquer que le site **TricTrac** était plutôt populaire pendant les années 2004 - 2018. Par conséquent, les jeux plus anciens auront plus d'avis et seront donc mieux représentés par rapport aux jeux modernes.

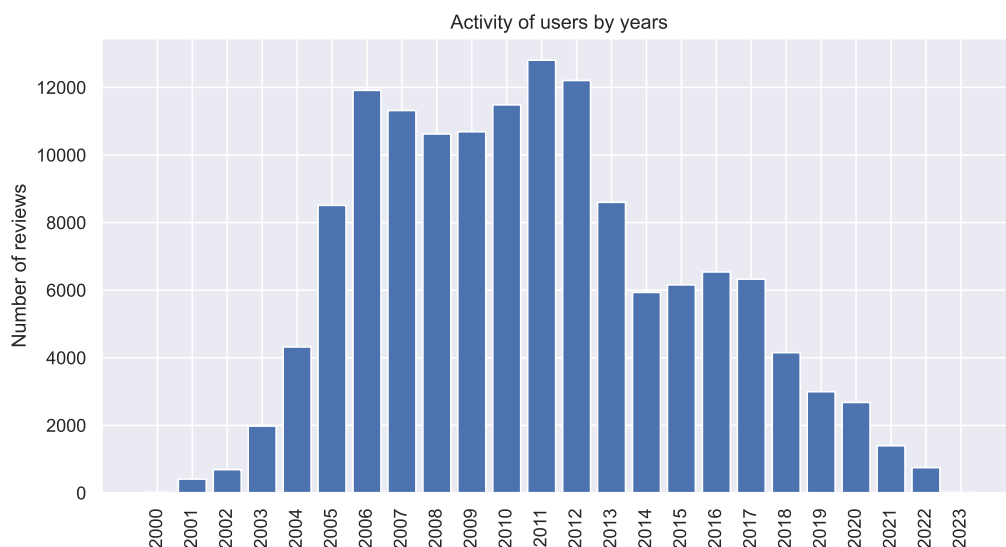


FIGURE 1 – Activité des utilisateurs par année

2.1 Choix des données

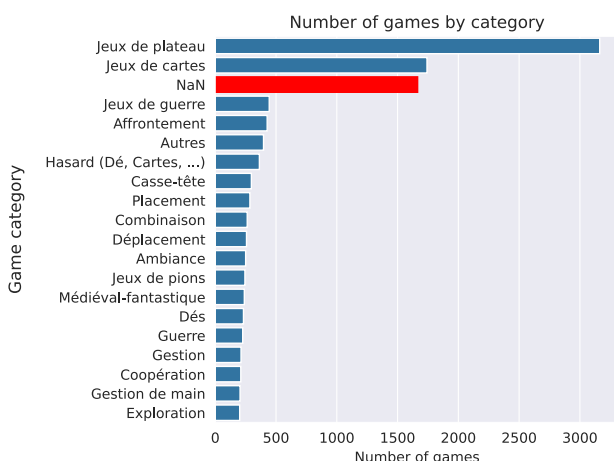


FIGURE 2 – 20 catégories les plus fréquentes

Parmi les **caractéristiques** des jeux, le nom, la durée de jeu, les tranches d'âges ainsi que le nombre de joueurs minimum et maximum ne sont pas vraiment des critères qui reflètent les préférences des utilisateurs. Elles ne sont pas non plus spécifiques à des catégories de jeux : il existe des jeux aussi courts que longs dans toutes catégories confondues. C'est pourquoi nous n'allons pas traiter ces données.

Au niveau des **catégories**, il existe 183 catégories uniques. La figure 2 illustre ceux qui apparaissent le plus souvent. La catégorie inconnue (NaN¹) est parmi 3 catégories les plus fréquentes (15% des jeux ont la catégorie inconnue). Notons qu'un jeu peut avoir plusieurs catégories, les rendant **génériques**. Si par exemple un joueur aime les jeux complexes, cela peut toucher à toutes catégories.

C'est pourquoi, nous allons nous restreindre aux données que nous jugeons pertinentes et exploitables : les notes des utilisateurs, les contenus de leurs commentaires et les descriptions des jeux. Les premières données les plus simples à exploiter sont les notes des utilisateurs qui vont nous permettre de construire des profils.

1. *Not a Number*, indique une valeur inconnue ou qui ne peut pas être représentée

2.2 Filtrage des données

Afin de pouvoir dessiner un profil pertinent pour un utilisateur (resp. un jeu), il est nécessaire de fixer une borne inférieure au nombre d'avis laissés par un utilisateur (resp. reçus par un jeu). Cela permet de limiter le problème du **cold-start** : celui-ci intervient lorsqu'un utilisateur (resp. un jeu) n'est pas associé à suffisamment d'avis pour juger de la pertinence du profil qui lui est associé.

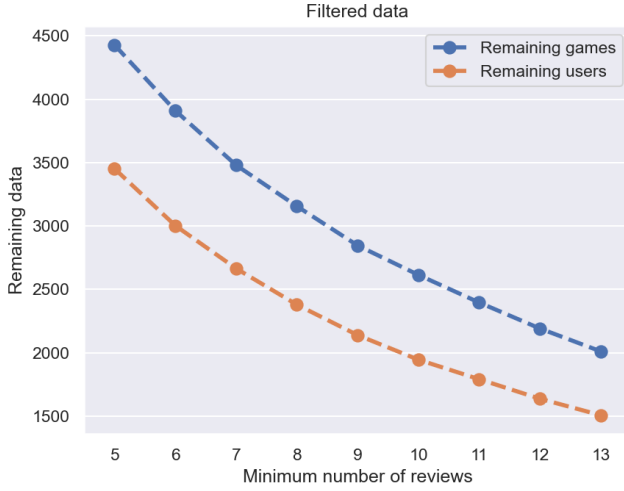


FIGURE 3 – Filtrage des données

Par la suite, nous fixons la borne inférieure à 10 avis ce qui permet de garder $N_j = 2614$ jeux, $N_u = 1945$ utilisateurs et 96 533 avis. Un tel filtrage peut être accepté à l'échelle du projet de recherche mais peut être problématique pour une application commerciale.

2.3 Matrice utilisateur-jeu pour filtrage collaboratif

Pour pouvoir traiter les notes des utilisateurs, une approche largement utilisée est de stocker les notes dans une matrice où chaque ligne représente un utilisateur et chaque colonne un jeu. Les valeurs manquantes (si un utilisateur n'a pas donné son avis sur un jeu) peuvent être représentées par des 0. Cette matrice utilisateur-jeu sera notée M_{uj} .

Étant donné que la grande majorité des utilisateurs n'a noté que quelques jeux parmi tous les jeux existants, la matrice M_{uj} est très creuse avec un pourcentage de sparsité à 98.1%. La figure 4 illustre la distribution des utilisateurs par leur activité (nombre d'avis qu'ils ont laissé). La grande majorité des utilisateurs, 74% n'ont laissé qu'un nombre limité d'avis (au plus 50 avis), ce qui explique le taux de sparsité très élevé.

De cette façon, nous pouvons représenter les utilisateurs comme des vecteurs dans un espace vectoriel de dimension N_j , où chaque axe représente un jeu. Le coefficient d'un vecteur de l'utilisateur u sur l'axe représentant le jeu j est la note m_{uj} donné par u à j . Si u n'a pas noté j , alors m_{uj} est nul.

Le filtrage collaboratif se base sur la recommandation de jeux que les utilisateurs similaires ont aimé, et que l'utilisateur est aussi susceptible d'aimer. Il nous faut pouvoir prédire la note que donnerait notre utilisateur à un jeu.

Une telle borne permet d'éliminer les jeux peu notés et les utilisateurs peu actifs. Lors du filtrage, l'élimination de certains utilisateurs réduit le nombre d'avis des jeux, et inversement (*problème du ping-pong*) : il faut donc les éliminer jusqu'à une stabilisation du nombre de jeux et d'utilisateurs.

À gauche, la figure 3 montre le nombre de jeux et d'utilisateurs restants en fonction du nombre minimum d'avis. Les données restantes décroissent très rapidement : lorsque la borne est fixée à 5, 58% des jeux et 69% des utilisateurs ont été perdus. Ces pourcentages élevés montrent que la base de données contenait un grand nombre d'utilisateurs peu actifs et des jeux très peu notés.

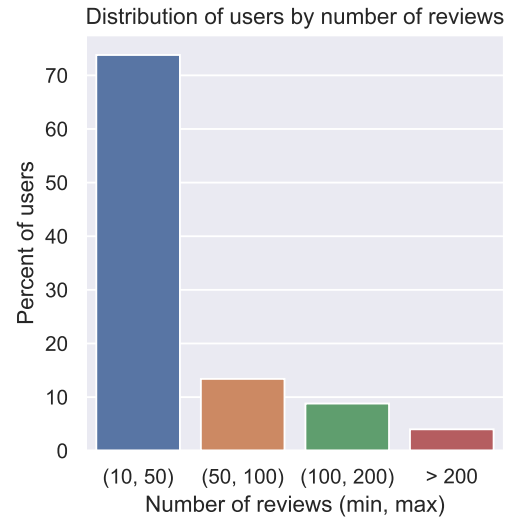


FIGURE 4 – Activité des utilisateurs

3 Estimation des notes par voisinage (k-NN)

Afin de prédire des notes pour un utilisateur-cible u_c , il faut identifier les utilisateurs ayant des goûts similaires. Comme un utilisateur est représenté par un vecteur, nous pouvons transformer ce problème en problème des k plus proches voisins (*k-nearest neighbors*) et calculer la distance entre les vecteurs.

Nous devons donc choisir un type de distance afin de construire la matrice de distance (de taille $N_u \times N_u$) entre tous les utilisateurs. Pour prédire la note $m_{u_c j}$, il suffira de choisir les k utilisateurs les plus proches à u_c et de prendre la moyenne de leurs notes pour j .

3.1 Distance euclidienne

La matrice M_{u_j} contient énormément de notes manquantes. Par conséquent, nous utiliserons la distance euclidienne pondérée qui permet de prendre en compte ces valeurs. Une telle pondération permet de prioriser les utilisateurs qui ont plus de jeux notés en commun.

Soient $u_1, u_2 \in \mathbb{R}^{N_j}$ deux vecteurs (utilisateurs) de notes

$$d_{\text{eucl}}(u_1, u_2) = \sqrt{\frac{1}{|C|} \sum_{j \in C} (u_{1j} - u_{2j})^2}$$

$$C = \{j \in \{1, \dots, N_j\} \mid u_{1j}, u_{2j} \text{ notes existantes}\}$$

Comme la matrice M_{u_j} est très creuse, la similarité entre les utilisateurs qui ont un nombre très limité de jeux notés en commun sera peu fiable. Ainsi, cette distance euclidienne soulève un problème : même si des utilisateurs apprécient les jeux d'une même catégorie, ils seront considérés comme très éloignés s'ils n'ont pas noté les mêmes jeux. Cette limitation s'applique particulièrement aux utilisateurs nouveaux ou peu actifs.

3.2 Distance cosinus

Une autre manière de mesurer la distance entre les utilisateurs consiste à regarder si leurs vecteurs pointent dans la même direction. La distance cosinus permet de quantifier cette mesure et de contourner les limitations posées par la distance euclidienne. Elle est définie de manière suivante :

Soient $u_1, u_2 \in \mathbb{R}^{N_j}$ deux vecteurs (utilisateurs) de notes

$$\text{dist}_{\text{cos}}(u_1, u_2) = 1 - \cos(\angle(u_1, u_2)) = 1 - \frac{u_1 \cdot u_2}{\|u_1\|_2 \cdot \|u_2\|_2}$$

3.3 Variantes étudiées

3.3.1 Biais des notes des utilisateurs

Chaque utilisateur est différent : certains notent plus sévèrement que d'autres. Afin de pouvoir comparer les notes des utilisateurs, il faut enlever leur biais sur les notes. Pour cela nous centrons les notes de chaque utilisateur autour de leur moyenne sur toutes leurs notes. Soit l'utilisateur u , sa moyenne des notes $\bar{m}_u$ et sa note m_{uj} pour le jeu j , alors la note centrée de l'utilisateur pour ce jeu est

$$m_{uj}^c = m_{uj} - \bar{m}_u$$

De ce fait, si un utilisateur a plutôt aimé un jeu alors $m_{uj}^c \geq 0$.

3.3.2 Pondération par distance

Une variante de la prédiction de note est celle basée sur la pondération par distance. En effet, plus un voisin de l'utilisateur-cible est proche de lui, plus sa note doit être prise en compte.

Soit $J_{u_c}^{abs}$ l'ensemble des jeux non notés (note absente) par l'utilisateur u_c , $\widehat{m}_{u_c j}$ la note prédite pour l'utilisateur u_c sur le jeu j , $w(u, u_c)$ la pondération appliquées entre les utilisateurs u et u_c et U_j l'ensemble des utilisateurs ayant noté le jeu j , il est possible d'estimer la note en calculant la moyenne pondérée par l'inverse de la distance :

$$\forall j \in J_{u_c}^{abs}, \widehat{m}_{u_c j} = \frac{\sum_{u \in U_j} w(u, u_c) \cdot m_{uj}}{\sum_{u \in U_j} w(u, u_c)}$$
$$w(u, u_c) = \frac{1}{d(u, u_c)}$$

Il est également possible d'appliquer une telle pondération pour la version avec les notes centrées.

3.4 Évaluation des différents modèles

Pour évaluer l'efficacité de notre modèle basé sur la similarité entre les utilisateurs, il est nécessaire de définir les fonctions de coût. Les métriques choisies pour quantifier nos erreurs sont **RMSE** (*Root Mean Square Error*) et **MAE** (*Mean Absolute Error*). MAE indique les écarts en moyenne en donnant le même poids aux grandes et aux petites erreurs. Par contre, RMSE, pénalise davantage les grands écarts. Nous souhaitons avoir un modèle qui minimise les grands écarts, tout en ayant une faible erreur en moyenne, d'où le choix de ces métriques. La base de données des notes sera divisée en une base d'apprentissage et de test.

La première méthode d'évaluation consiste à cacher 20% des notes existantes partout dans la matrice et de les prédire. Afin d'éviter le biais dans la base de test, nous prendrons la moyenne sur 3 évaluations.

