Videogames recommendations and gamers segmentation

SIADS 696 Milestone II Project Report

Team Members: Riccardo Ricci (rikricci@umich.edu), Royce Lam (roycelam@umich.edu), Candace Lei (candacl@umich.edu)

1. Introduction

There are several games to play but how to choose which to play. Recommendation systems come to the rescue helping gamers to know which games to play next and maximizing their satisfaction. Our team enjoys playing video games and want to help other gamers finding both popular games but also niche and hidden games using recommender systems.
Another interesting and related topic is allowing similar gamers to be connected based on similar interests as well as helping video gaming companies to provide better customer targeting Thus, the objective of this project is recommending videogames based on games and gamers similarity (supervise and segmenting gamers into groups of similar preferences

Regarding **supervised learning**, we adopted different techniques with the aim of testing these methods as well as finding the best method for games recommendation:

- Popularity-based model
- Content-based filtering
- Collaborative-based filtering
  - kNN
  - Matrix Factorization
  - Deep learning (Neural Collaborative Filtering)

Regarding **unsupervised learning**, we opt for *Principal Component Analysis* and *Cluster Analysis* which is a consolidated approach used in users' segmentation

In supervised learning, we acknowledged the supremacy of Deep Learning. Yet, as we will later, the application of each method depends on the context. When precision matters, Deep Learning is clearly the way forward.

2. Related Work

He et al. (2017) seminal paper on the application of deep learning provided some indication on how to implement deep learning on recommender systems. They also provide a comparative view of different machine learning approaches namely deep learning, matrix factorization and k-Nearest Neighbours (kNN)
Moreira (2020) provides a comparison of content-based and collaborative-based filtering while also suggesting an hybrid approach. This approach could be a future improvement of our work. From the two sources above we adopt the evaluation approaches. We used the *Hit Gain Ratio (HGR)* to evaluate how well data generalize on new unseen data.

3. Data Source

The datasets are retrieved from Kaggle, Kozyriev (2023). It consists of over 41 million cleaned and preprocessed data about users, games and recommendations scraped from the popular game store Steam (https://store.steampowered.com/). The datasets, available as *CSV* files, are the following (underlined are joinable fields):

- The *user dataset* contains anonymised user_id, number of purchased games and reviews by users.
- The *game dataset* contains game_id, title, release date, supported platforms (e.g. Windows, Mac), rating and price. There is an additional dataset with *games metadata* containing description and tags (e.g. Action, Adventure).
- The *recommendation dataset* contains game_id, user_id, review_id, recommended (true or false), number of hours played, review date, the number of users who found the review helpful and funny.

We used the same **datasets** for both supervised and unsupervised learning.

Filtering the Recommendation Dataset

Central to our collaborative-based filtering analysis is the recommendation dataset. This dataset in its entirety, filtering only for positive recommendations, contains *12,639,209 users* and *37,419 games*.
The recommendations range from October 2010 to December 2022.

When performing the user-game matrix but also in machine learning stages, we faced several memory issues. Thus, we decided to filter out the dataset to a subset of users:

- Keeping users that made at least 20 recommendations, helping also on the *cold-start* problem, an known issue usually solved with other techniques (e.g. content-based filtering);
- Randomly selecting 10000 users (note that in some specific cases we had to further reduce this number to allow for model convergence).

Furthermore, we decided to ***include only users who recommended games and filter out users who disrecommend games***. We did so because we wanted to differentiate between users who recommend a game with those not recommended. Having the same value (i.e. 0) for users that disrecommend and not recommend would have determined specification issues. In short, the coding of recommendations is the following

- 1, if a user *i* recommend a product *j*
- 0, if a user *i* did not recommend a product *j*

Indeed, we had to add some variations in our dataset especially in Deep Learning. Following the approach of He et., al (2017) we randomly add some items with 0s in the training set.

4. Feature Engineering

User-Game Matrix (for collaborative filtering)

We spent a great part of our project schedule building the User-Game matrix. **This is central for collaborative-based filtering.** It is a *sparse matrix* having a huge amount of null values compared to the tiny fraction of non-null values i.e. the recommendations of users *i* (with *i=1,...n*) to game *j* (with *j=1, …m*). In practice, users recommend only a tiny subset of games compared to the list of available games. The sparsity for the entire recommendation dataset with no filters applied amount to 99.9899% (density = 0.0101%). Filtering for users who made at least 20 recommendations, the sparsity reduces to 99.9652% (density = 0.0348%).
We adopted the python features that handle sparsity of matrices using the *scipy* library. We used the *Compressed Sparse Row Matrix (CSR)* and the *Dictionary of Keys (DOK) format.* The former consists of
We used this format because it provides efficient arithmetic operations, efficient row slicing and fast matrix vector products (Scipy). We preferred this over *Compressed Sparse Column Matrix (CSC)* because we have far more rows (users) than games (columns). The DOK format consists of a dictionary that maps (row, column)-pairs to the value of the elements. We used this given its simplicity in accessing the user-game pairs. This was particularly handy to build the training and test sets (see later). We were able to consistently switch between the formats when needed.

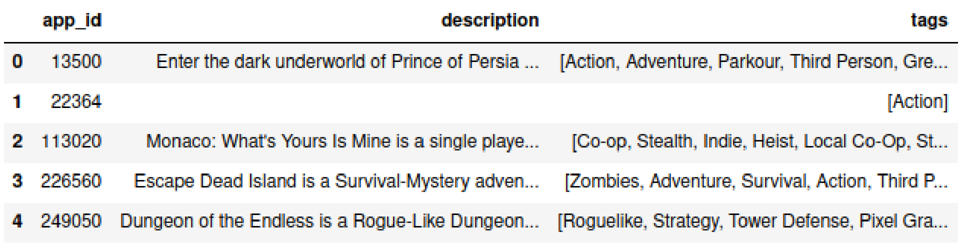
The creation of the user-game matrix (*Help* folder) took several steps to tackle memory issues:

1. Sliced the recommendation datasets by group of users to allow the following step without incurring memory issues;
2. For each portion we applied a pivot table using the *Pandas* library. Rows and columns of the pivot table are users and games respectively, while the values are 1 i
3. We assured that every portion of the dataset has all games to make possible the following horizontal concatenation;
4. To save memory, each pivot table was converted to a sparse CSC matrix;
5. Finally, we concat each sparse CSC matrix to one unique matrix and save it to a pickle file (matrix/user_game_matrix.pkl)

Update (in 3 days of submission): The part above was computational heavy and prone to error. We discovered a bug in 3 days of project submission which led us to rethink how to speed up the new building of the user-game matrix. After learning via trial and error, especially on sparse matrices, and looked more deeply into related work (He. et al, 2017), we discovered that there was a much easier way to build the user-game matrix i.e. using the dictionary of keys format provided by *scipy*. (see the Discussion section).

