Methods for generating recommendations based on user preferences and video game characteristics

Oleh Veres[*,†], Pavlo Ilchuk[†], Olha Kots[*,†] and Yurii Veres[†]

Lviv Polytechnic National University, Stepana Bandery str. 12, Lviv, 79013, Ukraine

Abstract

The research is devoted to studying the video game recommendation process based on the consideration of user preferences and video game characteristics to improve the efficiency of selection by implementing intelligent recommendation generation algorithms.

While there are many recommendation algorithms and techniques, most of them fall into the following broad categories: collaborative filtering, content filtering, and contextual filtering. Collaborative filtering algorithms recommend items based on information about the preferences of many users, and build a model based on a user's past behavior, such as items previously purchased or ratings given to those items, as well as similar decisions by other users. Content filtering uses attributes or features of an object to recommend other objects that are similar to the user's preferences. It is based on the similarity of the characteristics of the item and the user, taking into account information about the user and the items he or she has interacted with, and models the probability of a new interaction. Hybrid recommender systems combine the advantages of the above types to create a more comprehensive recommender system. Contextual filtering uses the sequence of contextual user actions to predict the likelihood of the next action.

In this article, three different recommender system algorithms, one content-based and two collaborative filtering algorithms (one with the ALS algorithm and the other with the EM and SVD algorithms), were proposed and studied to generate video game selection suggestions. To test the effectiveness of the algorithms, we used datasets of users and video game characteristics available on Kaggle. The recommendation system with the ALS algorithm provides the best recommendations based on the evaluation, so it was implemented in the prototype. The recommender application can significantly improve users' gaming experience, making the process of choosing games more personalized and efficient.

Keywords

collaborative filtering, recommendation system, video game, content filtering, matrix factorization

1. Introduction

The video game industry is currently experiencing rapid development and transformation. Each year, it becomes increasingly dynamic and diverse, providing players with endless options for entertainment and interactive experiences. In this context, the importance of recommender systems for video games has grown significantly in today's information society [1-5].

One of the main challenges facing the video game industry is the large number of games available. According to statistics, today, the number of games on Steam, one of the largest platforms for PC gaming, exceeds 30 thousand. At the same time, the number of games available in other stores and on other platforms is also impressive in its diversity and number. This is compounded by an overwhelming number of new releases, updates, and remasters, which makes the process of choosing a video game extremely difficult for a player. Selecting a game that is interesting to the user can be time-consuming and requires significant effort from players.

ISW-2025: Intelligent Systems Workshop at 9th International Conference on Computational Linguistics and Intelligent Systems (CoLInS-2025), May 15–16, 2025, Kharkiv, Ukraine

[*] Corresponding author.

[†] These authors contributed equally.

oleh.m.veres@lpnu.ua (O. Veres); pavlo.g.ilchuk@lpnu.ua (P. Ilchuk); olha.o.kots@lpnu.ua (O. Kots); yurii.o.veres@lpnu.ua (Yu. Veres)

0000-0001-9149-4752 (O. Veres); 0000-0003-4636-2309 (P. Ilchuk); 0000-0001-7123-3635 (O. Kots); 0009-0007-7750-7159 (Yu. Veres)

Recommender systems are designed to simplify the process of selecting video games by providing personalized recommendations that align with players' interests and preferences. To generate tailored suggestions, these systems analyze player activity data, including game history, interests, genres, ratings, and reviews. By doing so, they not only make it easier for players to discover new games but also enhance their gaming experience by making the selection process more informative and personalized.

The relevance of research in recommender systems for video game selection is particularly evident given the growth of the video game industry and the rapid expansion of its player base. These systems have become essential tools for consumers who utilize video games for entertainment and recreation. Research in this area holds significant practical importance, as it improves the gaming experience for millions of players worldwide and supports the ongoing development of the video game industry.