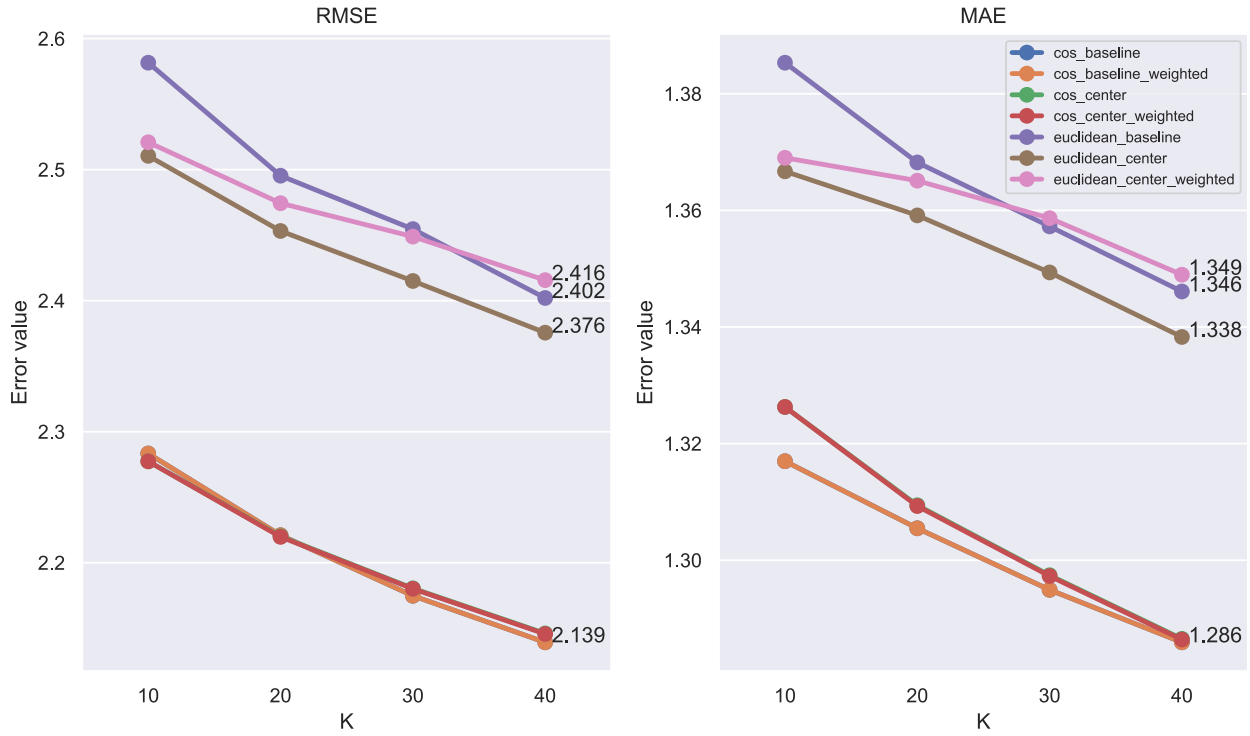


FIGURE 5 – RMSE, MAE en fonction de K voisins²

Nous observons (figure 5) que la mesure de similarité avec cosinus a produit les meilleurs résultats. Cependant, ni le **centrage** des notes, ni la **pondération** par distance n'ont permis d'améliorer la performance de ce modèle³.

Il est également intéressant de regarder les utilisateurs pour lesquels les erreurs étaient les plus élevées. Pour cela, nous divisons les utilisateurs par **tranches** en fonction du nombre d'avis qu'ils ont donnés. Plus un utilisateur a noté de jeux, plus son profil est complet. Nous pouvons donc nous attendre à obtenir des taux d'erreurs différents par tranche.

Nous n'évaluons que les modèles basés sur la **similarité cosinus** avec et sans l'élimination du biais (centrage) étant donné qu'ils ont montré les meilleurs résultats. La figure 6 avec les boîtes à moustaches montre que le taux d'erreur est plus élevé pour les utilisateurs qui ont donné peu d'avis (entre 10 et 50). De plus, les erreurs sont plus dispersées dans cette tranche ; il existe des utilisateurs pour lesquelles le taux d'erreur est supérieur à 5 points. Ainsi, le **modèle baseline** est plus performant que la version centrée sur tous les utilisateurs sauf les moins actifs (le taux d'erreur monte jusqu'à 7 points). Pour les deux modèles le taux d'erreur et son étendue diminuent à mesure que le nombre d'avis par tranche augmente, ce qui prouve la **robustesse** d'un profil complet.

2. baseline : version sans le centrage des notes, ni pondération par distance

center : version avec le centrage des notes

weighted : version avec la pondération par distance

3. la courbe bleue est derrière l'orange, la courbe verte de même est derrière la rouge

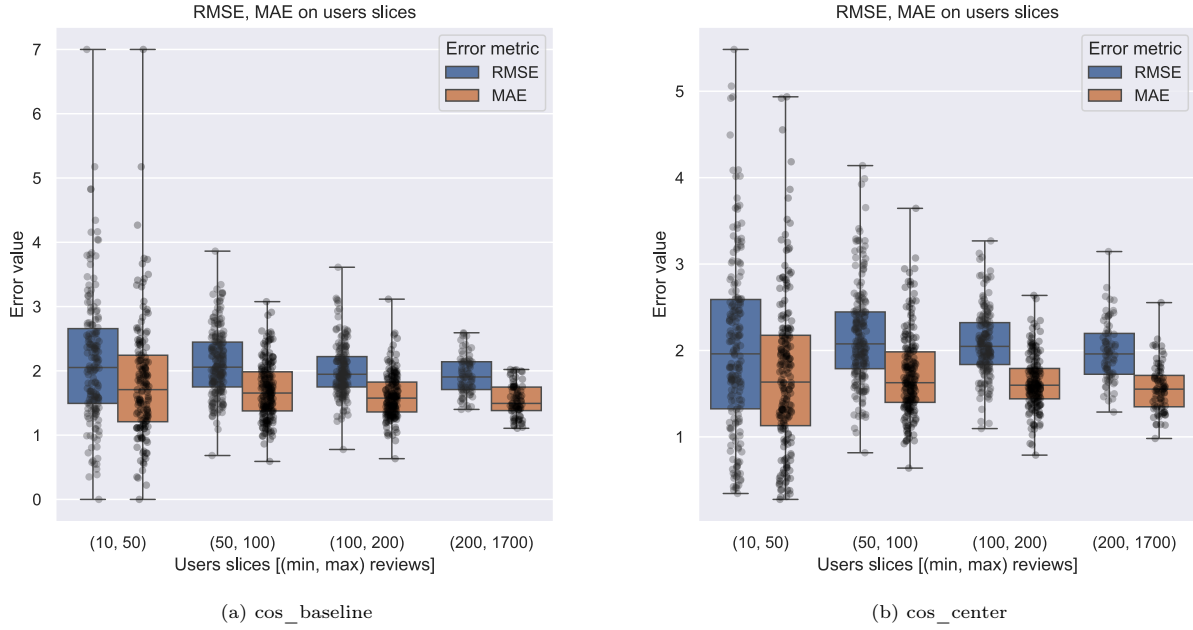


FIGURE 6 – Performance de cosinus par tranches

3.5 Limitations de k-NN

Un défaut majeur de l'estimation des notes par voisinage c'est le taux d'erreur très élevé pour les utilisateurs peu actifs, ce qui veut dire que les recommandations données ne sont pas satisfaisantes pour eux, pourtant 74% de nos utilisateurs ont donné moins de 50 avis. Ceci est un problème de filtrage collaboratif - afin de donner des bons résultats, nous avons besoin d'avoir pas mal de données sur l'utilisateur. De plus, si personne parmi les utilisateurs similaires n'a noté le jeu, alors il ne sera jamais recommandé. C'est aussi un problème de cold-start du côté des jeux.

Nous passerons par conséquent au filtrage par contenu afin de surmonter ce problème et de recommander des jeux selon leur similarité aux jeux déjà notés. Cela implique qu'il nous faut grouper les jeux similaires, ce qui revient à trouver des clusters de jeux, par exemple en appliquant un algorithme de K-means.

Un algorithme de K-means s'appuie sur les distances entre les points pour maximiser la similarité entre les éléments qui appartiennent au même cluster. La matrice $M_{u,j}$ souffre d'un fléau de dimension en raison de sa sparsité⁴ et de sa taille. La distribution des distances entre les utilisateurs est présentée en annexe. Dans la matrice des distances euclidiennes, 60% des valeurs sont manquantes, tandis que dans celle des distances cosinus, 60% des valeurs correspondent à la valeur maximale. Par conséquent, ces mesures de distance perdent leur importance. Il nous faut donc un moyen de réduire la taille de cette matrice tout en gardant les relations sous-jacentes.

4. Pour rappel, 98.1% des valeurs sont manquantes

4 Estimation des notes par factorisation de matrices

Nous allons donc nous intéresser à la réduction de dimension par factorisation de matrices afin de dépasser les limites causées par le fléau de dimension et faire ressortir les tendances implicites. Nous nous pencherons sur deux méthodes de factorisation : SVD (*Singular Value Decomposition*) et NNMF (*Non-Negative Matrix Factorization*). Notre matrice M_{uj} étant non-négative, nous choisirons NNMF pour la réduction de dimension. Elle nous permettra de trouver les relations cachées interprétables entre les jeux et les utilisateurs, ainsi que de regrouper, grâce à l'application de K-means, des jeux similaires, nous rapprochant du filtrage par contenu. Nous nous intéresserons également à la visualisation des clusters obtenus à l'aide de la méthode non-linéaire t-SNE (*t-distributed Stochastic Neighbor Embedding*).

4.1 SVD

Le principe de la **D**écomposition en **V**aleurs **S**ingulières est de factoriser une matrice rectangulaire M à coefficients dans le corps $K = \mathbb{R}$ en produit de 3 matrices :

$$M = U \Sigma V^T$$

$U_{m \times m}$ et $V_{n \times n}$: matrices orthogonales
 $\Sigma_{m \times n}$: matrice diagonale contenant les valeurs singulières⁵ de M
rangées dans l'ordre décroissant

Cette décomposition permet d'extraire les **facteurs latents**⁶ qui peuvent expliquer la relation entre les utilisateurs et les jeux et il est possible de garder uniquement les dimensions les plus influentes en ne considérant que les k plus grandes valeurs singulières.

Cependant, la méthode classique du SVD ne peut pas être appliquée dans notre cas car la matrice utilisateur-jeux M_{uj} contient beaucoup de valeurs nulles représentant à la fois des notes à 0 et des notes manquantes. Pour contourner ce problème de valeurs manquantes, une approche alternative a été proposée par Simon Funk lors du *Netflix Prize* en 2006. Son idée était d'éviter le calcul explicite de Σ et d'approximer $M_{uj} = PQ^T$ en effectuant une descente du gradient stochastique régularisée pour éviter le surapprentissage. Le but c'est de minimiser :

$$\min \left\| M - PQ^T \right\|^2 + \lambda_P \|P\|_F^2 + \lambda_Q \|Q\|_F^2 = \sum_{(u,j) \in E} (m_{uj} - \widehat{m}_{uj})^2 + \underbrace{\lambda_P \|p_u\|^2 + \lambda_Q \|q_j\|^2}_{\text{régularisation}} \quad (1)$$

p_u : vecteur des facteurs latents d'utilisateur u dans P
 q_u : vecteur des facteurs latents de jeu j dans Q
 λ : paramètres de régularisation

La note est prédite en prenant le produit scalaire et en la remettant à l'échelle :

$$\widehat{m}_{uj} = \mu + p_u q_j^T$$

μ : moyenne des notes existantes

Cependant, la présence de valeurs positives et négatives dans les matrices factorisées complexifie l'interprétation des facteurs latents. Nous utiliserons une autre méthode présentée dans les années 90 : la **F**actorisation **M**atricielle **N**on **N**égative (NNMF).

4.2 NNMF

L'idée de la NNMF est la même que celle de la SVD : factoriser la matrice de départ en identifiant des facteurs latents tout en imposant la contrainte que toutes les valeurs soient positives :

$$M_{n \times m} \simeq P_{n \times k} \cdot Q_{k \times m}$$

$$M, P, Q \in \mathcal{M}(\mathbb{R}^+)$$

k : nombre de facteurs latents $\leq \min(n, m)$

Comme en SVD, la matrice P (resp. Q) représente les utilisateurs (resp. les jeux) selon les facteurs latents. Ainsi, la formule de minimisation 1 et la note prédite sont définies de la même manière que pour SVD. Le pas

5. Racines carrées des valeurs propres

6. Facteurs abstraits

de descente du gradient est adapté afin d'annuler les termes négatifs lors de la mise à jour (voir l'annexe pour les formules complètes).

Il faut noter que nous n'utiliserons pas les biais des utilisateurs et des jeux : en les ajoutant, la factorisation devient susceptible de surapprentissage.

4.2.1 Nombre de facteurs latents

Afin d'avoir une idée du nombre de facteurs latents à considérer, nous procédons par validation croisée sur 5 itérations. Cette méthode est donc similaire à celle que nous avons utilisée pour k-NN consistant à cacher 20% de notes partout dans la matrice pour mesurer la qualité des notes prédites tout en gardant les mêmes métriques RMSE et MAE que nous avons utilisé pour évaluer le modèle basé sur le voisinage des utilisateurs (section 3). Selon la méthode du coude (*elbow method*) les valeurs optimales sont 20, 25 et 30 (figure 7). Au-delà de 35 facteurs, l'erreur de prédiction augmente, ce qui suggère que la factorisation capture le bruit et généralise moins bien. Dans les sections suivantes, nous appliquerons un clustering sur les jeux représentés selon les facteurs latents. Nous avons testé les valeurs 20, 25 et 30 pour le nombre de facteurs, et obtenu la meilleure compacité des clusters avec 20 facteurs latents. C'est donc cette valeur que nous retenons pour la suite.

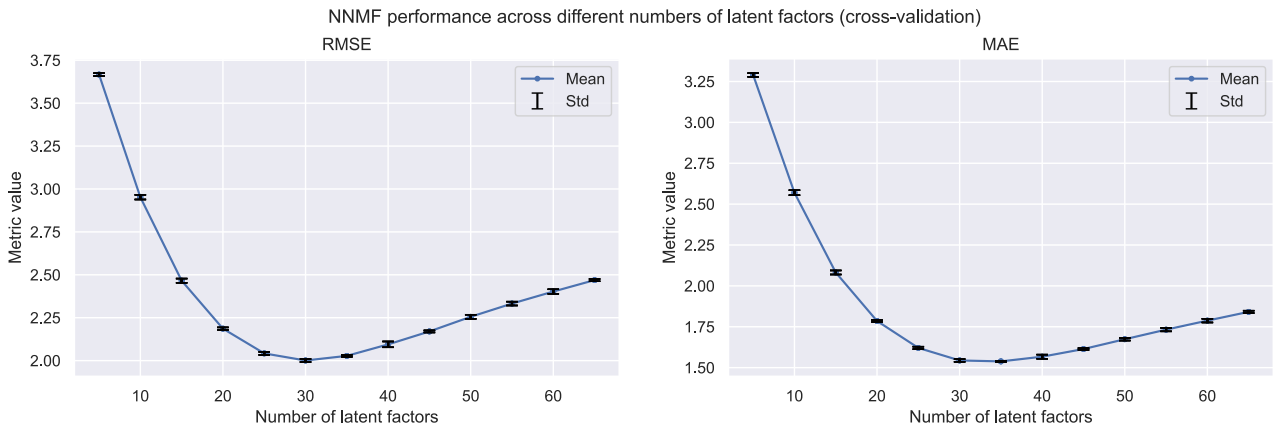


FIGURE 7 – Performance de NNMF pour différent nombre de facteurs latents

4.2.2 K-means

Une fois notre réduction de dimension obtenue avec NNMF, nous allons identifier avec K-means les bons clusters de jeux découlant des facteurs latents. Le clustering aide au système de recommandation en plusieurs manières : il permet de diversifier des recommandations en explorant d'autres clusters proches et permet de surmonter le problème du cold-start en considérant les clusters contenant les jeux déjà notés par l'utilisateur nouveau.