Games-Tags Matrix (for content-based filtering and unsupervised learning)

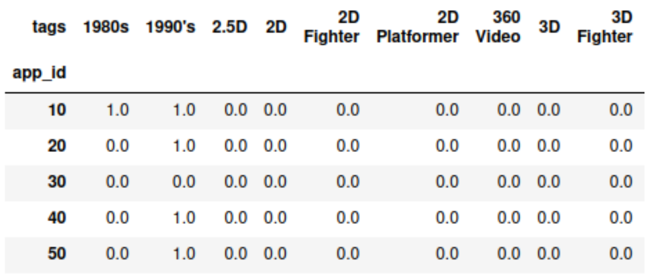
The game's metadata dataset contains the tags of games in a list of values. See an example below.



	app_id	description	tags
0	13500	Enter the dark underworld of Prince of Persia ...	[Action, Adventure, Parkour, Third Person, Gre...
1	22364		[Action]
2	113020	Monaco: What's Yours Is Mine is a single playe...	[Co-op, Stealth, Indie, Heist, Local Co-Op, St...
3	226560	Escape Dead Island is a Survival-Mystery adven...	[Zombies, Adventure, Survival, Action, Third P...
4	249050	Dungeon of the Endless is a Rogue-Like Dungeon...	[Roguelike, Strategy, Tower Defense, Pixel Gra...

Figure 1. Games' Tags

We first build a *vocabulary of tags*. To explore a bit the data, here are the top 5 tags with their respective count of games *('Indie', 27957), ('Singleplayer', 22566), ('Action', 21897), ('Adventure', 20183), ('Casual', 17844)*. Then, we build a *game-tag matrix* needed for **content-based filtering**.



tags	1980s	1990's	2.5D	2D	2D Fighter	2D Platformer	360 Video	3D	3D Fighter
app_id									
10	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 2. Games-Tags Matrix

4. Supervised Learning

4.1 Model Evaluation

Our model evaluation was inspired by the work of He el al., (2017). We used the Hit Gain Ratio (HGR) adopting the *Leave One Out* Approach i.e. taking out, for each user, the user-game couple corresponding to the maximum recommendation date. For each user, we sample 100 games that a user has never interacted with (Moreira, 2020). To this, we add the game in the test set. Finally, we ask the model to rank the 101 games checking if a game is among the top K recommendations.

To evaluate the *training set* we used the well-known metrics in recommendation systems (Malaeb, 2017):

- *Precision@k = (# of recommended items @k that are relevant) / (# of recommended items @k)*
- *Recall@k = (# of recommended items @k that are relevant) / (total # of relevant items)*

In other words, precision at k is the proportion of recommended items in the top-k set that are relevant, while recall at k is the proportion of relevant items found in the top-k recommendations (Malaeb, 2017).

Recommended items @k that are relevant is the intersection of two sets: the predicted games by the model and the games currently recommended. In the training set, the games currently recommended are all except the last one (by maximum recommendation date) excluded with the Leave One Out Approach.

Hit Gain Ratio can be seen as a simpler version of recall where we take only one element.
Future improvement could integrate the Net Cumulative Gain (NCG) metric.

This setting was used to tune hyper-parameters in the following machine learning models.

4.2 Popularity-based Model

A simple approach, sometimes difficult to beat, is a model in which we recommend to all users the most popular games. In some ways, the prediction of games to be played is always the same.
These are the top 10 popular games by the number of positive recommendations received

Game	# reviews
The Witcher® 3: Wild Hunt	1221
Portal 2	1187
Cyberpunk 2077	1170
DARK SOULS™ III	1161
Team Fortress 2	1161
Left 4 Dead 2	1139
DOOM	1116
Tomb Raider	1097
Fallout 4	1045
Half-Life 2	1037

Figure 1. Top 10 Popular Games

We assigned this list of games to each user computing the HGR iterating over an increasing number of popular games. The results are provided below.

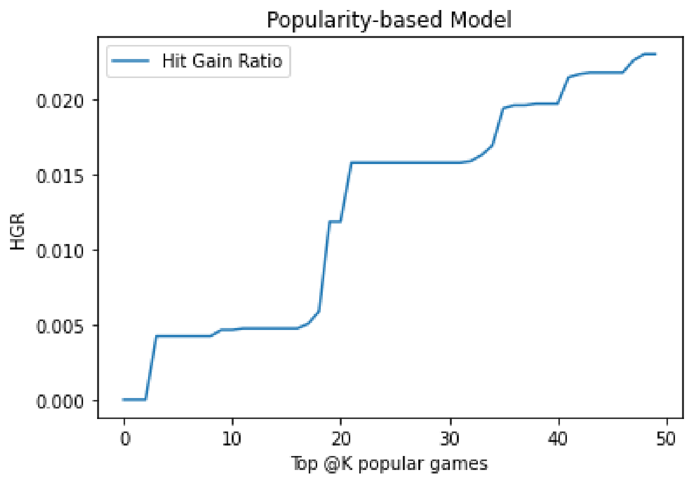


Figure 3. Popularity-based model evaluation

The HGR values are very low. Yet, this is reasonable considering that we are assigning the same (those that are most popular) games to all users. As we can see, there is a large room for improvement.

