Erwin Byll, Agathe Cauhapé, Walid Faouzi,
Arthur Heslault, Dilara Toygar

Abstract

In the highly competitive video game industry, understanding player preferences is essential for creating engaging and successful games. This project proposes a review-driven recommender system designed to support game developers and product teams by highlighting what players value most. Using a dataset of 47,774 video game entries enriched with structured attributes, such as genre, platform, user ratings, and game quality indicators as well as free-text user reviews, we build a simple content-based recommender system as our minimum viable product (MVP). This system suggests similar games based on shared characteristics, offering users personalized recommendations grounded in concrete features. In parallel, we explore the potential of sentiment analysis on user review text to extract emotional tone and deeper feedback, with the goal of integrating this dimension in future iterations. By combining structured metadata with user opinions, our system transforms raw review data into actionable insights, helping teams make better-informed decisions and improving the overall player experience.

1. Environment

1.1 Dataset

To develop the recommender system, we utilized a comprehensive dataset titled "Video Games Reviews", publicly available on Kaggle. This dataset contains 47,774 real user reviews collected from multiple gaming platforms, making it rich and diverse in terms of user opinions and game metadata.

Each entry in the dataset provides detailed information on various attributes of the video games, including:

- Game Title: The name of the video game.
- User Rating: The score given by users.
- Age Group Targeted: The intended audience age category for the game.
- Price: The retail price of the game.
- Platform: The gaming system on which the game is available (e.g., PC, PlayStation, Xbox).
- Developer and Publisher: The companies responsible for creating and distributing the game.
- Release Year: The year the game was launched.
- Genre: The game's genre classification.
- Multiplayer Support: Whether the game supports multiplayer modes.

The dataset also includes qualitative aspects that provide deeper insights into the gaming experience:

- Graphics Quality
- Soundtrack Quality
- Story Quality
- Game Mode

Moreover, it contains user-generated review text, which can be leveraged for natural language processing, as well as practical details such as:

- Minimum number of players required
- Whether the game requires special devices
- Estimated game length in hours

This rich set of features enables building a robust content-based recommendation system, capable of understanding user preferences and game characteristics on multiple levels.

1.2 Set-Up

The project was initialized by downloading the dataset from Kaggle and importing it into a Python environment using pandas. The raw data was stored in CSV format and loaded for preprocessing and analysis.

The data loading and initial exploration steps were documented in the project's README file and Jupyter notebooks to ensure reproducibility and transparency.

2. Exploratory Data Analysis

2.1 Data Quality

Before jumping into building the recommender system, we spent some time exploring the dataset to understand what it looks like and check its quality.

There were no missing values, which is great because it means the data is complete. No duplicates either, so every review and game is unique. We also checked for any outliers (weird or extreme values), but didn't find anything suspicious.

2.2 Data Analysis

Looking at the data, we analyzed key variables like user ratings, prices, release years, and playtime. We also examined relationships between some features, such as price and rating. However, no obvious patterns or strong correlations stood out.

Some interesting observations emerged. The average user rating is around 29.7 out of 50, showing a moderate level of satisfaction. Games cost about $40 on average, indicating that gaming can be expensive, though prices vary widely without clear links to quality. Most games were released between 2010 and 2023, with an average around 2016. Playtime ranges a lot, from a few hours to over 60 hours. Features like multiplayer support and quality ratings (graphics, story) showed no consistent trends.

Overall, this tells us that the player experience can't be fully explained by simple stats. To build effective recommendations, we need to consider both the quantitative data and the subjective aspects of gaming.

3. Content-Based Recommender System

3.1 Methodology

For this project, we implemented a content-based filtering recommender system. Unlike collaborative filtering, which relies on user behavior and interactions, content-based filtering recommends items based on the attributes of the items themselves. This approach is particularly useful when we have detailed metadata about each item in our case, video games.

The main idea is to suggest games that are similar in content to a game the user is interested in.

3.2 Data Preparation & Similarity Computation

Before we could build the recommendation engine, we had to prepare the dataset for efficient searching and comparison:

- Title normalization: We cleaned game titles by converting them to lowercase and trimming spaces. This step ensures that searches for game titles are case-insensitive and less prone to errors.
- Reindexing: We created a reindexed version of the dataset for consistent and straightforward retrieval of games by position.
- Similarity matrix: We computed a similarity matrix representing pairwise similarity scores between all games. These scores are calculated based on selected features such as genre, platform, and other relevant attributes. We used [insert similarity metric here, e.g., cosine similarity] to quantify the closeness between feature vectors of different games.