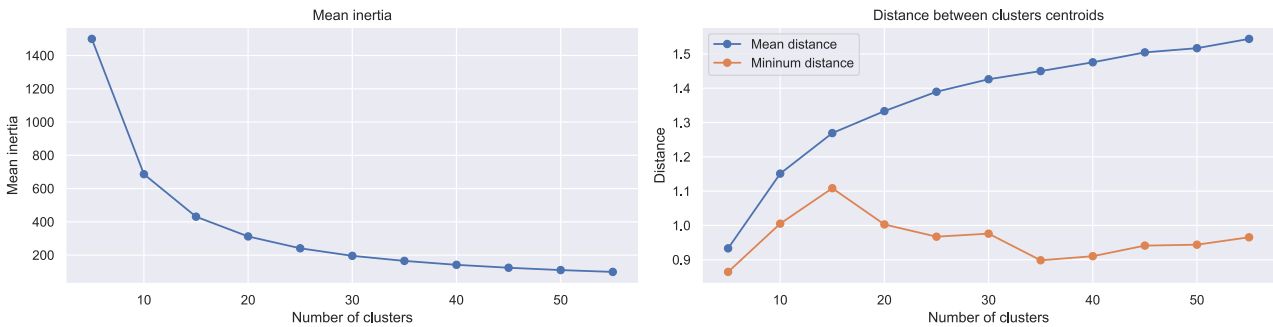


FIGURE 8 – Évolution des métriques en fonction de K dans K-means

Afin de trouver la bonne valeur pour K pour un algorithme de K-means, nous procédons par la méthode du coude. Elle nous permet de remarquer que 15, 20, 25 et 30 seraient des valeurs pertinentes de K. Nous verrons par la suite la visualisation de ces clusters en essayant plusieurs valeurs de K. La meilleure visualisation a été obtenue avec $K = 30$.

Avec $K = 30$, nous obtenons des clusters bien répartis contenant en moyenne 87 jeux (figure 9). Cette figure montre également la dispersion des distances des points par rapport au centroïde : nous observons que certains

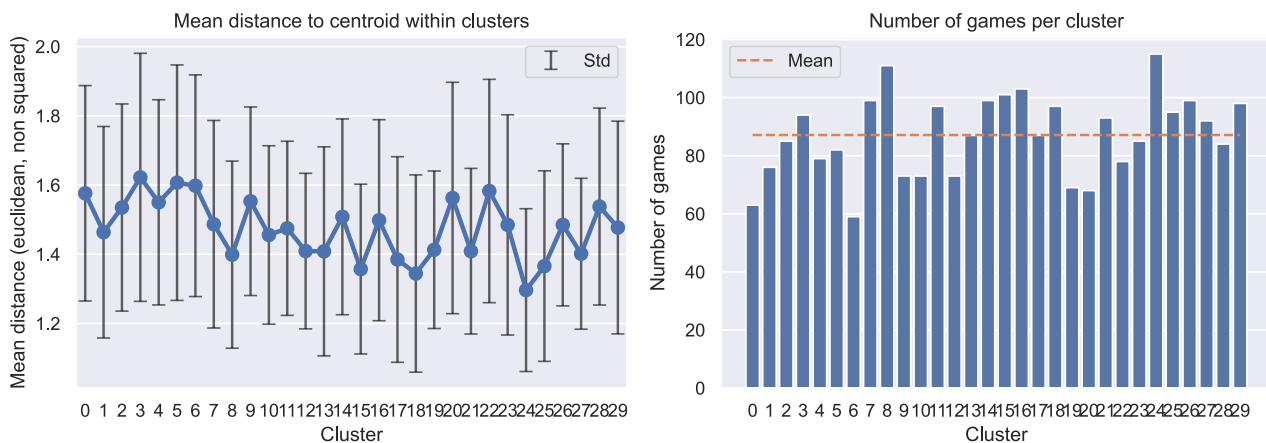


FIGURE 9 – Compacité des clusters et leurs tailles

clusters sont plus compacts que d'autres, indépendamment du nombre de points. Par exemple, le cluster 24 est celui avec le nombre maximum de points, pourtant la variation des distances y est minimale.

4.2.3 Visualisation à l'aide de t-SNE

Pour pouvoir représenter les données réduites avec NMF et colorer les clusters obtenus, nous utilisons la méthode statistique **t-SNE** qui permet de réduire la dimension en 2D. Cette méthode est non-linéaire et, par conséquent, permet de séparer les données qui ne sont pas linéairement séparables, ce qui aide à préserver le clustering présent dans une grande dimension, contrairement aux méthodes de réduction linéaires, par exemple PCA. L'avantage de t-SNE est qu'elle préserve les distances locales : les points proches les uns des autres dans une dimension élevée auront de fortes chances d'être également proches dans la dimension réduite.

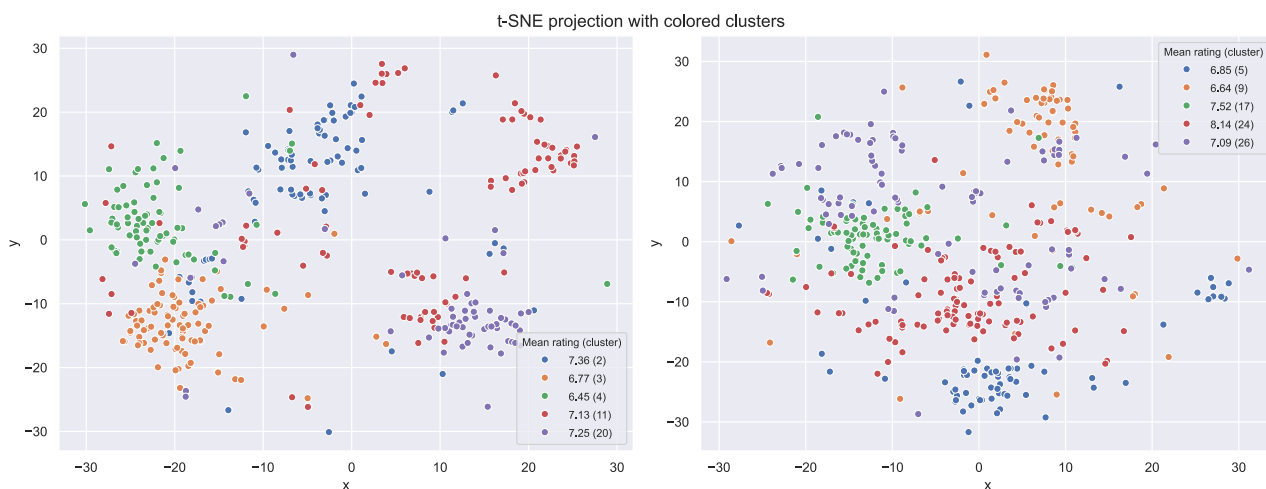


FIGURE 10 – Visualisation des jeux avec t-SNE (10 clusters parmi 30)

La figure 10 montre une partie de visualisation des jeux avec des clusters coloriés différemment. La visualisation de tous les 30 clusters est disponible dans l'annexe. Il est important de noter que, lors de la visualisation faite avec t-SNE, la distance réelle entre les clusters n'est pas forcément préservée. Ainsi, la taille du cluster ⁷ n'est pas non plus conservée : t-SNE a tendance à élargir les clusters denses et à resserrer ceux qui sont dispersés.

Afin d'expliquer les clusters nous avons tout d'abord analysé les catégories des jeux au sein de chaque cluster. Toutefois, nous n'avons pas réussi à conclure : chaque cluster mélange plusieurs catégories, dont la répartition est illustrée sur la figure en annexe. Une autre source plus riche permettant d'expliquer ces clusters sont les descriptions textuelles des jeux. L'objectif est d'identifier des thématiques partagées ou des motifs au sein d'un même cluster. Cette approche nécessite d'utiliser des techniques de traitement automatique du langage naturel (NLP).

7. Ici, il ne s'agit pas du nombre de points dans le cluster, mais de la surface qu'il occupe

5 NLP : explication à l'aide des commentaires et des descriptions

Dans cette section nous allons traiter et analyser des commentaires et des descriptions, ce qui fait partie du traitement automatique du langage naturel (*Natural Language Processing*). Notre premier objectif est d'expliquer les clusters de jeux obtenus à l'aide de la NNMF en exploitant les descriptions. Le second objectif est plus global : il s'agit d'expliquer la recommandation d'un jeu à un utilisateur en générant un commentaire qu'il pourrait formuler à propos de ce jeu.

5.1 Pré-traitement des commentaires

En analyse de texte, une étape préliminaire est de filtrer les mots et de définir le **corpus**, c'est-à-dire l'ensemble des mots pertinents lors de l'analyse des commentaires. Parmi l'ensemble des mots retrouvés dans les commentaires de la base de données, il faut enlever les mots trop fréquents ou qui n'apportent pas d'information utile appelés **stopwords** : les articles, pronoms, conjonctions etc. Nous devons également supprimer toute règle grammaticale des verbes, noms et adjectifs, alias les ramener à leur **lemme**, leur forme de base : l'infinitif pour les verbes, le masculin singulier pour les noms et adjectifs. Nos commentaires sont en français et nous avons environ 96 000 commentaires à lemmatiser avec plus de 18 000 de mots différents. Nous passons donc par le lemmatiseur rapide *TreeTagger*. Il faut noter qu'il arrive que des commentaires contiennent des fautes d'orthographe, ce qui amenuise la performance du lemmatiseur.

Une fois les mots lemmatisés, nous construisons le corpus en choisissant 5000 mots qui ressortent le plus en terme de fréquence, parmi les verbes, les adjectifs, les noms et la négation. Nous filtrons ensuite les commentaires pour ne contenir que les mots du corpus et ne gardons que ceux ayant au moins 10 mots.

5.2 Modèles-transformeurs

Par la suite, nous aurons besoin des modèles-transformeurs : décodeur et encodeur. Les modèles-transformeurs ont été présentés en 2017 dans un article de recherche très célèbre intitulé "*Attention is all you need*". À ce jour, ils restent des modèles *state-of-the-art*⁸ en NLP. Le décodeur nous permettra de générer du texte, tandis que l'encodeur servira à transformer le texte en vecteurs capturant les relations sémantiques entre les mots.

5.2.1 LLM

Le **LLM** (*Large Language Model*) est un modèle d'intelligence artificielle basé sur l'architecture transformeur-décodeur. Il est capable de comprendre et de générer du langage humain. Le modèle que nous avons choisi c'est LLaMa3.1, à 8 milliards (8B) de paramètres, nous l'utiliserons en locale sur nos machines. Le choix du modèle s'est fait principalement sous la contrainte de RAM minimale nécessaire. Sachant que nos ordinateurs ont 8Gb de RAM, ce modèle n'a besoin que de 4.7Gb. Cela représente un bon compromis entre le nombre de paramètres et l'usage de RAM.

En utilisant LLaMa nous pourrons, par exemple, résumer les commentaires d'un jeu ou les descriptions de plusieurs jeux, ou bien générer le commentaire qu'un utilisateur pourrait écrire sur un jeu.

5.2.2 Modèle-encodeur

Les modèles encodeurs permettent de transformer le texte en vecteurs, aussi appelés *embeddings*, qui à ce jour permet de capturer au mieux le sens des phrases. L'encodeur permet de placer le texte dans un espace de grande dimension, chacune des dimensions encodant un certain facteur latent. Les textes similaires, même si les mots utilisés sont différents, seront proches dans cette espace.

Le modèle-encoder que nous avons choisi c'est BAAI/BG3. Son gros avantage c'est qu'il est multilingue (nos commentaires sont en français) et il peut embedder à la fois 8192 tokens $\sim$ 6144 mots. Ce modèle transforme le texte passé en paramètre en un vecteur de dimension 1024.

5.3 Explication des clusters

Afin de pouvoir expliquer les clusters que nous avons trouvés grâce aux facteurs latents de NNMF, nous analysons les **descriptions** des jeux. Notre idée est d'utiliser le modèle-encodeur BAAI/BG3 pour transformer les descriptions en vecteurs. De cette manière, nous avons pour chaque cluster un ensemble des embeddings (représentations vectorielles) des descriptions des jeux qui y appartiennent. Ensuite, pour chaque cluster nous pouvons trouver la description représentative en calculant le centre des descriptions associées au cluster. Par

8. C'est à dire les modèles les plus performants

le suite, nous noterons cette description D_r qui varie pour chaque cluster. Afin de pouvoir décoder cette description et de comprendre le sens d'un cluster, il faut identifier les descriptions les plus proches à la description représentative trouvée.

Plutôt que de poser un nombre fixe de descriptions à sélectionner autour de D_r , nous déterminons ce nombre en fonction de la distance minimale trouvée. Tout d'abord nous trouvons la description qui est à la distance minimale d_{min} de D_r . Ensuite, nous ne retenons uniquement que les descriptions dont la distance à D_r est inférieure à 125% de d_{min} .

Une fois les descriptions proches à D_r identifiées, nous regardons les bigrammes⁹ les plus fréquents et nous faisons également un résumé de ces descriptions avec LLaMa3.1.

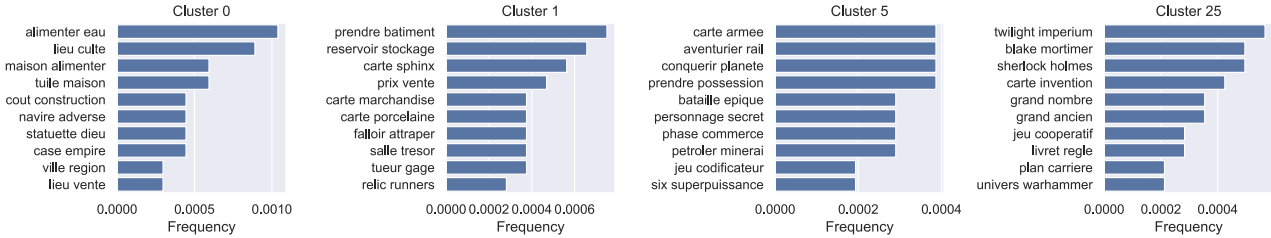


FIGURE 11 – Bigrammes les plus fréquents et explicites pour les 4 clusters

La figure 11 montre un exemple de clusters dans lesquels les bigrammes montrent explicitement la thématique générale. Pour ces 4 clusters nous voyons des sujets (dans l'ordre des images, de gauche à droite) : construction et expansion, collection des objets, guerre et conquête, jeux longs et complexes.¹⁰

L'analyse des bigrammes peut cependant devenir fastidieuse, notamment lorsque plusieurs sujets sont mélangés. Par exemple, nous avons plusieurs clusters des jeux complexes et les bigrammes seuls ne suffiront pas pour identifier telle thématique. C'est pour cette raison que les résumés produits par LLM rendent l'analyse du sens des clusters beaucoup plus facile et permettent de repérer des sujets cachés.