4.3 Content-based Filtering

We have conducted some further feature engineering on the game tags and recommend similar games to the users according to the game tags.
Below, we describe how we implement this model:

1. We implemented PCA to reduce the number of tags from 441 to 250 tags. The value of *k* is determined by exploring the Cumulative Explained Variance Ratio. (This is a different approach than the PCA conducted for Unsupervised Learning below.)

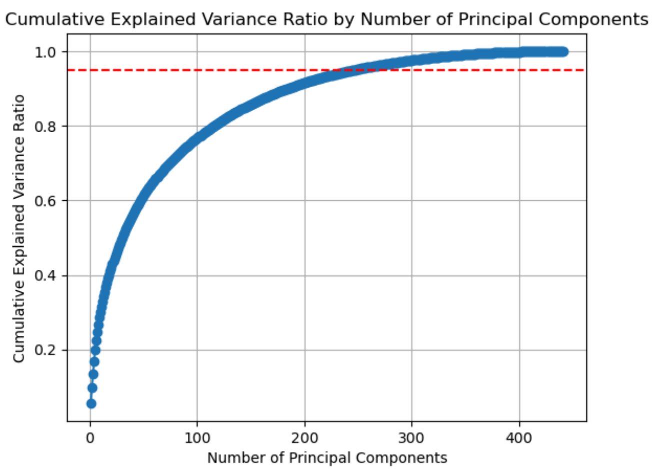


Figure 4. Hyperparameter tuning for PCA (k=250)

2. We used the reduced tags to build games-reduced tags matrix

4.4 Collaborative-Based Filtering

Collaborative-based filtering can be categorized into memory-based and model-based. Below, we present one memory-based approach and two model-based approaches.

4.4.1 Memory-based k-Nearest Neighbors

In collaborative filtering, the user-game matrix represents users' preferences or interactions with different games. Each user is represented as a vector in the matrix, where each element corresponds to the user's rating or interaction with a specific game. By applying a similarity measure such as the cosine similarity between user vectors, we can measure how similar their preferences are based on their past behaviors.

We choose *cosine similarity* compared to *Euclidean distance* because cosine similarity is more suitable for measuring the similarity between high-dimensional vectors, such as user profiles in collaborative filtering (see the Appendix for more details on cosine similarity)

After applying cosine similarity, we applied the k-NN algorithm. The main idea behind this model in collaborative filtering is to find the K most similar users to a target user based on their past behaviors and then recommend items that similar users have also liked or interacted with (Nguyen et al., 2023).

The advantage of cosine similarity and K-NN is its *simplicity*. The disadvantage is *low scalability* as the number of users increases. In fact, computing the pairwise correlation becomes expensive and slow. Indeed, using different training sizes, we acknowledged that time to train the model increased exponentially.

Below, we describe how we implemented this model:

1. We removed from the user-game matrix the user-game items corresponding to the test set;
2. We applied the cosine similarity to the user-game matrix which resulted in a square matrix of users similarity;
3. For each user and different levels of K, we found the K-neighbors users as well as their distances;
4. Finally, we evaluate the model for each level of K (K ranges from 1 to 5).

We provide below the results of this model at different levels of K neighbors.

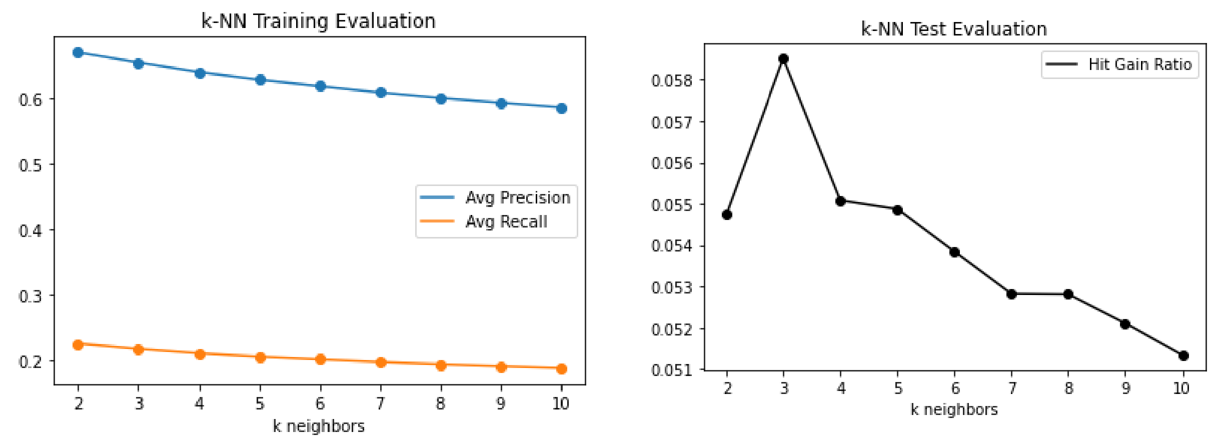


Figure 5. Model Evaluation (Top @10k) - k-NN

We see that precision and recall in the training set as well as the HGR in the test set decreases as we increase the number of neighbors. This may suggest that the model has increased difficulty to converge as we increase the number of neighbors. The best number of neighbors lies between 2 and 3.
Overall, we see that these results, though not very high, provide a first improving model compared to the popularity-based model.
To summarize, using cosine similarity in memory-based k-Nearest Neighbors provides a foundation for generating personalized recommendations. However, we will see what other models can provide. However, this model can be good to apply in low-dimensional context and for rapid exploration.

4.4.2 Model-based Matrix Factorization

To address the scalability limitations of memory-based approaches, model-based approaches can be used. One of these approaches is the **Matrix Factorization** which is a latent-based model that learns the embeddings from the simple representation if a user recommends a game. In short, this model fills all the empty cells of the user-game matrix.

A critical hyper-parameter is the *k* number of factors or features to be used in the SVD. The more (less) the number of factors the more (less) precise is the factorization in the original matrix reconstruction. The key here is finding a number of factors that perform well in the training data but also it is able to generalize on new unseen data. It's important to note that like K-means clusters, the k features are not inherently interpretable.