2. Analysis of the current state and prospects in the field of research

The utilization of recommender systems in the entertainment industry is crucial for several reasons:

- *Content personalization.* Recommender systems allow users to receive individualized content that best suits their interests and preferences. This increases the user's enjoyment of entertainment content as they get access to what interests them the most.
- *Convenience and time-saving.* Thanks to recommendation systems, users do not need to spend time searching for entertainment content from a large number of games, movies, music, etc. The systems automatically suggest options that match their tastes, saving time and effort.
- *Enhance user retention.* Recommender systems encourage users to spend more time on a platform. When users receive personalized and engaging content, they are more likely to remain and continue using the services.
- *Improvement in profitability.* Recommender systems can enhance sales and user spending on entertainment, as users are more likely to purchase or consume content that aligns with their interests.
- *Exploring new opportunities.* Recommender systems assist users in discovering new genres, authors, games, movies, music bands, and more that they may not have otherwise considered. This contributes to broadening their interests and cultivating new hobbies.
- *Content analysis and improvement.* Recommender systems provide valuable insights to platform operators and content creators about which content is popular among users. This information helps them analyze and enhance the quality of the content they offer.

2.1. Recommender systems types

Many algorithms and recommendation methods available today can be categorized into three main types: collaborative filtering, content filtering, and contextual filtering [1-5].

Collaborative filtering algorithms recommend items based on the preferences of many users. This approach takes advantage of the similarities in user behavior. Recommending algorithms can predict future interactions between users and items by analyzing past interactions. These systems build a model using a user's past behaviors, such as previously purchased items or ratings given to them, as well as the choices made by similar users [1]. The underlying idea is that if several people have made similar decisions or purchases in the past, there is a high likelihood that they will make similar choices in the future, such as selecting movies.

Content filtering, in contrast, utilizes the attributes or features of an object to recommend similar objects based on user preferences. This approach relies on the similarity of characteristics between the items and the user, considering information such as the user's age and average movie reviews. It then models the probability of a new interaction based on this data [6-8].

Hybrid recommender systems integrate the strengths of various types to build a more effective recommender system.

Contextual filtering involves incorporating user context into recommendations. This method utilizes the sequence of user actions and the current situation to predict the likelihood of the next action [6,9-13].

2.2. E-commerce Recommendation Models

Here are the most commonly used approaches for generating recommendations by e-commerce companies:

Popularity-based Recommendations. These involve showcasing products that are currently bestsellers. For instance, "Among Us" surged in popularity in 2020 and reached the top of the Steam bestseller list. This category also includes long-standing popular games, like "Counter-Strike: Global Offensive." Popularity-based recommendations are primarily aimed at new users of the website.

Quality-based Recommendations. This model displays games with a large number of positive reviews and ratings. While it recommends them to users, it may not always be the best approach, as individual tastes can vary greatly. Additionally, a game may be perceived as "overrated" due to its popularity. Newer games often lack sufficient reviews, even if they align well with a user's preferences.

Content-based Recommendations. This model suggests products based on their similarity to other products. It utilizes product descriptions, content, and an understanding of the user's consumption history. For instance, players who enjoy Overwatch and Counter-Strike: Global Offensive may be recommended Valorant, as it shares similar characteristics with both of these games [14].

Collaborative Filtering. In a more specific context, collaborative filtering is a method used to automatically predict a user's interests by gathering preferences from many users (collaboration). The system generates recommendations based solely on the rating information provided by different users or items.

Hybrid recommender systems, which combine multiple models, are the most effective choice for delivering personalized recommendations to customers. For example, Netflix utilizes collaborative filtering to make recommendations by analyzing the viewing and search habits of similar users. Additionally, it employs content-based filtering to suggest items that share characteristics with content the user has highly rated.

Metadata is essential for understanding your products effectively. It helps you organize and categorize your product database more efficiently. High-quality, complete metadata provides personalization algorithms with better data to learn from.

2.3. Matrix Factorization for Recommendations

Matrix factorization (MF) methods form the foundation of many widely-used algorithms, including word embedding and topic modeling. They have emerged as the primary approach in collaborative filtering for recommendations. MF can be utilized to calculate the similarity between user ratings or interactions, which helps in providing personalized recommendations [15,16].