En général, nos 30 clusters sont associés à des thématiques telles que la déduction et la logique, les jeux longs et complexes, la collection d'objets (cartes ou trésors), la construction et l'expansion, les jeux de type civilisation, les eurogames, les jeux de guerre, les jeux rapides tactiques, les jeux de très bonne qualité (bons éléments visuels, mécaniques bien pensés), ou encore des jeux avec des univers culturels, exotiques ou fantastiques. Nous avons également un cluster regroupant les jeux les mieux notés (cluster 24), ainsi qu'un autre qui contient les jeux les pires notés (cluster 10), visible dans la figure 12.

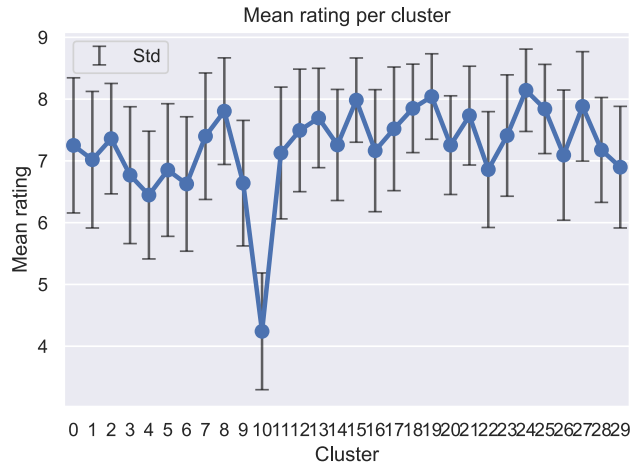


FIGURE 12 – Note moyenne des jeux par cluster

5.4 Génération des commentaires

Afin de voir s'il est possible d'expliquer la recommandation du jeu, nous allons générer le commentaire que l'utilisateur-cible pourrait donner au jeu recommandé grâce aux commentaires d'autres utilisateurs.

5.4.1 Limitation de LLM

Tout d'abord, nous avons essayé de simplement utiliser LLaMa pour générer le commentaire en lui fournissant en prompt¹¹ l'ensemble des avis laissés sur ce jeu écrits par des utilisateurs similaires. Le problème de cette méthode c'est le fait que le temps du calcul peut être assez consécutif¹², surtout si le prompt passé est très grand. C'est le cas pour la génération de commentaire vu que nous donnons au LLM plusieurs commentaires

9. Nous n'analysons pas les unigrammes car ils sont très génériques et dans la majorité des cas il est difficile d'en déduire le contexte

10. Le jeu *Twilight Empirium* est très connu pour sa durée (de 2h à 10h) et la complexité des décisions stratégiques. De même, le jeu *Sherlock Holmes – Détective Conseil* repose sur la réflexion et la logique et peut durer plus d'1h30.

11. Instruction ou série de données fournies au modèle d'IA pour générer des réponses

12. Par exemple, génération d'un seul commentaire peut prendre 5-15 minutes, cela dépend de la taille du prompt

entiers. De plus, la fenêtre de contexte (*context window*¹³) est limitée pour chaque grand modèle de langage. Par exemple, pour Llama3.1 la taille maximale est de 128 000 tokens $\sim$ 96 000 mots (prompt et réponse générée compris).

Afin de contourner cette limitation il nous faut une manière de choisir des mots ou des phrases pertinents pour réduire significativement la taille du prompt et ensuite les passer au LLM. Il pourra ainsi les combiner et de produire un commentaire.

5.4.2 Prédiction par fréquence lexicale des voisins

Nous commençons par le moyen le plus simple de sélectionner des termes : sélectionner les plus fréquents parmi les commentaires des voisins. L'idée est que si un terme est très fréquent chez ses voisins, nous pouvons nous attendre à ce qu'il soit aussi présent dans le commentaire de notre utilisateur.

Pour quantifier l'importance des mots et améliorer le filtrage des commentaires, nous utilisons le **TF-IDF**¹⁴ (*Term Frequency - Inverse Document Frequency*). Cela nous permet de garder dans chaque commentaire, les mots les plus importants dans le document par rapport à l'ensemble des commentaires. Nous pouvons donc récupérer les notions essentielles et réduire le bruit en enlevant des mots peu pertinents, qui peuvent rendre le commentaire généré incohérent ou trop générique.

Dans un premier temps, nous réalisons une analyse simple et directe des mots en considérant chaque mot comme une unité lexicale : les unigrammes. Toutefois, analyser les mots individuellement peut entraîner une perte d'informations contextuelles et négliger des dépendances entre les mots. C'est pourquoi nous analysons aussi les bigrammes afin d'enrichir le contexte.

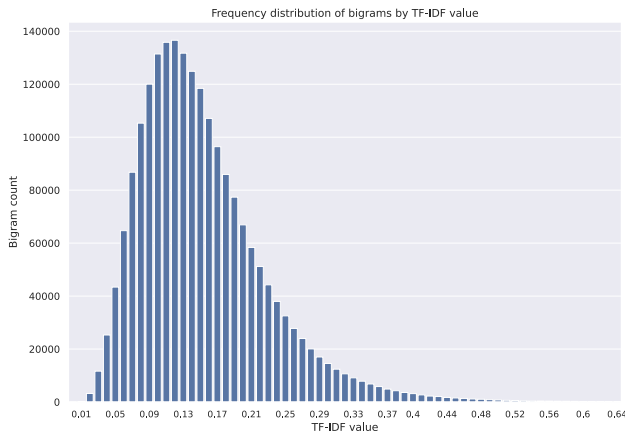


FIGURE 13 – Répartition des bigrams par TF-IDF

Pour cela, nous allons filtrer tous les commentaires des voisins en ne conservant que les termes au TF-IDF **supérieur** à certain seuil. Ensuite, pour limiter la taille du commentaire, nous ne sélectionnons que les X termes les plus fréquents dans ces commentaires. Le choix du seuil a été déterminé en analysant la répartition du TF-IDF des bigrammes comme montré dans la figure 13. Pour supprimer les mots trop communs, nous posons le seuil juste après le pic, à 0.13.

Le choix de X quant à lui a été fait en observant l'évolution des scores ROUGE-2. Comme constaté dans la figure 14, le score se stabilise autour de la valeur 150, que nous prendrons comme seuil pour X.

Nous verrons une évaluation quantitative dans la section 5.5.1.

5.4.3 Prédiction par phrases des voisins

Une autre approche, plus complexe, consiste à utiliser des phrases extraites des commentaires des utilisateurs. Afin de transformer les phrases en vecteurs, nous utilisons le modèle-encodeur BAAI/BG3 présenté dans la section 5.2.2. Notons que les commentaires peuvent mélanger plusieurs sujets : les aspects positifs et négatifs des jeux, la description du matériel ou des techniques, comparaisons avec d'autres jeux. Par conséquent, nous avons choisi d'embedder des commentaires par phrases au lieu de commentaires tout entiers.

Ensuite, nous avons besoin de grouper des phrases proches en sens en appliquant l'algorithme des K-means sur ces embeddings. Au total, nous avons 590 000 phrases différentes et nous fixons le nombre de clusters à 500. Nous éliminons les clusters trop dispersés (la section dans l'annexe donne plus d'informations sur ces clusters.)

Une fois les phrases réparties par clusters et les mauvais clusters éliminés, nous pouvons chercher des phrases pour former un commentaire de l'utilisateur-cible u_c sur un jeu j . Nous souhaitons choisir 15 phrases. Pour cela nous procédons de manière suivante :

1. Trouver k utilisateurs similaires à u_c , trouver les clusters dans lesquels des phrases de leurs commentaires sur le jeu j apparaissent. Notons c_k le nombre de clusters identifiés.

13. Un nombre de tokens que LLM peut considérer à un instant

14. Dans notre cas un document est un commentaire

2. Idéalement, il faut prendre une phrase par cluster pour diversifier au plus le commentaire. Cela peut varier selon les utilisateurs similaires et c_k trouvé à l'étape précédente. Nous avons 2 cas :
 - Si $c_k \leq 15$, alors dans chaque cluster nous sélectionnons $15/c_k$ phrases. Celles-ci sont choisies en calculant le barycentre des phrases du cluster, puis en identifiant les $15/c_k$ phrases les plus proches par distance euclidienne.
 - Si $c_k > 15$, alors il faut une manière de choisir 15 clusters parmi c_k . Nous le faisons en donnant des poids pour chaque cluster, ces poids sont basés sur l'entropie de Shannon.

Plus précisément, pour chaque cluster c , il y a N_u ¹⁵ probabilités associées, avec une probabilité p_u par utilisateur u . On définit pour l'entropie p_u comme la proportion des phrases de u qui appartiennent à c .

De cette manière, plus l'entropie est haute, plus les phrases dans le cluster sont spécifiques. À l'inverse, plus l'entropie est basse, plus les phrases sont communes et génériques. Afin de diversifier le type de phrases, nous prenons la moitié des clusters avec une entropie faible et l'autre moitié avec une entropie élevée.

Nous verrons dans la section 5.5.2 une évaluation quantitative.

5.5 Évaluation quantitative : score BLEU-ROUGE

En NLP, il existe 2 métriques permettant de comparer un texte de référence à un texte généré (ou prédit) :

- Le score **BLEU** (*Billual Evaluation Understudy*) est orienté vers la **précision** : il mesure la proportion de mots du texte en sortie qui apparaissent dans le texte de référence.
- Le score **ROUGE** (*Recall-Oriented Understudy for Gisting Evaluation*) quant à lui est similaire au **rappel** : il mesure le nombre de mots dans le texte de référence capturé par le texte en sortie.

Ici, nous considérons le texte de référence comme le commentaire laissé sur un jeu et supposons que le commentaire "prédit" d'un utilisateur est représenté par la fusion des commentaires de ses voisins sur le même jeu. En notant *hypothesis* le commentaire prédit et *reference* le commentaire de notre utilisateur, la différence principale entre le score BLEU et le score ROUGE repose sur la partie principale de leur formule :

Score BLEU (voir l'annexe pour la formule complète) :

$$p_n = \frac{\sum_{n\text{-grams} \in hypothesis} Count_{match}(n\text{-grams})}{\ell_{hyp}^{n\text{-grams}}}$$

Score ROUGE :

$$ROUGE = \frac{\sum_{n\text{-grams} \in reference} Count_{match}(n\text{-grams})}{\ell_{ref}^{n\text{-grams}}}$$

$\ell^{n\text{-grams}}$: taille du texte en nombre de n -grammes

Il existe plusieurs versions de ROUGE : ROUGE-N, basé sur les n -grammes ; ROUGE-L, basé sur la plus longue sous-séquence commune ; et ROUGE-S, basé sur les paires de mots qui ne sont pas forcément adjacents mais avec au plus S mots d'écarts entre eux. Il faut noter que le score ROUGE peut également être calculé comme un score F1. Toutefois, nous utiliserons uniquement le rappel sachant que notre objectif est de capturer un maximum d'éléments essentiels du commentaire de référence, sans pénaliser les cas où le commentaire généré est plus long que le commentaire de référence.

Afin de pouvoir évaluer de la même manière nos 2 méthodes de la génération des commentaires présentées dans les sections d'avant, nous nous concentrons sur le score ROUGE-N, avec $N = 1$ (unigrammes) et $N=2$ (bigrammes) et le score BLEU-2 (unigrammes et bigrammes).

Pour une **évaluation**, nous procédons de manière suivante : pour un ensemble donné d'utilisateurs (les 50 plus actifs ou 200 piochés aléatoirement), nous traitons chaque utilisateur, noté par la suite utilisateur-cible u_c , un par un en cachant 10% de ses notes et en les prédisant à l'aide de $k = 40$ utilisateurs les plus proches selon la distance cosinus. Pour la prédiction de commentaire, nous gardons uniquement les jeux dont la note a été bien prédite (au plus 2 points d'écarts par rapport à la vraie note). Soit nb_u le nombre d'utilisateurs parmi les k

¹⁵. Nombre d'utilisateurs au total. Pour rappel, pour nous $N_U = 1945$

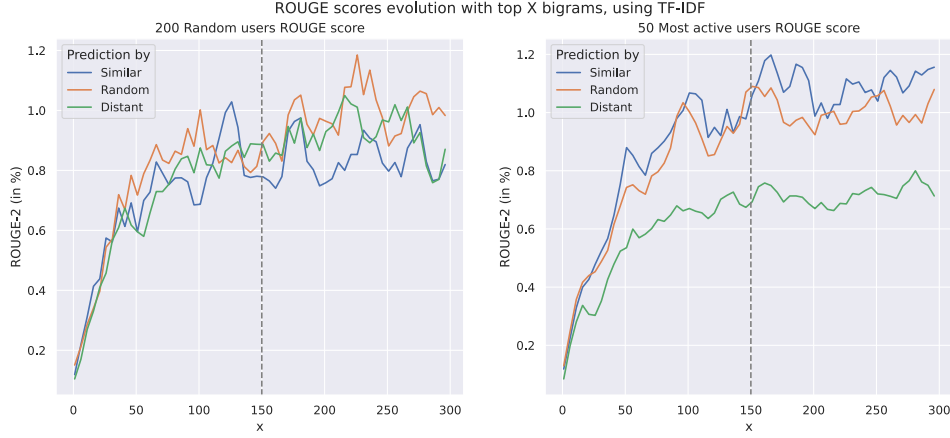


FIGURE 14 – Score ROUGE en prenant X bigrammes les plus fréquents avec TF-IDF

voisins qui ont noté le jeu ($nb_u \leq k$). Pour prédire le commentaire que u_c pourrait donner à un jeu recommandé, nous prenons les commentaires des nb_u utilisateurs proches, et calculons les scores BLEU et ROUGE pour estimer la qualité de prédiction. Pour pouvoir comparer si cela vaut la peine de prendre des utilisateurs proches, nous prédisons également le commentaire en prenant nb_u utilisateurs aléatoires ainsi que nb_u utilisateurs moins similaires par rapport à u_c parmi les utilisateurs ayant noté le jeu.