Below, we describe how we implemented this model:

1. Similarly to the k-NN, we removed from the user-game matrix the user-game items corresponding to the test set;
2. We used the Truncated SVD from the *sklearn* library;
3. We check for each user the value of the Root Mean Squared Error (RMSE). This is to evaluate the model on the training set,
4. For each user, we sort items in descending order keeping the top K games. We calculate the Hit Gain Ratio to evaluate the model on the test set
5. We iterate over different number of factors to optimize RMSE and HGR

The results will be available on request. Due to time constraints, the model evaluation has not been performed.

4.4.3 Model-based Deep Learning

Applying Deep learning in recommender systems, also called Neural Collaborative Filtering (NCF) (He et al., 2017), can be seen as a generalization of matrix factorization. Using the *Keras* package, we built a Multi-Layer Perceptron (MLP) with two embedding layers, one for users and one for games, 2 hidden layers and one final output layer. The framework is similar to the work of He et al., 2017. To find the best model and to assess the sensitivity, we explore different numbers of neurons in the two fully connected layers.

Below, we can see how the convergence of the model is reached completely in the most complex model with 80 and 40 neurons in the first and second hidden layers respectively.

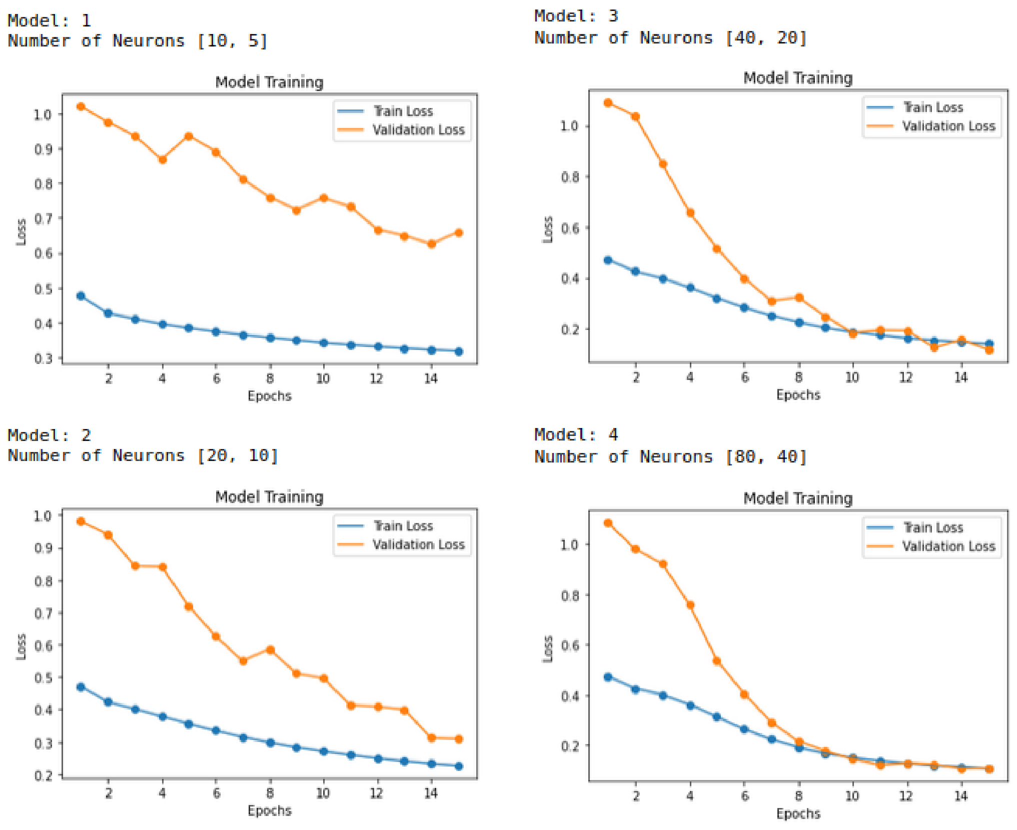


Figure 6. Training and Validation Loss at different neurons

We found that the choice of the *number of neurons* and the *learning rate* were critical for model convergence (i.e. training and validation loss). We first iterate over different numbers of neurons and then with the best available we analyze sensitivity with respect to the learning rate (results available on request).