This matrix is the backbone of the recommendation system since it allows fast lookup of the most similar games for any given title.

3.3 Recommender Function

The core recommendation logic is implemented in the function *recommend_games()*. Here is how it works:

Input cleaning: The function takes the user's input (a game title) and normalizes it by stripping whitespace and converting it to lowercase.

Search: It searches the reindexed dataset for entries that contain the cleaned input title.

Match handling: If no match is found, the function returns an empty result and informs the user.

Similarity retrieval: If a match is found, the function retrieves similarity scores for that game from the similarity matrix.

Sorting and selecting: It sorts these similarity scores in descending order and selects the top-N most similar games, excluding the game itself.

Output: Finally, it returns the recommended games with key information such as their title, genre, and platform.

3.4 Evaluation

To assess how well our content-based recommender performs, we implemented a simple yet meaningful evaluation metric: precision at k, based on genre matching. The idea behind this metric is that games recommended to a user should ideally belong to the same genre as the game they searched for. Although this is a simplistic evaluation, it gives us a baseline indication of recommendation quality.

Applying this evaluation to our recommender system, we found that the precision at 5 is around **60%**. This means that, on average, 3 out of 5 recommended games belong to the same genre

as the user's chosen game. While this result indicates that the system can reasonably recommend genre-relevant games, there is still room for improvement

3.5 Streamlit Application

To make our content-based recommender system accessible and easy to use, we built a web application using Streamlit. Streamlit is a Python framework perfect for showcasing data science projects without needing extensive frontend development skills.

Our goal was to let users type in the name of a game they like (or pick one from a list), choose how many recommendations they want, and get personalized suggestions with lots of useful info about each game.

When the app starts, it loads the cleaned dataset and sets up the recommender system. Since calculating the similarity between games can take some time, we use Streamlit's caching feature to save the results and avoid waiting every time someone interacts with the app.

Users can either type in the name of a game they like or choose one from a dropdown list that contains all the game titles in the dataset. There is also a slider where users can select how many recommendations they want to see, anywhere from 1 to 10.

Once the user clicks the "Get Recommendations" button, the app looks for games that are similar to the one they entered. If it can't find the game in the dataset, it shows a warning message asking the user to try a different title.

If recommendations are found, the app displays them in expandable sections. Each section shows useful details about the game like its genre, platform, developer, publisher, release year, price, and ratings for graphics, soundtrack, and story.

4. Text Analysis

To better understand what players say about the games, we analyzed the review texts using TF-IDF (Term Frequency-Inverse Document Frequency). TF-IDF is a method that helps us find important words by balancing how often a word appears in a review with how common it is across all reviews. This way, common words like "the" or "and" get ignored, and more meaningful words stand out.

After applying TF-IDF to all the user reviews, we identified the top 15 most important words that appear across the dataset. Some of the top words include "game," "amazing," "bugs," "better," "graphics," and "gameplay."

This analysis helps us understand the overall player sentiment and what features stand out in user opinions, which can be useful for improving recommendations and game development alike.

5. Hybrid Recommender System

We think it's important to mention that this part of the project was more challenging and exploratory for us. While the content-based recommender was straightforward and gave decent results, we wanted to push ourselves further by combining different types of data and techniques. This hybrid approach allowed us to experiment with new ideas and apply more sophisticated models, even if it meant dealing with added complexity.

5.1 Methodology

Our goal was to improve recommendation quality by combining different kinds of data. The content-based recommender looked only at structured game attributes, which misses out on valuable user opinions hidden in reviews. By building a hybrid system, we merge structured features like genre, platform, and price with semantic information extracted from user reviews.

To represent user reviews in a way the computer can understand, we use sentence embeddings. Specifically, we applied the pre-trained **all-MiniLM-L6-v2** model from the SentenceTransformers library. This model transforms each review into a dense numerical vector (embedding) that captures the semantic meaning of the text.

This embedding is lightweight and fast but still powerful enough to encode the important context of reviews. Each game review is converted into one embedding vector, summarizing the overall user sentiment and experience.