5.5.1 Prédiction par bigrammes et unigrammes

Les scores BLEU des prédictions sont réalisés en utilisant la librairie *SacreBLEU*. Comme la fonction d'évaluation utilisée *sentence_score()* prend en paramètre les hypothèses et les références en tant que phrases et non comme liste de tokens, il n'est pas possible d'évaluer le score BLEU pour les bigrammes après application du filtrage par TF-IDF. En effet, nous décomposons les commentaires des voisins en tokens pour filtrer ces derniers, avant de les rassembler en une unique phrase. En passant cette nouvelle phrase à la fonction de score, cette dernière va, lors de la tokenisation¹⁶, générer de nouveaux bigrammes et donc fausser le résultat.

Par exemple, les bigrammes [(‘hello’, ‘world’), (‘data’, ‘science’)] rassemblés puis re-décomposés en bigrammes donnera la liste [(‘hello’, ‘world’), (‘world’, ‘data’), (‘data’, ‘science’)].

De ce fait, le score BLEU calculé sur les bigrammes ne concerne que les commentaires non filtrés par TopX et TF-IDF, comme présenté dans la figure 15. Les scores ROUGE sur les commentaires filtrés avec TopX et TF-IDF sont illustrés en figure 16.

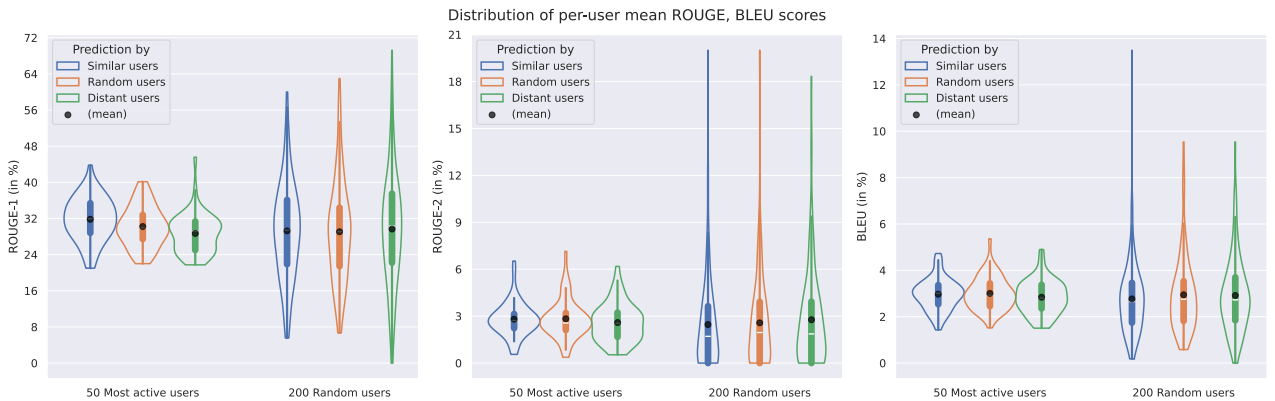


FIGURE 15 – Scores ROUGE-1, ROUGE-2, BLEU sans application de TF-IDF ni de TopX

Notons qu'en sélectionnant toutes les unités lexicales, il n'y a pas de réduction et nous ne pourrions pas utiliser LLM pour générer le commentaire cohérent en temps raisonnable. De ce fait, la version sans TopX et sans TF-IDF nous servira donc comme un axe de comparaison.

16. Découpage d'un texte brut en unités linguistiques, les tokens

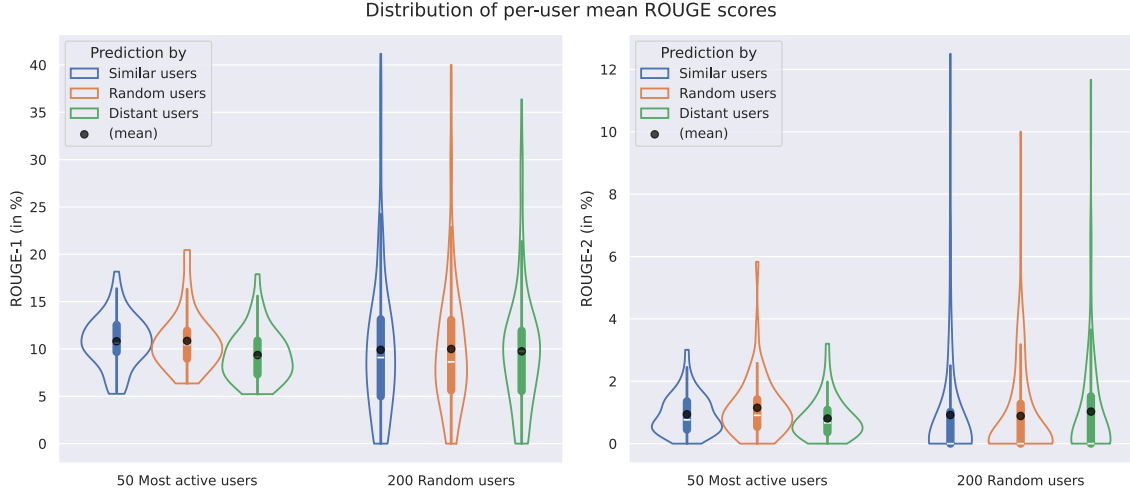


FIGURE 16 – Scores ROUGE-1, ROUGE-2, BLEU avec application de TF-IDF et TopX

L'application de TF-IDF et TopX fait baisser les scores ROUGE-1, ROUGE-2 et BLEU de moitié vu que les bigrammes et les unigrammes les plus génériques sont éliminés et que nous prenons moins de mots. Dans les deux versions, nous n'observons pas de différence significative en fonction du type d'utilisateurs pris en compte pour prédire le commentaire. Notons que le score ROUGE-2 est assez faible même si on prend spécifiquement des bigrammes pour la prédiction.

Dans l'annexe se trouve la distribution des scores ROUGE-1, ROUGE-2 par jeu pour 15 utilisateurs les plus actifs et 15 aléatoires¹⁷.

Sans le filtrage par TF-IDF et l'application de TopX nous pouvons pas non plus espérer d'avoir un score BLEU élevé vu que, de manière générale, en prenant tous les mots la taille de commentaire est largement surestimée.

5.5.2 Prédiction par phrases

La figure 17 illustre la distribution des scores moyennes en prenant les différents types des utilisateurs pour la prédiction. Il n'y a pas non plus de différence notable de scores en prenant des utilisateurs proches, aléatoires ou distants. Remarquons que, même en utilisant un nombre réduit d'unigrammes et de bigrammes, les scores ROUGE-1, ROUGE-2 restent similaires à ceux de la figure 15, ce qui indique une bonne couverture lexicale malgré la sélection. Le score BLEU est plus élevé, car nous prenons un nombre limité des phrases et, par conséquent, un nombre limité d'unigrammes et de bigrammes - les commentaires prédits sont donc de taille raisonnable.

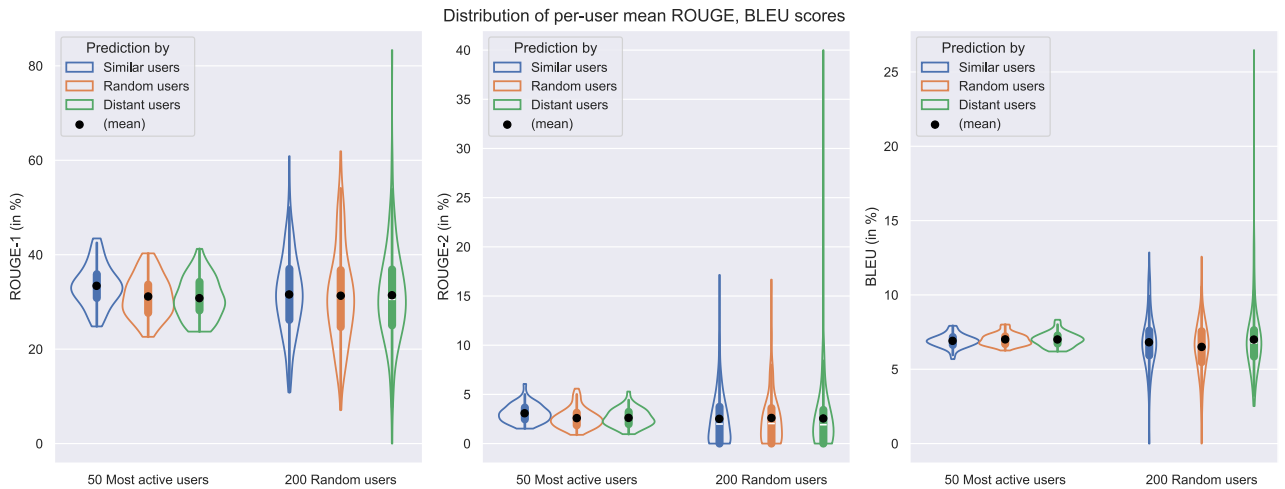


FIGURE 17 – Scores ROUGE-1, ROUGE-2, BLEU

17. Nous avons pris 15 utilisateurs pour rendre les graphiques visibles dans le rapport

Comme pour la méthode de prédiction par bigrammes et unigrammes, l'annexe contient la distribution des scores ROUGE-1 et ROUGE-2 par jeu. Il est intéressant de se pencher sur les utilisateurs les plus actifs, car beaucoup de notes cachées ont été bien prédites pour eux. Ces graphiques permettent de remarquer qu'il existe des jeux pour lesquels nous avons réussi à avoir un score ROUGE-1 à plus que 60%, voire 100% pour un jeu, ce qui sont des résultats assez encourageants.

5.6 Evaluation qualitative : LLM

Nous avons vu dans la section précédente qu'il n'y a pas de grosse différence en scores si nous prenons des utilisateurs similaires, aléatoires ou distants pour la prédiction de commentaire. Par conséquent, nous faisons également une évaluation qualitative : nous fournissons des bigrammes / phrases au LLM pour qu'il génère le commentaire et comparons le commentaire généré avec celui réellement donné au jeu.

Par exemple, prenons le jeu *Zombie Dice* et le vrai commentaire de l'utilisateur du nom *Gigi* :

"Zombie Dice est un petit jeu de stop ou encore sans prétention mais aussi sans grande saveur. On ne réfléchit pas beaucoup et la chance est omniprésente. Le thème bien qu'anecdotique colle finalement bien aux mécanismes... Qu'importe ! Le but ici est plutôt de se dérouiller la cervelle (à défaut d'en manger !) après un gros jeu ou encore pendant un apéritif. On s'amuse de la malchance des adversaires et on se moque des prises de risque trop audacieuses qui achoppent. Même si les parties sont très courtes, on n'en fait pas plusieurs d'affilées tant le jeu devient vite répétitif et lassant. Zombie Dice ne restera pas dans les annales du genre. Dans la même catégorie il y a des jeux bien plus réussis et plus attractifs tant par leur matériel que par le fun qu'ils engendrent autour d'une table. Il ne suffit pas de coller des cerveaux sur des dés à 8 faces et une image de zombie racoleuse sur une boîte pour transformer un banal jeu de dés en distraction originale."

Les mots surlignés en bleu représentent les points essentiels constituant l'avis du joueur. Ici, nous remarquons que le jeu est assez rapide, se base plus sur la chance que sur les compétences des joueurs et devient vite ennuyeux aux yeux de notre utilisateur. Par la suite, nous écrirons en vert les aspects correctement repris du commentaire original, et en rouge, le choix des mots qui au contraire, vont à l'encontre de l'avis initial.

5.6.1 Prédiction par bigrammes, unigrammes

	Proches	Aléatoires	Distants
ROUGE-1	17.91	16.27	16.86
ROUGE-2	0.66	0.51	1.09

TABLE 1 – Scores des prédictions par bigrammes et unigrammes filtrés pour le jeu *Zombie Dice*

Prédiction grâce aux utilisateurs :

Proches (bigrammes) : « Après cette partie, je pense que les jeux sont un peu trop répétitifs. Le jeu de découverte est vraiment la part la plus agréable du jeu, mais il y a beaucoup de fois où tout se fait par erreur et alors on se rend compte qu'il n'y a pas de règles pour savoir si c'est vrai ou faux, ce qui pénètre dans l'esprit de déception. Je pense que cela devrait être rectifié afin d'éviter ces erreurs et de rendre le jeu plus agréable à jouer (je ne suis pas sûr si c'est réellement possible mais il faut essayer). »

Aléatoires (unigrammes) : « J'ai été dépassé par un jeu de défaut qui est une boîte de chance. Je suis tombé dans l'abîme d'une zombie des décès (je ne comprends pas pourquoi je dois encore jouer...). Et pourtant, la règle du jeu s'appelait 'dice', et c'est avec ce tour que j'ai eu le défi de réussir à sortir de cette situation désastreuse. Malheureusement, il y a d'autres défis que les zombies des décès peuvent me porter ! Mais comme un vrai géant du monde virtuel, je préfère continuer à jouer pour essayer de gagner...mais c'est pas facile dans ce jeu-là. »

Distants (bigrammes) : « Laissez les zombies aller, il est trop facile. Je trouve la situation très rassurante pour un jeu de rôle comme ça. C'est toujours plus agréable de faire des révélations sur le plan et les défauts du personnage à partir d'un lancement rapide, mais la qualité et la chance ont beaucoup changé tout le jeu en plus. Les mains sont désormais devenues les yeux, et il y a si peu de main qui se cache dans les ruses de ce jeu... Mais puisque c'est facile comme ça, on peut aussi jouer à stop comme un jeu d'arcade ! »

On remarque que les scores ROUGE-1 sont similaires pour les utilisateurs proches, aléatoires et distants. Pour les scores ROUGE-2, seuls les utilisateurs distants ont un score plus élevé que les deux autres, même si les scores restent très faibles ici. Nous constatons que les commentaires générés à partir des bigrammes ou des unigrammes reprennent parfois quelques notions du commentaire de notre utilisateur, mais leur nombre est faible. Les bigrammes ou unigrammes peuvent aussi être interprétés dans le mauvais sens : le commentaire

généralisé avec les unigrammes indique que l'utilisateur souhaite continuer à jouer, ce qui n'est pas le cas. Enfin, nous notons que cette génération conduit aussi à des phrases qui n'ont aucun sens, comme la phrase en italique dans le commentaire des utilisateurs distants.