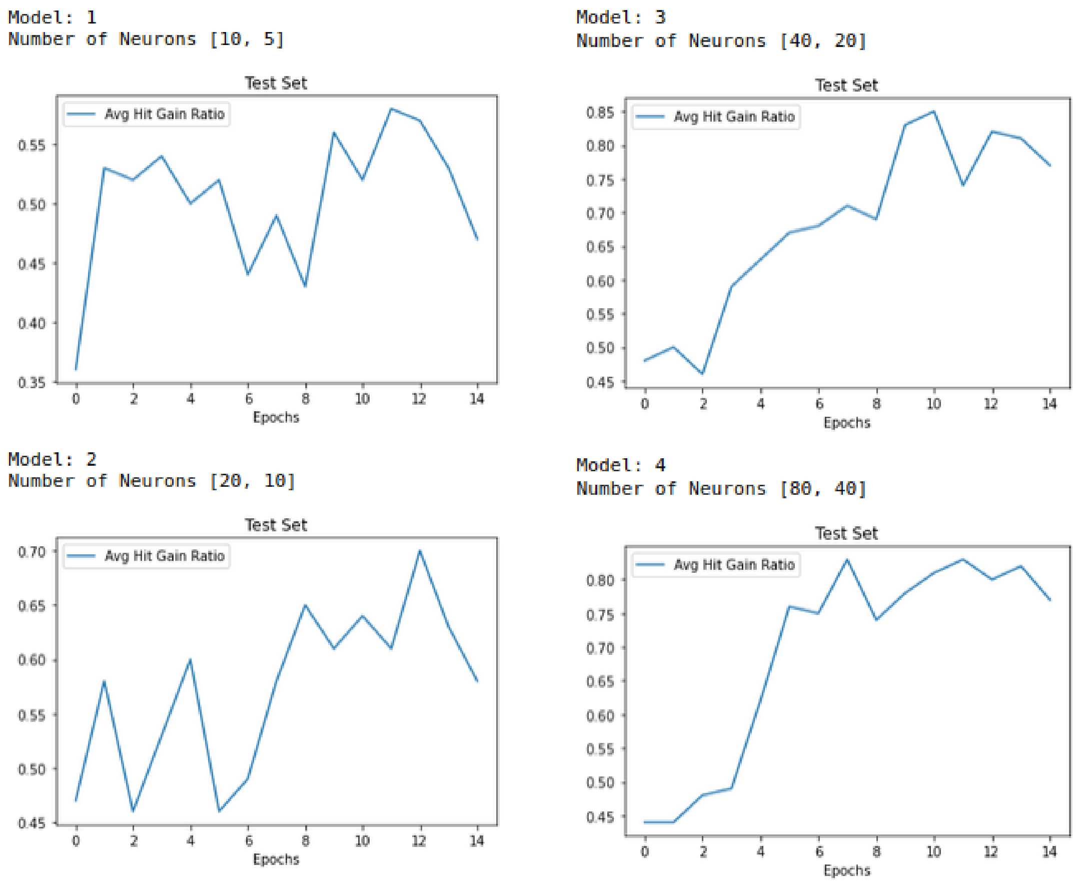


Figure 7. Hit Gain Ratio (Top @10k) - Deep Learning

Figure 6 shows that model 3 and 4 have the best values of HGR on 15 epochs. Yet, the last model given the convergence of loss (Figure 5).

5. Unsupervised Learning

In the unsupervised learning approaches, our goal is to identify patterns and segments within the gamer population based on their gaming preferences. We want to understand the underlying structure of the dataset and group gamers with similar preferences together. This can help in targeted marketing, game design improvements, or community management strategies.

We used the game metadata dataset containing tags about games (e.g. Adventure, RPG). Below, we describe how we implemented this model:

1. We apply the *Principal Component Analysis* (PCA) to the games tags to find a smaller set of games tags. In fact, the number of tags available (more than 4000) would make it impossible to use in the following cluster analysis as they become unmanageable for humans to define cluster assignment.
2. We then named each component based on the top 10 tags sorted by PCA loading (see PCA Results). Defining the name of a component is a manual process and requires several iterations on the number of components to retain in PCA. Opting for a small

number can cause difficulties naming the components due to heterogeneous tags. With a large number of components, the manual activity increases. Thus, we decided for a trade-off opting for **6 components**: *"Adventure", "War Muliplayer", "Management", "Strategy", "Simulation", "RPG","Romantic", "Driving", "Realistic", "Horror".* The top 10 tags by loading for each component is provided in the appendix.

3. Each tag was mapped against the component according to the maximum component loading.
4. Finally, we apply the hierarchical cluster analysis to group users with respect to these components

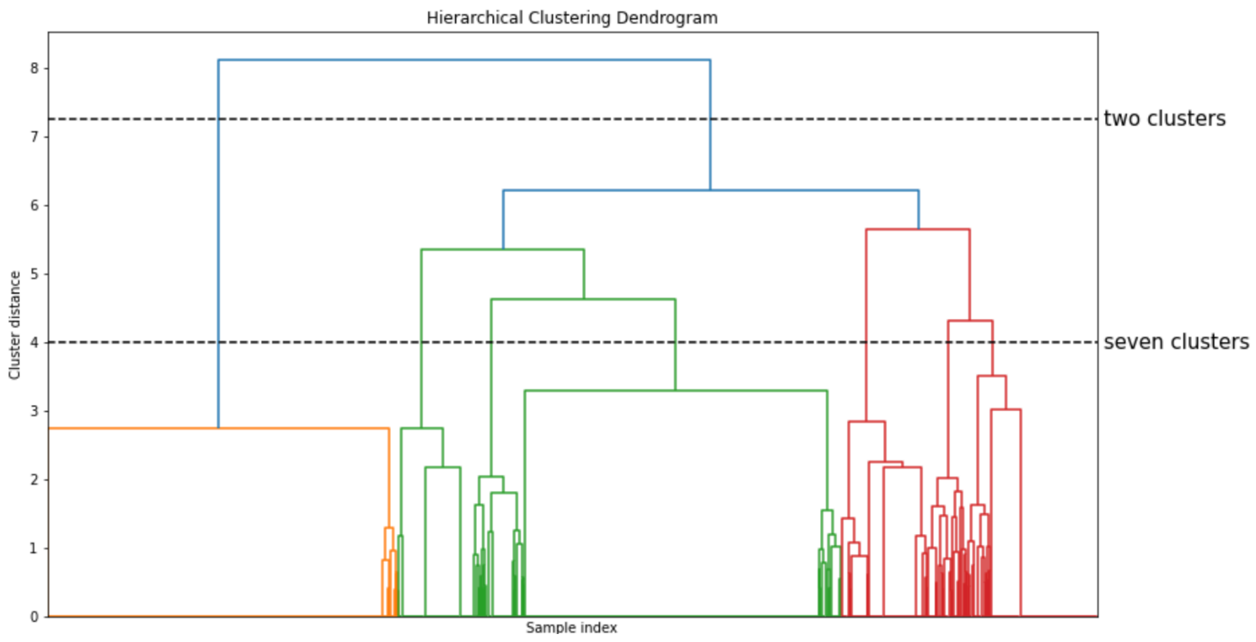


Figure 8. Dendrogram (Hierarchical Clustering)

In Hierarchical Clustering, the **evaluation** is usually performed looking at the dendrogram which shows the distance of each user with respect to cluster assignment. The dendrogram above shows that a good solution is having 7 clusters. If we were to choose 2 clusters we would get heterogeneous users inside the cluster. In fact, the cluster distance is very high. If we were to choose 8 or more clusters then it would become difficult to understand and manage for possible market targeting.