Matrix factorization using the Alternative Least Squares (ALS) algorithm approximates the sparse user-item rating matrix *(u-by-i)* as the product of two dense matrices: the user factor matrix and the item factor matrix. The user factor matrix is of size $u \times f$, while the item factor matrix is of size $f \times i$, where u is the number of users, i is the number of items, and f is the number of hidden features. These factor matrices represent the hidden features that the algorithm aims to identify. One matrix captures the characteristics of each user, while the other describes the hidden attributes of each item. The ALS algorithm iteratively learns (f) numerical factors for both users and items, which represent their respective features [17]. In each iteration, the algorithm alternately fixes one factor matrix and optimizes the other, continuing this process until convergence is reached.

The Alternating Least Squares (ALS) algorithm is a widely used method for factorizing matrices, particularly in the field of recommender systems. The fundamental concept of the algorithm involves finding the factorization matrices iteratively. In each step, one matrix is held constant while the other is updated to minimize the error.

2.4. Deep Neural Network Models for Recommendations

There are various types of artificial neural networks (ANN) [18-20]:

- Artificial Neural Networks (ANNs) that transmit information solely from one layer to the next are known as feedforward neural networks. A specific type of feedforward ANN is the multilayer perceptron (MLP), which comprises at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. MLPs are versatile networks that can be applied in various scenarios.
- Convolutional neural networks are tools used for identifying objects.
- Recurrent neural networks are mathematical models used for analyzing language patterns and data sequences.

Deep learning (DL) recommender models are based on existing techniques such as factorization, which models interactions between variables, and embedding, which is used to manage categorical variables. An embedding is a learned vector that represents the characteristics of an entity, allowing similar entities (whether users or items) to maintain similar distances in vector space. For instance, a deep learning approach to collaborative filtering learns user and item embeddings (latent feature vectors) by analyzing the interactions between users and items using a neural network.

DL technologies also utilize advanced network architectures and rapidly evolving optimization algorithms to learn from large datasets. By harnessing the power of deep learning to extract features, these technologies create more expressive and effective models.

2.4.1. Neural Collaborative Filtering

Neural Collaborative Filtering (NCF) is a neural network that enables collaborative filtering through user and item interactions [15, 18]. The model takes into account matrix factorization with a focus on nonlinearity. NCF TensorFlow accepts a sequence of pairs (user ID, item ID) as input, which it then processes separately. The pairs are sent to the matrix factorization stage, where the embeddings are multiplied, and to the multi-level perceptron (MLP) network.

The matrix factorization results and MLP network outputs are combined and input into a dense layer that predicts if a user is likely to interact with a given item.

2.4.2. Variational Autoencoders for Collaborative Filtering

The autoencoder's neural network reconstructs the input at the output level by leveraging the representation obtained from the hidden layer. In the context of collaborative filtering, the autoencoder learns a nonlinear representation of the user-item matrix and is capable of reconstructing it by identifying and filling in missing values [19].

The model is composed of two main components: an encoder and a decoder. The encoder is a fully connected feed-forward neural network that transforms an input vector, which contains user-specific interactions, into an n-dimensional variation distribution. This variation distribution is utilized to create a hidden representation of the user's features, also known as an embedding. This hidden representation is then passed to the decoder, which is another feed-forward network with a structure similar to that of the encoder. The output is a vector of probabilities representing the likelihood of interaction between items for a specific user.

2.4.3. Learning Sequences in Context

Recurrent Neural Networks (RNNs) are a type of neural network that incorporates memory or feedback loops, enabling them to better recognize patterns in data. RNNs are particularly effective in handling complex tasks related to context and sequence, such as natural language processing and contextual sequence recommendations. What sets sequence learning apart from other tasks is the necessity for models with active data memory, like Long Short-Term Memory (LSTM) networks or Gated Recurrent Units (GRU), to understand the temporal dependencies in the input data [21-22]. This retention of past inputs is essential for achieving successful sequence learning.