5.6.2 Prédiction par phrases

	Proches	Aléatoires	Distants
ROUGE-1	42.5	40.0	43.75
ROUGE-2	6.33	6.33	5.06
BLEU	5.29	7.59	7.14

TABLE 2 – Scores des prédictions pour le jeu *Zombie Dice*

Prédiction grâce aux utilisateurs :

Proches (commentaire c_1) : « Je vais dire que *Zombie Dice* est un bon choix pour un **apéritif rapide** et amusant, car **il se joue très vite et on passe à autre chose**. Le thème est attrayant et **les dés ont différentes répartitions de faces**, ce qui modifie les probabilités. Mais pour moi, le jeu est **trop simpliste** et aléatoire, et je préfère apparemment les jeux plus stratégiques. Cependant, il est simple à expliquer et à jouer, même si ça me semble **un peu basique** en l'état. En fin de compte, c'est un bon moyen pour discuter avec vos amis pendant **l'apéritif**, mais **je ne pense pas qu'il ajoute trop de profondeur au jeu**. »

Aléatoires (commentaire c_2) : « C'est un jeu de société basique, mais ultra amusant ! Il repose sur le principe classique du **"stop ou encore"**, mais avec un thème original de zombie qui peut surprendre et déranger certaines sensibilités. Les parties sont **rapides** et **ne nécessitent pas trop de réflexion**, ce qui en fait idéal pour des séances courtes de 10 minutes. Le **matériel est correct**, sauf le "sac à dés" qui n'est pas fourni, et il est facile de s'adapter aux règles avec un peu de pratique. Il est **simple**, rapide et **basé sur la chance**, ce qui en fait une bonne alternative au "Jeu des cochons". En résumé, c'est un jeu de **stop ou encore** très amusant et idéal pour un moment de **détente**. »

Distants (commentaire c_3) : « **Je l'aime !** C'est un **jeu qui va faire réveiller tes neurones d'après-midi**, c'est **drôle**, sympa et **parfait pour finir une soirée** jeux avec des collègues ou la famille. Le principe est simple mais ça fait de vous, on lance les dés, on **prend des risques**, on **hésite**... et souvent, on en sort avec le dos au mur ! C'est un jeu qui demande un peu de stratégie, mais autant que la malchance aux dés puisse vous faire perdre votre sain-saige pour le grand prix. Idéal pour **les soirées apéritives ou fin de session ludique** quand on veut faire rire et **se détendre** en même temps. »

Nous observons que les scores ROUGE-1, ROUGE-2 sont proches pour les prédictions en prenant les utilisateurs proches, aléatoires et distants. Cependant, le sens de certaines phrases n'est celui de l'utilisateur-cible. Par exemple, l'utilisateur n'a pas aimé le matériel (le commentaire c_2 a dit le contraire), ou bien le commentaire c_3 a mentionné que le jeu fait réfléchir, mais l'utilisateur a dit l'inverse. En effet, les scores ROUGE-1, ROUGE-2 et BLEU ne peuvent pas capturer le sens global des phrases ; les unigrammes et les bigrammes peuvent apparaître dans des contextes très différents — même s'ils sont présents dans un commentaire de référence. Notons que le commentaire généré grâce aux utilisateurs similaires est le seul qui ait capturé le bémol du jeu *Zombie Dice* d'un point de vu de l'utilisateur-cible, c_1 est donc le commentaire qui se rapproche le plus du commentaire de référence.

6 Conclusion

Nous avons ainsi, au cours de ce projet, pu étudier les deux grandes familles d’algorithme de recommandation afin de pouvoir expliquer les **recommandations** faites à des utilisateurs dans le contexte des jeux de sociétés en passant d’abord par le filtrage collaboratif avec k-NN, qui a une vision local avec les voisins et NNMF, qui a une vision globale avec toute la matrice, puis le filtrage par contenu avec le clustering sur les jeux. Le filtrage collaboratif souffre fortement d’un problème de cold-start à la fois côté utilisateurs et côté jeux. Le filtrage par contenu, quant à lui, nous permet de limiter le problème du cold-start et de diversifier les propositions en ayant la possibilité d’explorer d’autres clusters.

Grâce à l’utilisation des LLM pour l’analyse de texte, nous avons pu expliquer nos clusters de jeux grâce à leurs descriptions. Nous avons également exploré l’analyse des commentaires des utilisateurs pour expliquer la recommandation. Si le choix des voisins ne donnait pas de différence significative au niveau des scores, nous avons toutefois constaté que la prédiction et la génération du commentaire pour l’utilisateur-cible avec le filtrage des **bigrammes** et des **unigrammes** était peu prometteurs. En revanche, la prédiction du commentaire avec les **phrases** des voisins a montré des résultats nettement plus encourageant.

Afin de visualiser l’étendu de notre travail, nous avons codé **l’application ELA**¹⁸ qui permet de visualiser les clusters de jeux en 3D de t-SNE et proposer des recommandations. Il y a 2 modes : **exploration**, qui permet de voir les différents clusters, et **recommandation**, où, étant donné une liste d’utilisateurs sélectionnables, on peut voir les prédictions de note à l’aide de NNMF, vu que c’est rapide et scalable, de cette utilisateur pour tous les jeux, et ainsi recommander les mieux notés. On peut voir pour chaque jeu ses informations (nom, catégorie, tranche d’âge, description), sa note moyenne ainsi que le type de cluster auquel il appartient. Nous prévoyons d’ajouter un résumé des commentaires de quelques jeux.

18. Acronyme des initiales des prénoms des membres du groupe, ou bien un rétroacronyme de "**E**xplicabi**L**ité des systèmes de re-comm**A**ndation"

Bibliographie

- Clement BM. (2021, décembre 23). *ROUGE and BLEU scores for NLP model evaluation*. ClementBM.github.io. <https://clementbm.github.io/theory/2021/12/23/rouge-bleu-scores.html#bleu>
- Papineni, Kishore & Roukos, Salim & Ward, Todd & Zhu, Wei Jing. (2002). *BLEU : a Method for Automatic Evaluation of Machine Translation*. 10.3115/1073083.1073135.
- Luo, M. Zhou, Y. Xia and Q. Zhu, *An Efficient Non-Negative Matrix-Factorization-Based Approach to Collaborative Filtering for Recommender Systems*, May 2014, doi : 10.1109/TII.2014.2308433.
- K. Doshi. (2021, mai 9). *Foundations of NLP explained — BLEU score and WER metrics*. TowardsDataScience.com. <https://towardsdatascience.com/foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b/>
- Y. Koren, R. Bell and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009, doi : 10.1109/MC.2009.263.
- Cedias. (2019). *RS1.ipynb — Recommender Systems Notebook*. GitHub. <https://github.com/cedias/reco2019/blob/master/RS1.ipynb>
- Cedias. (2019). *RS2.ipynb — Recommender Systems Notebook*. GitHub. <https://github.com/cedias/reco2019/blob/master/RS2.ipynb>
- Li, Yingxian & Xu, Junwu & Yang, Min. (2021). Collaborative filtering recommendation algorithm based on KNN and Xgboost hybrid. Journal of Physics : Conference Series. 1748. 032041. 10.1088/1742-6596/1748/3/032041.
- Wen, Zheng. (2008, dec 12). Recommendation System Based on Collaborative Filtering. <http://www.zheng-wen.com/WenRecommendation.pdf>
- Lü L., Medo M., Yeung C.H., Zhang Y.-C., Zhang Z.-K., Zhou T. (2012). *Recommender systems*. Physics Reports, 519(1), 1–49. <https://doi.org/10.1016/j.physrep.2012.02.006>
- Kumar, Balraj & Sharma, Neeraj. (2016). Approaches, Issues and Challenges in Recommender Systems : A Systematic Review. Indian Journal of Science and Technology. 9. 10.17485/ijst/2015/v8i1/94892.
- Chen, Li & Chen, Guanliang & Wang, Feng. (2015). Recommender systems based on user reviews : the state of the art. User Modeling and User-Adapted Interaction. 25. 10.1007/s11257-015-9155-5.
- Silveira, T., Zhang, M., Lin, X. et al. How good your recommender system is? A survey on evaluations in recommendation. Int. J. Mach. Learn. & Cyber. 10, 813–831 (2019). <https://doi.org/10.1007/s13042-017-0762-9>
- V. Guigue <https://vguigue.github.io/pres/chatGPT-Caf%C3%A9Agro.pdf>

Annexe

A Matrices de distance. Fléau de dimension

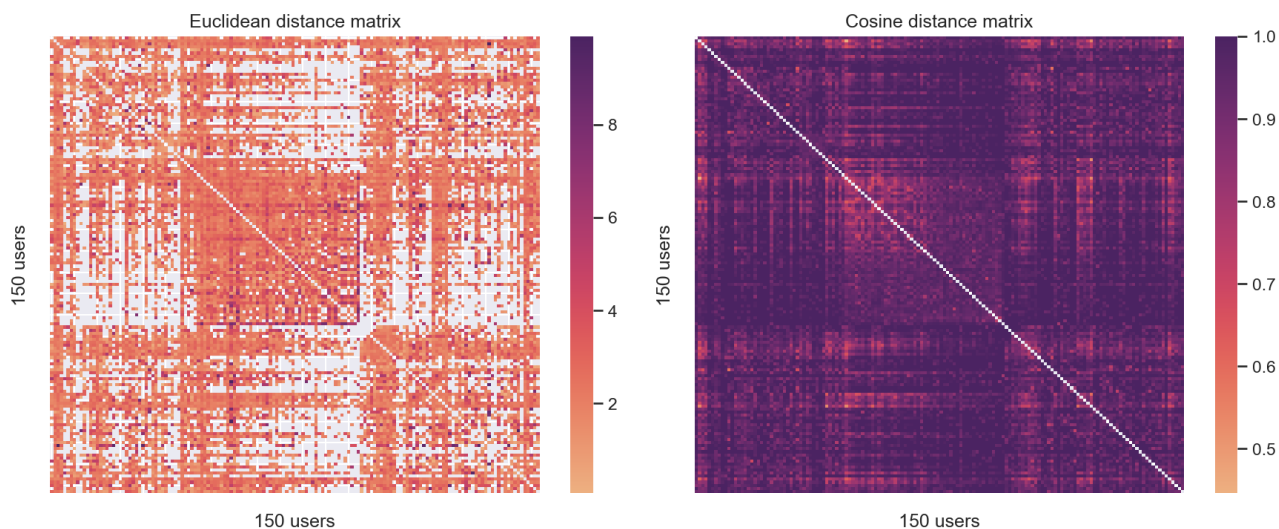


FIGURE 18 – Extrait des matrices de distance (150 premiers utilisateurs)

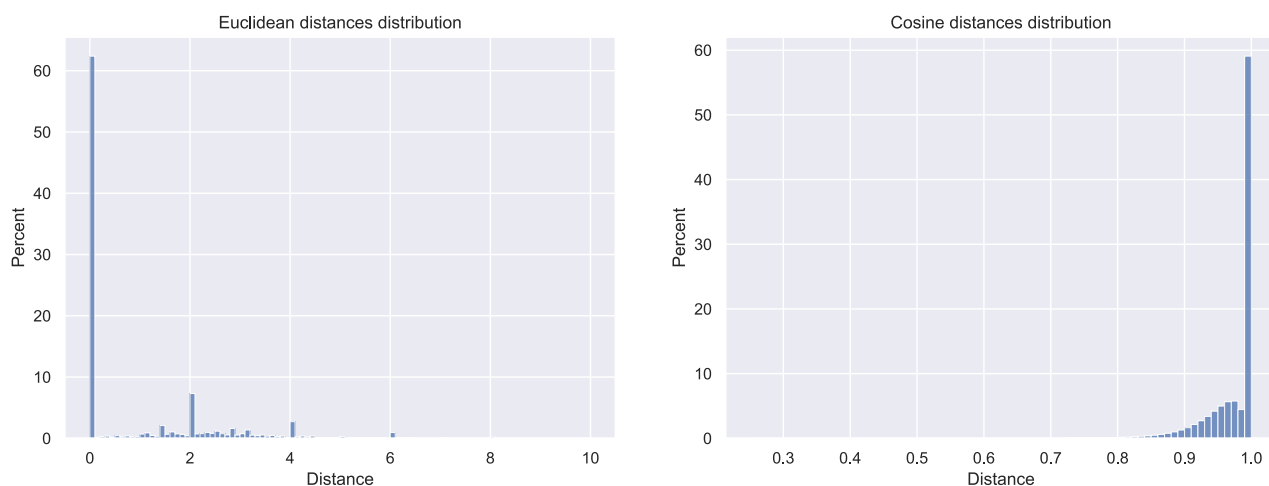


FIGURE 19 – Distribution des distances dans les matrices

Les matrices de distances entre les utilisateurs utilisées pour la méthode d'estimation des notes avec k plus proches voisins souffrent d'un fléau de dimension. Dans la matrice de distance cosinus 60% des valeurs correspondent à la distance maximale (1.0). Pour la matrice de distance euclidienne, 60% des valeurs sont manquantes, c'est-à-dire que les utilisateurs n'ont pas de jeux notés en commun.

B SVD. Descente de gradient

Le but de la descente de gradient pour la SVD est de minimiser l'erreur de prédiction quadratique (distance euclidienne entre les matrices) :

$$\min \|M - PQ^T\|^2 = \sum_{(u,j) \in E} (m_{uj} - \widehat{m}_{uj})^2$$

$E = \{(u, j) \mid m_{uj} \text{ note existante pour jeu } j \text{ et utilisateur } u\}$

M : matrice des notes donnés par les utilisateurs aux jeux

P : les vecteurs des utilisateurs selon les facteurs latents

Q^T : les vecteurs des jeux selon les facteurs latents

Afin d'éviter le surapprentissage, la régularisation avec la norme de Frobenius¹⁹ (notée $\|\cdot\|_F$) est ajoutée :

$$\min \|M - PQ^T\|^2 + \lambda_P \|P\|_F^2 + \lambda_Q \|Q\|_F^2 = \sum_{(u,j) \in E} (m_{uj} - \widehat{m}_{uj})^2 + \underbrace{\lambda_P \|p_u\|^2 + \lambda_Q \|q_j\|^2 + \underbrace{\lambda_{b_u} b_u^2 + \lambda_{b_j} b_j^2}_{\text{optionnel}}}_{\text{régularisation}} \quad (2)$$

p_u : vecteur des facteurs latents d'utilisateur u dans P

q_u : vecteur des facteurs latents de jeu j dans Q

b_u, b_j : biais d'utilisateur u et de jeu j resp.²⁰

λ : paramètres de régularisation

La note est prédite en prenant le produit scalaire et en la remettant à l'échelle :

$$\widehat{m}_{uj} = \mu + p_u q_j^T + \underbrace{b_u + b_j}_{\text{optionnel}}$$

μ : moyenne des notes existantes

Les facteur latents et les biais sont appris avec la descente du gradient régularisée :

$$p_{u,k} = p_{u,k} + \epsilon \sum_{j \in J_u} (d_{uj} \cdot q_{k,j} - \lambda_P \cdot p_{u,k})$$

$$q_{k,j}^{21} = q_{k,j} + \epsilon \sum_{u \in U_j} (d_{uj} \cdot p_{u,k} - \lambda_Q \cdot q_{k,j})$$

$$b_u = b_u + \epsilon \sum_{j \in J_u} (d_{uj} - \lambda_{b_u} \cdot b_u)$$

$$b_j = b_j + \epsilon \sum_{u \in U_j} (d_{uj} - \lambda_{b_j} \cdot b_j)$$

$$d_{uj} = m_{uj} - \widehat{m}_{uj}$$

J_u : ensemble des jeux notés par l'utilisateur u

U_j : ensemble des utilisateurs qui ont noté le jeu j

ϵ : le pas

19. définie comme $\sqrt{\sum_{i=0}^n \sum_{j=0}^m a_{ij}^2}$, $A \in \mathcal{M}_{n \times m}(\mathbb{R})$

20. peuvent être mis à 0 et la prédiction sera basée uniquement sur les facteurs latents

21. $q_{k,j}$ indique la valeur du vecteur q_j^T

C Biais des jeux et des utilisateurs

Les biais permettent de prendre en compte les déviations des notes des utilisateurs et des jeux par rapport à la moyenne μ .