Cluster Labeling. Using the *fcluster* method of the *scipy* library, we assign to each user a cluster. The heatmap below shows the number of occurrences of the reduced number of tags (i.e. components) for each cluster.
Cluster 1 - *Play All*; Cluster 2. *All except Horror and Realism*; Cluster 3. *All except Management*, Cluster 4. *All except Horror*; Cluster 5. *Adventurous and Realism*, Cluster 6. *All*; Cluster 7. Strategy, *Adventurous and Realism*.

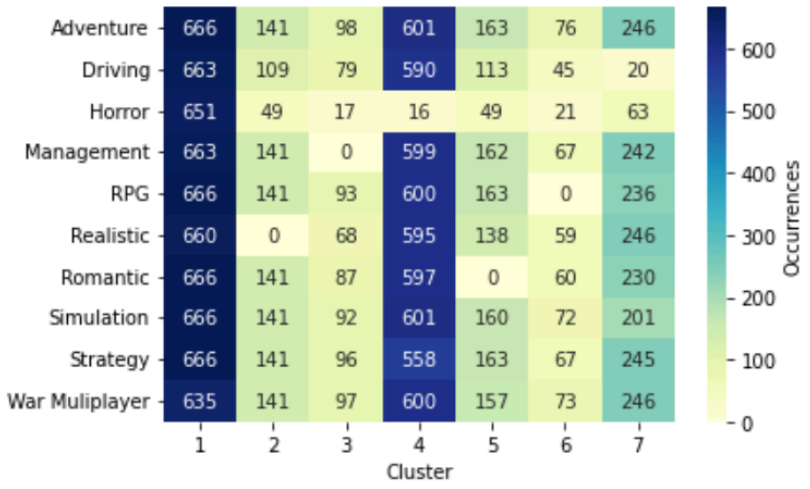


Figure 9. Tags-Cluster Heatmap

6. Discussion

In this project, we experimented with different machine learning approaches, both supervised and unsupervised.

Supervised Learning. Our results show that the Deep Learning model is the best model in terms of metrics evaluation. This result provides support to the current literature (He et al., 2017). We discovered that the model is sensitive to the number of neurons. Critically, we discovered that the optimal number depends on the data volume. Yet, this model can be particularly complex and heavy to train. In some situations, where easiness to explain is key especially with practitioners and low data volume, approaches like k-NN and Matrix Factorization provided valuable alternatives.

Although not shown above, we also checked that there is a ***trade-off*** *between the amount of data used in training and loss convergence*. Increasing the size of the training data determined an increase in the amount of time to reach convergence, and in specific cases convergence never occurs. We opted for a balance between time and data needed for convergence and amount of training data for better generalization on new unseen data.

Unsupervised learning. We recognized that it is heavily dependent on humans to correctly interpret components and clusters. As a result, it can be less *robust* compared to supervised learning. The best approach is to integrate these models e.g. using PCA to find a set of manageable game tags and then applying a Neural Network along with. PCA helps reduce the *curse of dimensionality* providing a great support to supervised learning techniques. We aim to experiment with this hybrid approach in the future.

On our hierarchical clustering, surprisingly we acknowledged that it is highly dependent on the number of users. We tried with a higher number of users (greater than 10000 users) but the kernel interrupted most of the times. The clustering assignment can be further improved by applying iterative approaches like k-means clustering. Knowing the number of clusters and the clustering centers from the hierarchical clustering, we could evaluate a better k-means clustering. In this respect, we could evaluate using metrics like *silhouette score* or *within-cluster sum of squares*.

Challenges, Lesson Learned and Future Work

Data size. We faced several challenges related to the size of our data which slowed our analysis. We decided only at later stages to filter out data.
Lesson learned: *starting with small data and increasing later when the initial exploration is done and a model is stable*. In particular, the scalability of memory-based k-NN in collaborative filtering is often limited by the size of the data.
As the number of users and items in the dataset increases, the computational resources required to calculate and store the similarity matrix also increase significantly. Additionally, calculating the cosine similarity between user vectors has been computationally expensive. As the number of users increases, the time required to find the K nearest neighbors for each user also grows, making the algorithm less efficient. To address these limitations future work could investigate *nearest neighbor search algorithms*, such as *locality-sensitive hashing* (LSH) or *tree-based* methods like KD-trees or ball trees. These techniques allow for efficient retrieval of nearest neighbors without explicitly calculating the similarity between all pairs of users. Furthermore, *distributed computing frameworks*, such as Apache Spark or Hadoop, can be employed to parallelize the computation and handle larger datasets. By distributing the workload across multiple machines, it becomes feasible to process and analyze big data in collaborative filtering.

Code Efficiency. We spent several days building the user-game matrix while recognizing at later stages that there were some data inconsistencies. Forced to build the user-game matrix in a short time, we had to find easy and rapid solutions. We looked at similar work (He et al., 2017) and documentation on sparse matrices and we found a much easier way.
Lesson learned**:** when the code becomes too complicated it is very often not the right path and very likely to lead to error. In other words, in such situations, there is often another more simple way to do it. Look at previous work and study the documentation.

Data testing*.* Another challenge was related to data inconsistency which wrongly led us to believe that the model was not tuned perfectly. We spent several hours tuning the hyper parameter while recognizing, only at later stages, there were some inconsistencies in the data that affected the model training.
Lesson learned: It is really important to test the data before initiating the model training and hyperparameter tuning

Limited Number of Features and Hybrid Models. Constrained to a user-game matrix composed of 1s and 0s, it was not easy to integrate other query features such as game tags. In fact, the models proposed in this paper do not simultaneously consider user-game interaction and users or game features. Adding side features is not easy but the deep learning framework offers interesting insights in this respect (He et al., 2017). Future work could add, for instance, games tags along with the games embedding matrix (see the Appendix). Summarizing, incorporating other types of information, such as item features or contextual information, into the recommendation process can further enhance the accuracy and relevance of recommendations. Future work could also experiment with *hybrid approaches* that combine memory-based methods with model-based or deep learning techniques (Moreira, 2020; He et al. 2027).