On the other hand, transformer-based deep learning models, such as Bidirectional Encoder Representations from Transformers (BERT), offer an alternative to RNNs by utilizing an attention mechanism. This allows them to analyze a sentence by focusing on the most relevant words both before and after a target word. Unlike RNNs, transformer-based models do not require sequential data processing, enabling greater parallelization and significantly shorter training times on GPUs.

2.4.4. Wide & Deep

Wide & Deep refers to a type of network architecture combining the outputs of two parallel components: the Wide and Deep models. The results from these models are then summarized to calculate the probability of interaction. The Wide model is essentially a generalized linear model that focuses on the objects and their transformations. In contrast, the Deep model consists of a dense neural network (Deep-NN) made up of five hidden layers of multilayer perceptrons (MLP), with each layer containing 1,024 neurons. The Deep model begins with a dense feature embedding, where categorical variables are transformed into continuous vector representations. These embeddings can be learned automatically or defined by the user before being input into the Deep-NN [23].

The success of this model in recommendation tasks stems from its dual approach to learning patterns in data: both "deep" and "shallow." The deep neural network (DNN) component excels at learning complex, nonlinear relationships within the data and can generalize similar elements through embeddings. However, this DNN requires many examples of these relationships to perform well. In contrast, the linear component is adept at "memorizing" simpler relationships that may appear only a few times in the training dataset.

2.4.5. Deep Learning Recommendation Model (DLRM)

DLRM, developed by Facebook Research, is a deep learning-based recommendation model. It effectively handles both categorical and numerical inputs, often found in recommender systems' training data. To manage categorical data, embedding layers convert each category into a dense representation before passing it into multi-layer perceptrons (MLPs). Numerical features can be directly inputted into the MLP [24].

DLRM distinguishes itself from other deep learning-based recommender systems in two key ways. First, it focuses on feature interactions by limiting them to pairwise interactions. Second, DLRM treats each embedded feature vector corresponding to categorical features as a single unit. In contrast, other methods consider each individual element of the feature vector as a separate unit, which leads to the generation of different cross-terms. These design choices help to reduce computational and memory overhead while still achieving competitive accuracy.

3. The Significance of Recommendations in E-Commerce Systems

Personalization has become a key success factor for online retailers. These businesses are increasingly focused on enhancing personalization, whether by addressing customers by name in their communications or by offering special promotions based on individual interests.

Recommendations represent the highest level of *personalization* and are vital for customers and the companies serving them. For customers, the benefits include an enhanced user experience, a sense of being understood and valued, and more tailored offers and promotions. For businesses, the advantages are equally significant: improved customer engagement, substantial customer growth, increased web traffic, and higher sales and revenue.

A good example of the impact of recommender systems on business is Netflix. Although Netflix started as a movie rental service, it now streams movies and has over 200 million paying customers worldwide [25-27]. A crucial aspect of this development is their personalized recommendation system.

Netflix's recommendation engine provides a diverse range of content tailored to your preferences. Over the years, hundreds of engineers have developed these recommendation systems by analyzing millions of users. When a new subscriber joins, Netflix prompts them to select their favorite shows and movies. As subscribers watch more content over time, the platform offers suggestions based on these selections, as well as other factors [25].

Steam is a digital game distribution system with 69 million daily active users and a catalog of over 101,035 games available on Steam as of June 2024 [27]. Steam users logged 37.87 billion hours of playtime in 2021. Closer to the topic of our study is Steam, a digital game distribution system with 69 million daily active users and a catalog of over 101,035 games available on Steam as of June 2024 [27]. Steam users logged 37.87 billion hours of playtime in 2021.The platform features a powerful video game recommendation system designed to help players discover games they are likely to enjoy. It suggests games based on factors such as your gaming history, purchase records, browsing activity in the store, and preferences of other players with similar tastes.