Par exemple, dans notre base de données $\mu = 7.41$ et le jeu Antiquity a le biais 1.98 (c'est le plus grand biais par rapport aux autres jeux, ce jeu a 46 avis et 76% d'entre eux sont entre 8 et 10). Un tel biais signifie la tendance naturelle, intrinsèque des joueurs à attribuer des notes plus élevées à ce jeu par rapport à la moyenne.

Et inversement, si le jeu a tendance à recevoir des notes très basses, son biais sera négatif. C'est le cas pour le jeu Le morpion qui a le biais -4.09. En effet, ce jeu a 27 notes parmi lesquelles 77% sont entre 0 et 4.

Et de même pour les utilisateurs. La figure 20 représente la distribution de ces biais.

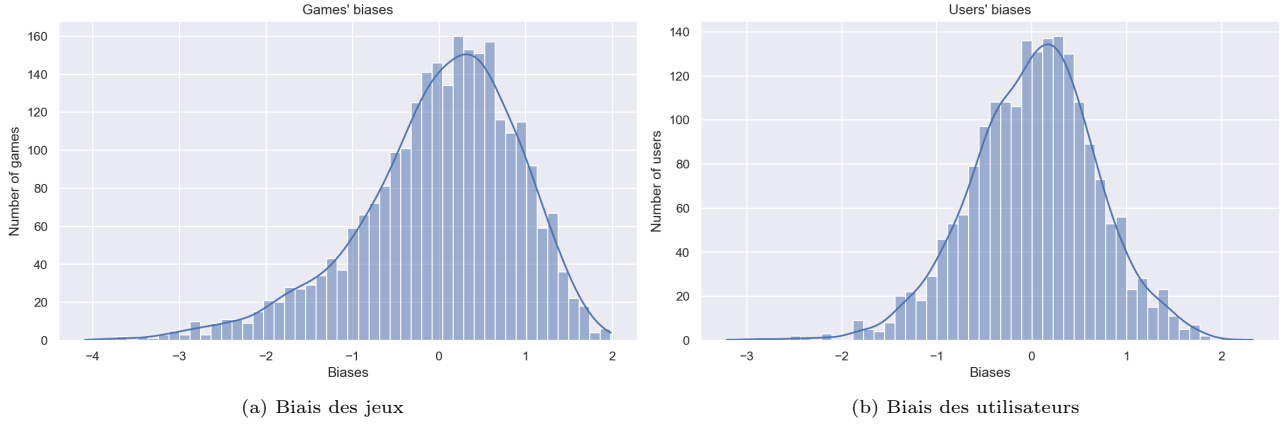


FIGURE 20 – Biais b_u, b_j

D NNMF. Descente de gradient

Pour la NNMF, la descente de gradient vise à minimiser la même fonction de coût 2 que pour SVD avec la contrainte de préserver la positivité des valeurs de P et Q . Afin de l'assurer, le pas de descente du gradient est adapté dans le but d'annuler les facteurs négatifs et, par conséquent, la mise à jour de p_u et q_j est adaptée. En considérant les formules de cette mise à jour,

$$\begin{aligned} p_{u,k} &= p_{u,k} + \epsilon \sum_{j \in J_u} ((m_{uj} - \widehat{m}_{uj}) \cdot q_{k,j} - \lambda_P \cdot p_{u,k}) \\ q_{k,j} &= q_{k,j} + \epsilon \sum_{u \in U_j} ((m_{uj} - \widehat{m}_{uj}) \cdot p_{u,k} - \lambda_Q \cdot q_{k,j}) \end{aligned}$$

il est facile de voir que les composantes négatives sont :

$$\begin{aligned} & -\epsilon \sum_{j \in J_u} (\widehat{m}_{uj} \cdot q_{k,j} + \lambda_P \cdot p_{u,k}) \text{ pour } p_{u,k} \\ & -\epsilon \sum_{u \in U_j} (\widehat{m}_{uj} \cdot p_{u,k} + \lambda_Q \cdot q_{k,j}) \text{ pour } q_{k,j} \\ & \lambda_P, \lambda_Q \in \mathbb{R}^+ \end{aligned}$$

Pour les annuler, chaque valeur de p_u et q_j , ϵ est remplacée par

$$\begin{aligned} \epsilon_{u,k} &= \frac{p_{u,k}}{\sum_{j \in J_u} (\widehat{m}_{uj} \cdot q_{k,j} + \lambda_P \cdot p_{u,k})} \text{ pour } p_{u,k} \\ \epsilon_{j,k} &= \frac{q_{k,j}}{\sum_{u \in U_j} (\widehat{m}_{uj} \cdot p_{u,k} + \lambda_Q \cdot q_{k,j})} \text{ pour } q_{k,j} \end{aligned}$$

Une fois la valeur de ϵ intégrée dans la formule et les composantes négatives annulées, on obtient la mise à jour suivante :

$$\begin{aligned} p_{u,k} &= p_{u,k} \cdot \frac{\sum_{j \in J_u} q_{k,j} \cdot m_{uj}}{\sum_{j \in J_u} (q_{k,j} \cdot \widehat{m}_{uj} + \lambda_P \cdot p_{u,k})} \\ &= p_{u,k} \cdot \frac{\sum_{j \in J_u} q_{k,j} \cdot m_{uj}}{\sum_{j \in J_u} (q_{k,j} \cdot \widehat{m}_{uj} + \lambda_P |J_u| p_{u,k})} \\ q_{k,j} &= q_{k,j} \cdot \frac{\sum_{u \in U_j} p_{u,k} \cdot m_{uj}}{\sum_{u \in U_j} (p_{u,k} \cdot \widehat{m}_{uj} + \lambda_Q \cdot q_{k,j})} \\ &= q_{k,j} \cdot \frac{\sum_{u \in U_j} p_{u,k} \cdot m_{uj}}{\sum_{u \in U_j} (p_{u,k} \cdot \widehat{m}_{uj} + \lambda_Q |U_j| q_{k,j})} \end{aligned}$$

La mise à jour des biais reste la même que pour SVD, et la note est prédite de la manière suivante :

$$\widehat{m}_{uj} = \mu + p_u q_j^T + \underbrace{b_u + b_j}_{\text{optionnel}}$$

μ : moyenne des notes existantes

E t-SNE sur NNMF, visualisation des clusters

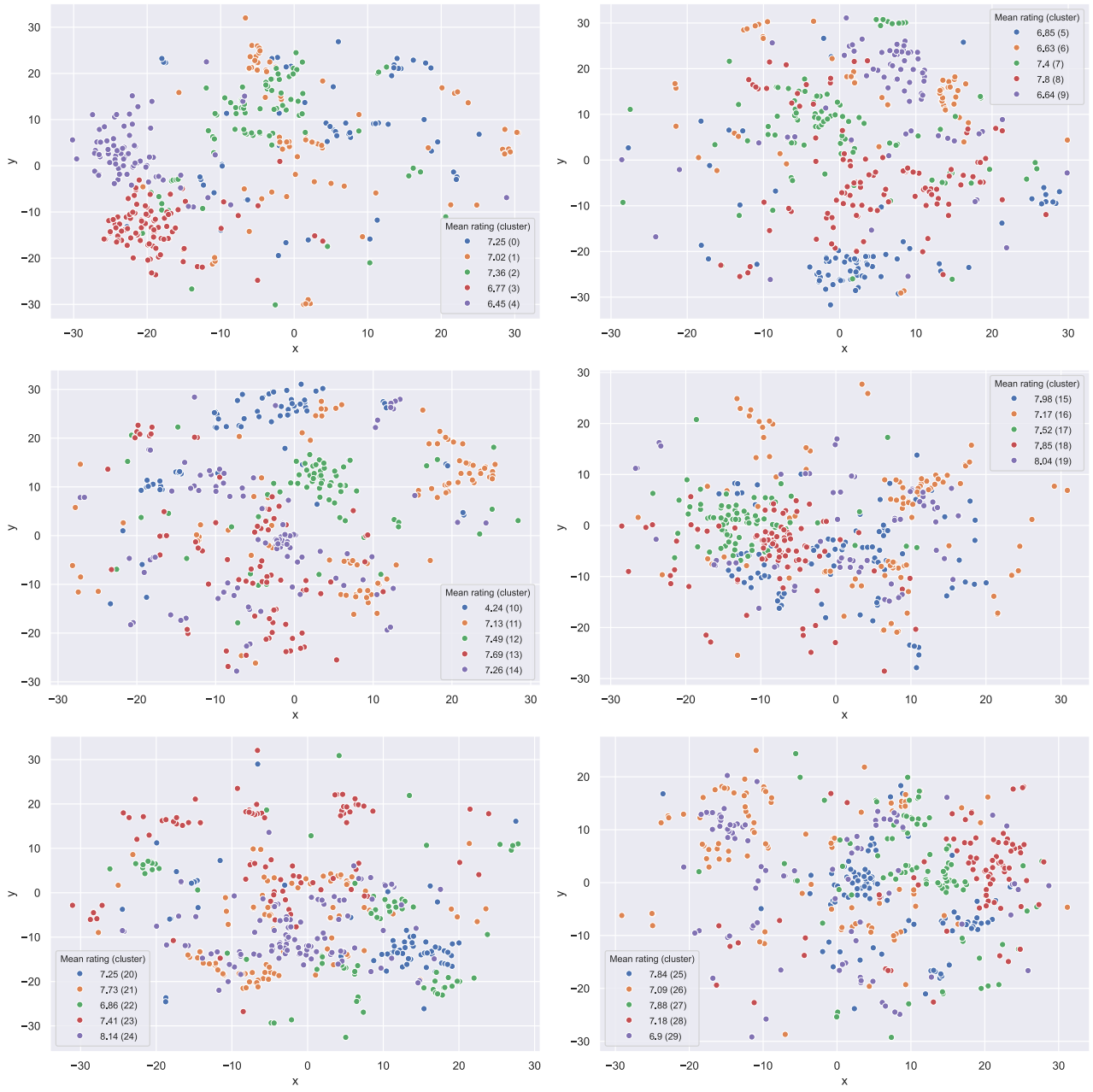


FIGURE 21 – Visualisation des tous 30 clusters des jeux

F Explication des clusters des jeux avec des leurs catégories

Avant de passer à l'explication des clusters à l'aide du NLP, nous avons d'abord essayé de les expliquer à partir des catégories de jeux. Nous avons commencé par supprimer les catégories très rares (moins de 20 occurrences) ainsi que les catégories trop génériques, comme jeux de plateau ou jeux de cartes. La figure 22 montre la distribution des fréquences au sein des différents clusters. Ces fréquences peuvent être interprétées comme des probabilités d'appartenance de chaque catégorie aux différents clusters. Nous observons que les clusters mélangent beaucoup catégories à la fois, ce qui rend difficile l'interprétation des clusters. De plus, nous avons vu dans la section d'explication des clusters à l'aide du LLM et des bigrammes qu'il existe des clusters qui regroupent des jeux longs et complexes, un facteur difficile à trouver en se basant uniquement sur les catégories.