7. Ethical Considerations

Model Bias due to the minimum K recommendations per game (k=20)
The concern with filtering out games with less than k=20 recommendations, critical for model evaluation, is that it may introduce bias in the training dataset used for the model. This approach is commonly used to prevent the *cold-start problem* but it can introduce harm and bias.
By excluding games with recommendations below this threshold, the training data may not accurately represent the diverse interests and preferences of users. This can lead to a biased model that primarily recommends popular or mainstream games, while neglecting niche or less well-known titles. This can affect poor people that may not have enough money to buy costly mainstream games

For example, suppose a user has a particular interest in indie games or niche genres. If the training data primarily consists of costly mainstream titles, the resulting model may not effectively recommend lesser-known games that the user might enjoy. As a result, the system may inadvertently favor mainstream choices and overlook the long-tail interests of users.
To address this concern, it is essential to carefully select the threshold value for K (the number of recommendations) but also to include other side features (e.g. games tags) and other diversity-aware techniques allowing to safely decrease K can provide a greater representativeness of all games and therefore of all users' preferences even of minorities.
Summarizing, fine-tuning the minimum number of recommendations per game, and the incorporation of diversity-aware techniques can help address biases, promote fairness, and provide users with a personalized and enriching gaming experience.

8. Statement of Work

Riccardo Ricci led the initial and final drafting of this project, data loading, the building of the user-game matrix, models evaluation and the Neural Collaborative FIltering. He also led the PCA part of the unsupervised learning.
Royce Lam led the k-NN model, Matrix Factorization and hierarchical clustering. He also provided support drafting this document.
Candace Lei led the development of the popularity-based model, the PCA for content-based filtering and provided support drafting this document. She also explored how to access the *Great Lakes Computing* infrastructure provided by the University Michigan, which has been largely used in this project especially to train the Neural Network.

References

Kozyriev, A. (2023). Game Recommendations on Steam [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DS/2871694

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017, April). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173-182)

Malaeb, M., (2017) https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

Moreira, G. (2020) https://www.kaggle.com/code/gspmoreira/recommender-systems-in-python-101#Collaborative-Filtering-model

Maklin, C. (2022) https://medium.com/@corymaklin/model-based-collaborative-filtering-svd-19859c764cee

Nguyen, L. V., Vo, Q. T., & Nguyen, T. H. (2023). Adaptive KNN-Based Extended Collaborative Filtering Recommendation Services. *Big Data and Cognitive Computing*, *7*(2), 106.

Appendix

Cosine Similarity

Cosine similarity measures the cosine of the angle between two vectors and is based on the dot product of the vectors. It ignores the magnitude of the vectors and focuses on the direction or orientation of the vectors in the vector space. This makes it well-suited for comparing user profiles, where the magnitude of the vectors (i.e., the ratings or interactions) may vary significantly across different users.

Matrix Factorization

Among the Matrix Factorization approaches, we choose the the *Singular Value Decomposition* (SVD) because

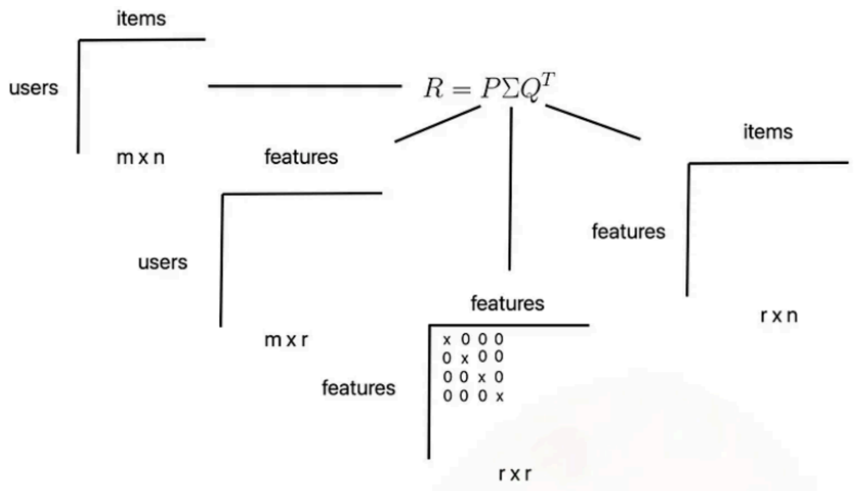


Figure X. Singular Value Decomposition (Maklin, 2022)