However, despite recognizing the importance of high-quality recommendations by online retailers, they have yet to perfect the art of product suggestion. The main issue with current online recommendation systems is that they are primarily behaviorally driven rather than motivational. While many people may play the same game, their reasons for doing so can differ significantly.

A player's motivation is shaped by their emotional and psychological makeup, which includes aspects such as values, personality, and life circumstances. To improve video game recommendations, it's essential to start by understanding the games you recommend and the reasons behind why players enjoy them. Next, analyze your user base to understand each individual on a deeper level. Finally, determine why players choose specific games and leverage that insight to enhance your video game recommendations.

Machine learning algorithms that are intelligent, when paired with high-quality data, are your greatest assets.

4. Selection Methods of Generating Recommendations for Choosing Video Games

Recommender systems typically consist of three key components:

- *Candidate generation* is responsible for creating smaller subsets of candidates to recommend to users from a large pool of thousands of products.
- *Evaluation systems.* We need to standardize candidate generation from various sources and implement a scoring system to assign scores to each item in the subsets.
- *Ranking systems.* After the evaluation process, the system considers additional constraints to produce the final ratings. [28].

4.1. Content-Based Filtering

Content-Based Filtering (CBF) is a type of recommendation system that predicts a user's preferences or behaviors by analyzing the features of items they have responded to positively.

Once we understand the user's preferences, we can represent them in an embedding space using a generated feature vector. Recommendations are then made based on this representation. During the recommendation process, similarity scores are calculated by comparing the feature vectors of the items with the user's preferred feature vectors from their previous selections. Finally, the top recommended items are presented to the user [29].

Content-based filtering recommends items to a user based solely on that user's preferences without relying on data from other users.

4.2. Collaborative Filtering (CF)

The collaborative filtering system operates without the need for specific functions of the provided elements. Instead, it describes each user and item through a feature vector or embedding. The system independently generates embeddings for users and items, positioning them within the same embedding space.

Additionally, it considers the reactions of other users when recommending items to a particular user. It observes not only the products that a specific user likes but also the products favored by users with similar behaviors and preferences. This information is used to suggest relevant products to the user.

Moreover, the system collects user reviews for various products and employs these insights to enhance its recommendations. The sources of user interaction with an item include:

- *Implicit feedback.* User preferences and dislikes are recorded based on actions like clicks, searches, and purchases. There are many preferences noted, but negative reviews are fewer.
- *Explicit feedback.* The user expresses their preferences by reacting to or rating a product. There are both positive and negative reviews, but the negative ones are fewer.

4.2.1. Types of Collaborative Recommender Systems

Collaborative filtering based on memory refers to remembering the user's interaction matrix with an object and how the user responds to it, specifically the user's evaluation of that object. This approach does not involve dimensionality reduction or model fitting.

Collaborative Filtering Based on a Model. In this approach, there's no need to memorize the interaction matrix itself. Instead, we analyze how a particular user or item behaves by compressing the large interaction matrix through dimensionality reduction or clustering algorithms. Machine learning models are utilized to predict the ratings a user is likely to give a product. Several methods exist, including clustering algorithms, matrix factorization techniques, and deep learning methods.

Clustering Algorithms. Typically, simple clustering algorithms are employed, such as K-Nearest Neighbors (KNN), to identify the K nearest neighbors or embeddings based on the chosen similarity metrics.

Algorithms Based on Matrix Factorization. Just as any large number can be decomposed into smaller components, a user-item interaction matrix can also be decomposed into two smaller matrices. These matrices can then be used to reconstruct the interaction matrix. We generate factor matrices that represent features for users and items. These feature matrices serve as embeddings for each user and item. To create these feature matrices, it is necessary to perform dimensionality reduction [30]. The number of feature vector functions depends on the number of domains or features we need to consider to effectively represent users and items. To achieve this, we must identify the principal components of the distributions of users and items. This process, known as dimensionality reduction, aims to clearly represent the distribution with the smallest possible number of features. There are several methods for reducing dimensionality, including Singular Value Decomposition (SVD), Probability Matrix Factorization (PMF), and Non-Negative Matrix Factorization (NMF).