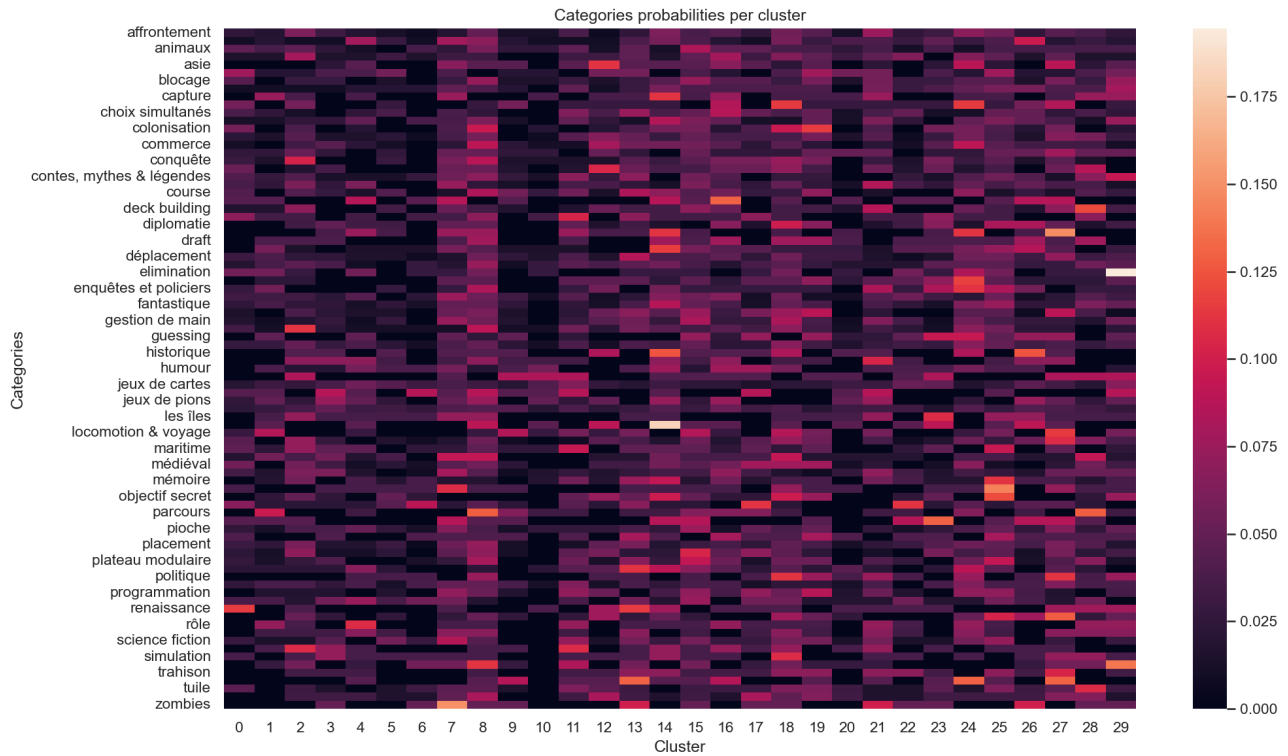


FIGURE 22 – Répartition des catégories des jeux par cluster

G K-means sur embeddings des phrases

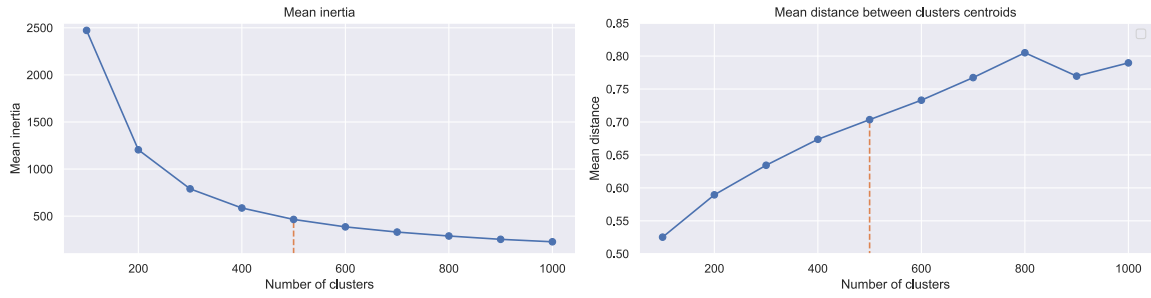


FIGURE 23 – Évolution des métriques en fonction de K dans K-means

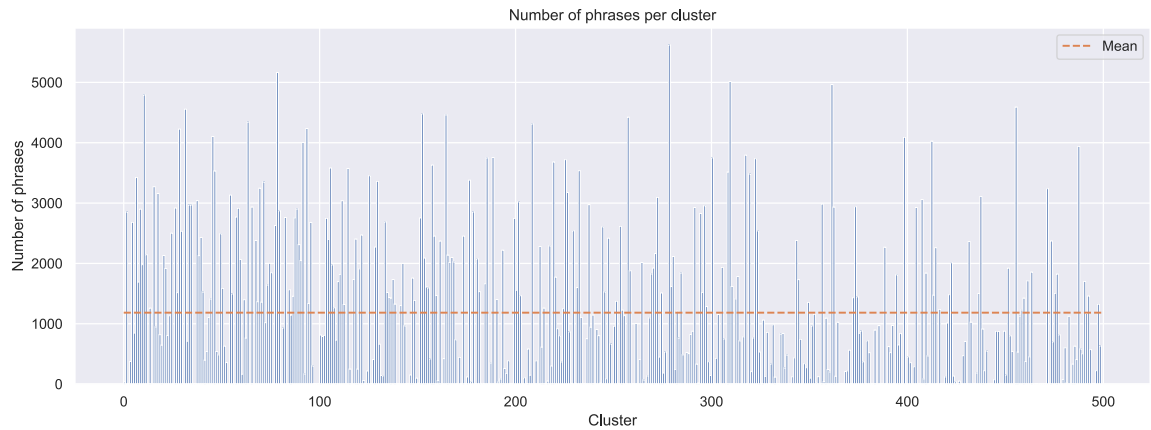


FIGURE 24 – Répartition des phrases par cluster

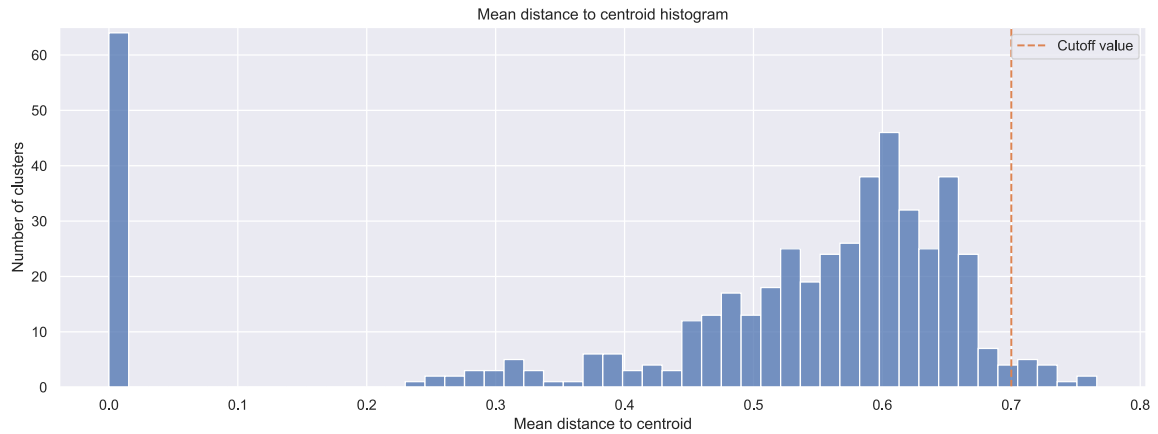


FIGURE 25 – Répartition des distances moyennes entre les embeddings et leurs centroïdes

La figure 24 montre qu'il y a des clusters beaucoup plus grands que les autres. Ils peuvent contenir des phrases génériques et communes. Par contre, les clusters plus petits, peuvent contenir des phrases très spécifiques. Par exemple, un des petits clusters, qui ne contient que 5 phrases, regroupe des phrases où les sites sont mentionnés. Par exemple, *"possibilité d'y jouer online sur www"*. Par ailleurs, un autre cluster qui regroupe plus que 1000 phrases qui parlent de différentes possibilités ou de plusieurs choix. Par exemple, *"les différentes possibilités sont également intéressantes"* ou *"aujourd'hui, il existe en plein de versions et"*.

Nous éliminons également certains clusters qui sont moins compacts que les autres. Pour cela, nous calculons, pour chaque cluster, la distance moyenne entre les points (embeddings) et le centroïde. La distribution de ces distances moyennes est représentée dans la figure 25. En effet, si la distance moyenne vers le centroïde est élevée, alors ce cluster peut contenir des phrases rejetées par des autres clusters, plus compacts, et par conséquent, les phrases dans ces clusters sont moins cohérentes sémantiquement. Nous fixons un seuil de distance à 0.7.

H Score BLEU, formule

Nous détaillons le score BLUE. En notant *hypothesis* le commentaire prédit et *reference* le commentaire de l'utilisateur-cible, le score BLEU est défini comme suit :

$$\begin{aligned} BLEU_N &= BP \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right) \\ BP &= \begin{cases} 1 & \text{si } \ell_{hyp} > \ell_{ref} \\ e^{1-\ell_{ref}/\ell_{hyp}} & \text{sinon} \end{cases} \\ p_n &= \frac{\sum_{n\text{-grams} \in hypothesis} Count_{match}(n\text{-grams})}{\ell_{hyp}^{n\text{-grams}}} \\ w_n &= \frac{1}{N} \quad \text{pondération uniforme des } n\text{-grams}^{22} \\ \ell^{n\text{-grams}} &: \text{taille du texte en nombre de } n\text{-grammes} \end{aligned}$$

La pénalité de brièveté (BP : Brevity Penalty) est un facteur qui permet de pénaliser les prédictions trop courtes par rapport au texte de référence, donc ne couvrant pas toute l'information et rendant le score non représentatif de la qualité de la prédiction. Elle est multipliée à la moyenne géométrique des p_i qui est utilisée pour équilibrer les précisions des différentes n-grammes.

²². Définie comme tel dans K. Papineni, S. Roukos, T. Ward, & W.-J. Zhu, BLEU : A method for automatic evaluation of machine translation (2002). <https://doi.org/10.3115/1073083.1073135>

I Score ROUGE, distribution des scores

En complément des diagrammes en violon nous ajoutons également la distribution des scores ROUGE-1, ROUGE-2 pour nos 2 méthodes de sélection des mots / phrases pour la génération des commentaires.

I.0.1 Prédiction par bigrammes et unigrammes

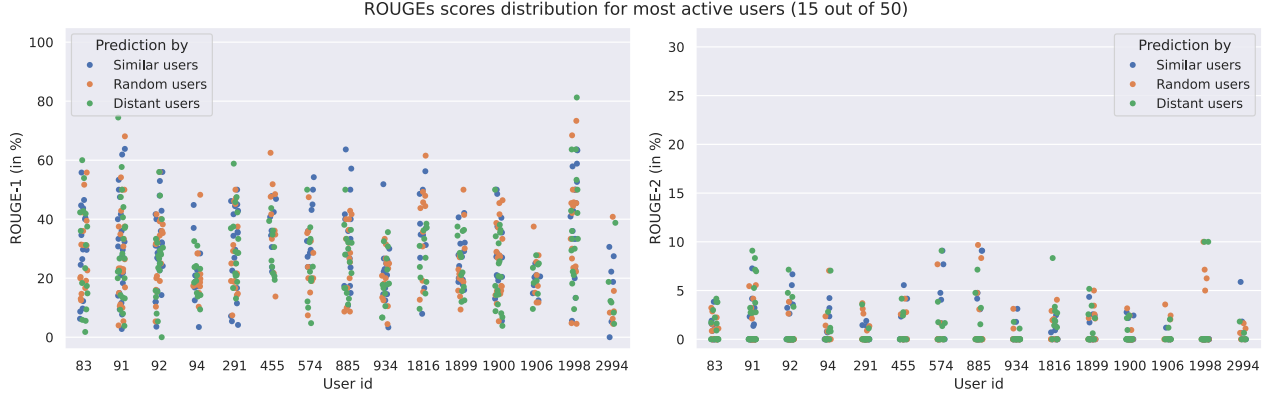


FIGURE 26 – Distribution des scores par jeu pour 15 utilisateurs les plus actifs avec filtrage TopX et TF-IDF

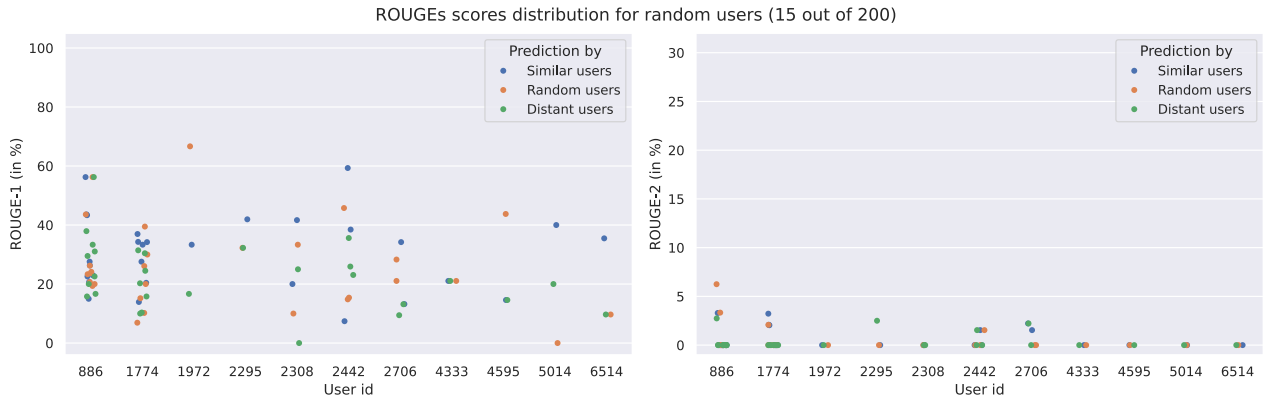


FIGURE 27 – Distribution des scores par jeu pour 15 utilisateurs aléatoires avec filtrage TopX et TF-IDF

I.0.2 Prédiction par phrases

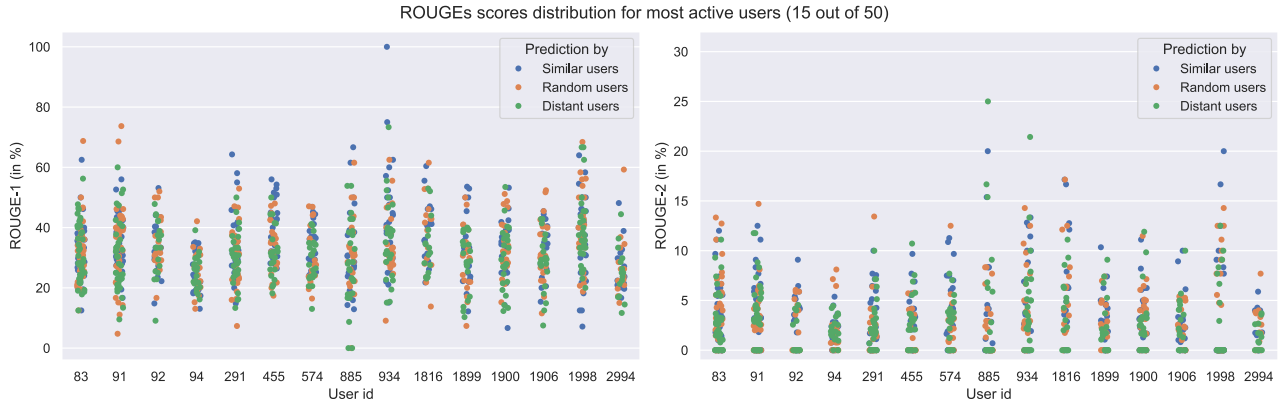


FIGURE 28 – Distribution des scores par jeu pour 15 utilisateurs les plus actifs

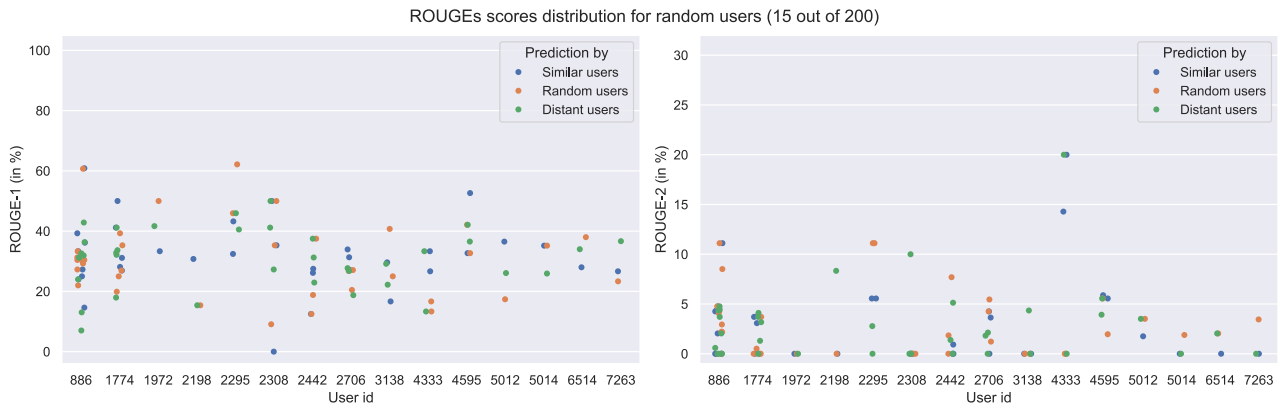


FIGURE 29 – Distribution des scores par jeu pour 15 utilisateurs aléatoires