Neural Collaborative Filtering

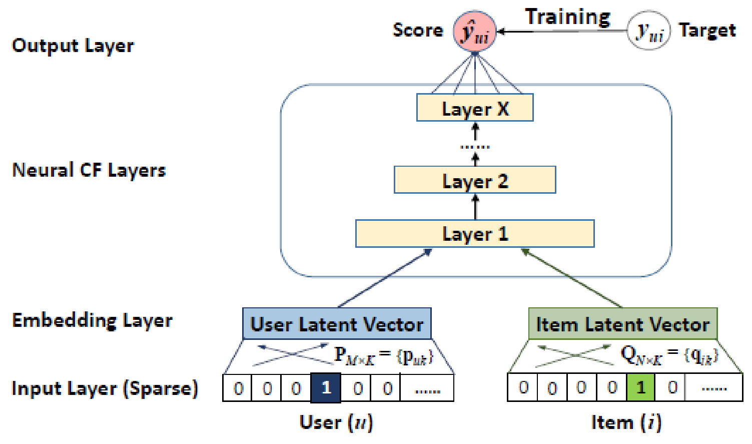
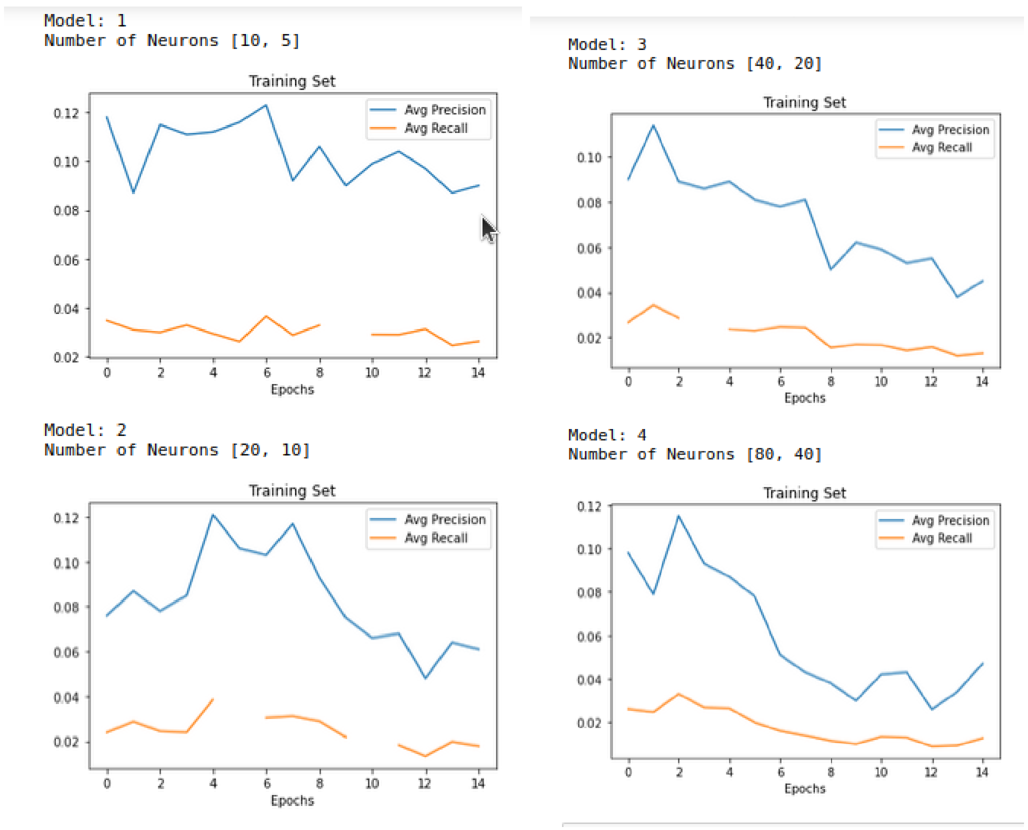


Figure A2. Neural Collaborative Filtering Framework (He et al., 2017)



```
Model: 1
Number of Neurons [10, 5]
```

```
Model: 3
Number of Neurons [40, 20]
```

```
Model: 2
Number of Neurons [20, 10]
```

```
Model: 4
Number of Neurons [80, 40]
```

PCA on Games Tags

Component: Adventure
Top Game Tags in this component:

Singleplayer	157.4
Action	157.2

```
Adventure         156.9
Atmospheric       155.5
Multiplayer       155.4
Indie             153.9
Great Soundtrack  152.2
Story Rich        150.0
Open World        149.5
Co-op             148.0
Name: Adventure, dtype: float64

Component: War Muliplayer
Top Game Tags in this component:
Military              54.0
Massively Multiplayer 52.4
Team-Based            51.5
War                   46.9
PvP                   45.1
Moddable              43.9
Tactical              43.5
Realistic             40.4
RTS                   39.8
Base Building         37.4
Name: War Muliplayer, dtype: float64

Component: Management
Top Game Tags in this component:
Roguelike             55.4
Roguelite             53.5
Procedural Generation 48.0
Turn-Based Strategy   47.6
Management            46.3
Turn-Based            46.1
Resource Management   45.6
Building              44.8
Turn-Based Tactics    44.7
Dungeon Crawler       43.3
Name: Management, dtype: float64

Component: Strategy
Top Game Tags in this component:
Dark Fantasy    43.7
Historical      36.8
Turn-Based      36.4
Medieval        34.6
```

```
Action RPG            34.4
Magic                 33.6
Turn-Based Combat     32.9
RTS                   31.7
Real-Time with Pause  30.8
Turn-Based Strategy   30.7
Name: Strategy, dtype: float64

Component: Simulation
Top Game Tags in this component:
Old School           34.1
Top-Down Shooter     33.8
Aliens               32.4
Fast-Paced           31.3
Local Co-Op          30.2
Beat 'em up          29.0
Local Multiplayer    28.9
Fighting             28.5
Twin Stick Shooter   28.3
Top-Down             27.0
Name: Simulation, dtype: float64

Component: RPG
Top Game Tags in this component:
JRPG             42.1
Anime            39.7
Hack and Slash   35.3
Beat 'em up      31.2
Action RPG       30.3
Fighting         30.2
Sexual Content   29.3
Souls-like       25.7
Dating Sim       25.0
2D Fighter       22.2
Name: RPG, dtype: float64

Component: Romantic
Top Game Tags in this component:
Competitive      36.3
Fighting         34.4
Dating Sim       34.3
Sexual Content   30.6
Visual Novel     29.9
Beat 'em up      28.9
```

```
Military        27.3
War             27.2
Mature           26.0
Anime            25.9
Name: Romantic, dtype: float64

Component: Driving
Top Game Tags in this component:
1990's              31.7
Military            25.1
Visual Novel        23.2
Top-Down Shooter    23.1
Retro               22.1
Dating Sim          21.8
1980s               21.6
Silent Protagonist  21.5
Team-Based          20.0
Aliens              19.3
Name: Driving, dtype: float64

Component: Realistic
Top Game Tags in this component:
Combat                 45.0
PvE                    38.1
3D                     34.0
Realistic              30.6
Early Access           30.3
Crafting               27.2
Stylized               22.2
Immersive Sim          22.2
Swordplay              21.3
Massively Multiplayer  20.6
Name: Realistic, dtype: float64

Component: Horror
Top Game Tags in this component:
Dating Sim           27.9
Sexual Content       26.5
Crafting             25.9
Building             24.4
Hentai               23.5
Resource Management  22.5
Mature               21.8
Management           21.8
```

```
Anime          21.1
JRPG           19.5
Name: Horror, dtype: float64
```