However, it is important to note that matrix factorization methods also have their limitations.

- The challenge of utilizing additional characteristics that may influence recommendations, such as a movie's U/PG rating or the user's country, can be significant. In the case of matrix factorization, only the product ID and user ID can be employed. Additionally, this approach does not permit querying for products or users that are not included in the training set.

- Matrix factorization faces a cold start problem because it lacks a feature vector or embedding for new items.
- It often recommends popular products to all users, which may not accurately reflect their individual interests, especially when using dot multiplications.
- Matrix factorization relies on basic user intrinsic product embeddings and product feature embeddings, which are often inadequate for capturing and representing complex relationships between users and products.

Deep neural networks have been developed to address the shortcomings of matrix factorization methods.

4.3. Algorithms for Generating Recommendations for Choosing Video Games

We decided to use three different algorithms to generate recommendations for each user. Two approaches will utilize collaborative filtering—one based on the ALS algorithm and the other using the EM and SVD algorithms. Additionally, we will implement a content-based method for recommendations.

4.3.1. Model Alternating Least Squares (ALS)

We present a straightforward implementation of a collaborative recommendation filtering algorithm using matrix factorization with implicit data. Matrix factorization maps the latent vectors of users and items—or dense feature vectors used to describe objects or users—into a single latent or embedding space. The interaction between a user and an item is represented as the dot product of the user and item vectors.

In this approach, the Alternating Least Squares (ALS) model is employed to analyze the data and generate recommendations using the least squares method [29,30]. The ALS (Alternating Least Squares) algorithm utilizes matrix factorization, which involves splitting a large matrix into smaller matrices whose product equals the original matrix. In the context of a collaborative recommender system with implicit data, this technique reduces the original matrix, which represents "all users vs. all features," into smaller matrices: "all users vs. some features" and "all features vs. some features." The features derived from the data do not necessarily correspond to real metadata.

ALS is an iterative optimization process that aims to achieve a closer factorized representation ($U \times V$) of the original matrix R (Figure 1) at each iteration. Here R is the original user-item matrix containing implicit data. The matrices U and V contain weights that indicate the correlation between each user and item with respect to various characteristics. The objective is to determine the weights of U and V such that R is approximately equal to $U \times V$ ($R \approx U \times V$).

The ALS algorithm alternates between optimizing U while keeping V fixed, and vice versa, until it converges to a solution that best approximates R.

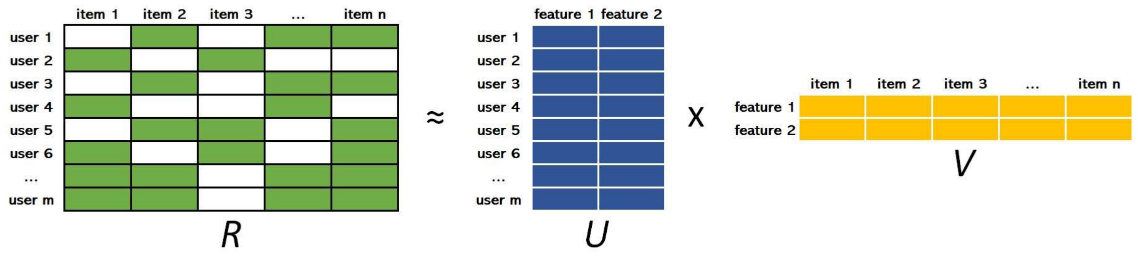


Figure 2: Scheme of the ALS algorithm

Suppose we have a matrix R of size $m \times n$, where the elements R_{ij} represent ratings or relationships between objects (for example, user ratings for movies). The objective is to factorize the matrix R into two matrices: $U(m \times k)$and $V(k \times n)$, where k is the number of latent factors we want to identify. The goal is to minimize the reconstruction error:

$$\sum_{(i,j)\in K} \left(R_{ij} - U_i V_j^T\right)^2 + \lambda(\|U\|^2 + \|V\|^2), \tag{1}$$

where $\lambda(\|U\|^2 + \|V\|^2)$ — the Frobenius norm measures the distance between the original matrix and its approximation through factorized matrices.