5.2 Data Preparation

We then prepared the structured game features. For categorical variables like genre and platform, we applied one-hot encoding to turn categories into numbers the system can use. Numerical features such as price, user rating, and release year were scaled using a standard scaler so all features would have similar ranges and importance during similarity calculations.

Once the review embeddings and structured features were ready, we normalized both sets to avoid scale differences. Then, we combined them into one single hybrid feature vector for each game by joining the processed structured data with the text embeddings side by side. This hybrid vector contains both information and the semantic content of reviews.

5.3 Hybrid Recommender Function

Using hybrid feature vectors, we calculate a **cosine similarity matrix**. Cosine similarity measures the angle between two vectors, giving a value between -1 and 1 that indicates how similar two games are based on their combined features.

The recommendation function works in these steps:

Input Cleaning and Search: It takes the user's game title input, strips whitespace, and converts it to lowercase for consistency. Then it searches the dataset for game titles that contain

this input (allowing partial matches). This fuzzy search helps when the user types incomplete or slightly incorrect titles.

Match Handling: If no matching game is found, the function returns an empty result and informs the user.

Similarity Scores Retrieval: Once a match is found, the function uses the index of that game to retrieve its similarity scores with all other games from the cosine similarity matrix.

Sorting Recommendations: The scores are sorted in descending order, so the most similar games come first. The original game itself is not included in the recommendation list.

Return Top Recommendations: The function returns the top-N most similar games, including useful details like title, genre, and platform.

This hybrid recommender combines structured features with textual review embeddings to capture both the facts about a game and the feelings players express about it. The recommendation function efficiently finds similar games by calculating cosine similarity on these hybrid vectors and returning the closest matches. This approach improves the relevance of recommendations compared to using structured data alone, offering users a richer experience.

5.4 Streamlit Application

This app provides game recommendations by combining both structured game information and the content of user reviews. When the app starts, it loads the dataset and the necessary models in the background, including computing the review embeddings and similarity matrix. This initial loading is done only once and cached, so subsequent interactions are faster.

Users can enter a game title manually or select one from a dropdown list containing all available games. They can also choose how many recommendations they want, from 1 to 10.

When the user submits a request, the app searches for the input game and finds the most similar games based on a hybrid similarity score, which includes both metadata and review text information. If the game is not found, the app notifies the user to try another title.

The recommendations are displayed in expandable sections that show detailed information about each suggested game, such as genre, platform, developer, publisher, release year, price, and ratings for graphics, soundtrack, and story quality. If available, a short excerpt from a user review is also presented to provide additional context.

Overall, the app aims to deliver relevant and informative recommendations with a user-friendly interface that allows easy exploration of suggested games.

6. Conclusion

Throughout this project, we explored different ways to build a video game recommender system, starting from basic methods and moving toward more advanced techniques. We began with a content-based approach using structured game features, which gave us a solid foundation and helped us understand the main characteristics of our dataset.

To go further, we experimented with natural language processing by analyzing user reviews through TF-IDF and sentence embeddings. This allowed us to capture deeper information, especially when game descriptions or scores didn't reflect how players felt. From there, we built a hybrid recommender system that combined both structured data and unstructured text to offer more nuanced and personalized suggestions.

We also created two Streamlit apps (one for each recommendation system) to make our work accessible and interactive. This part helped us better understand the importance of usability and performance in real applications.

Even though some parts were technically challenging, especially the hybrid system and embedding computations, this project helped us apply a variety of data science tools in a practical context. We learned to clean and explore a real-world dataset, build and evaluate recommender systems, and deploy our work through an app interface. Looking back, we realize that there's a lot more we could do. With more time, we would have liked to fine-tune the content-based model, especially to better weight the importance of different features. For the hybrid model, we saw that there's room to improve the quality and speed of recommendations, and we were curious about trying more advanced techniques. We also considered exploring deployment tools like Docker.

Another learning experience was using GitHub. Since it was our first time using it, we had to overcome a few technical issues along the way. Still, it taught us good habits for working on group projects and maintaining reproducibility.

In the end, this project was both technically enriching and creatively engaging. We got to work with real data, apply natural language processing, test different recommendation strategies, and build a functional product. It was a valuable introduction to the world of recommender systems and a great challenge overall.