The first part of the formula focuses on minimization, while the second part is for regularization to prevent overfitting and manage the size of the elements in the U and V matrices.

Initialization: The matrices U and V can be initialized with either random values or alternative methods, such as singular value decomposition (SVD).

Alternative updates.

Step 1: Update the matrix U.

Update the matrix U while keeping V fixed. For each user i, we minimize the function

$$\min_{\{U,V\}} \sum_{(i,j)\in K} \left(R_{ij} - U_i V_j^T\right)^2, \tag{2}$$

where R_{ij} — this is a recognized element of the matrix R; $U_i$and V_j^T — are the corresponding rows from the matrices U and V.

This equation can be solved using the least-squares method, providing a value for each user.

Step 2: Update the matrix V.

Update the matrix V while keeping U fixed. For each element j, we minimize function (2), which can be solved using least squares. We apply the least squares method to reduce the error during each update.

The process of updating the matrices U and V continues until a certain stopping condition is reached, for example, when the changes in the matrices between iterations become very small or the maximum number of iterations is reached.

4.3.2. Collaborative Recommender for EM and SVD

Expectation-Maximization (EM) is an approach for estimating the Maximum Likelihood (ML) or posterior estimation (MAP) of model parameters in cases where the data have hidden (latent) variables. It is advisable to use it to estimate the parameters of a given data distribution. It is an iterative algorithm for estimating the parameters of statistical models when part of the data is hidden (unknown) or when probabilities have a complex structure.

Suppose we have some observable data X, but there are also hidden variables Z that we cannot see. We want to find the model parameters θ that maximize the likelihood:

$$L(\theta) = P(X|\theta). \tag{3}$$

However, because of the hidden variables Z, it is difficult to calculate this probability directly. The EM-algorithm approximates the solution using *two phases*: E-step (*expectation*) – estimating the hidden variables Z based on the current parameters $\theta^{(t)}$; M-step (*maximization*) – updating the parameters θ to increase the likelihood.

These two steps are repeated until the changes in the parameters become insignificant.

EM maximizes the expected logarithm of the likelihood:

E-step: Calculate the mathematical expectation of the log-likelihood given the current parameters:

$$Q\left(\theta|\theta^{(t)}\right) = E_{Z|X,\theta^{(t)}}[logP(X,Z|\theta)]. \tag{4}$$

M-step: Update the parameters by maximizing the function $Q\left(\theta|\theta^{(t)}\right)$:

$$\theta^{(t+1)} = arg \max_{\theta} Q\left(\theta \middle| \theta^{(t)}\right). \tag{5}$$

In order to come up with a rating system (since the user dataset contains implicit data), it was decided to use the distribution of hours played for each game using the EM algorithm rather than percentiles.

We used the Singular Value Decomposition (SVD) algorithm to factorize the matrix of user items into singular vectors and singular values (similar to the eigendecomposition) and the gradient descent approach to fill in the missing data using prediction. Gradient descent is a convex optimization method used to find the optimal matrices U and V representing the original matrix R of user-items, replacing missing values with new ones estimated from similar users and games.

Any matrix R can be decomposed as follows:

$$R = U\Sigma V^T, \tag{6}$$

where U — matrix of left singular vectors (users) $U(m \times k)$: та $V(k \times n)$;

Σ — diagonal matrix of singular values;

V^T — a matrix of right singular vectors (elements rated by users).

Each column in U and V can be interpreted as a latent factor that describes hidden features of users and objects.

SVD is a powerful method for recommender systems, but it is rarely used in its pure form due to scaling issues. However, it is the basis of modern hybrid recommender models.

4.4. Class Diagram

A class diagram is constructed to study the algorithms for generating recommendations for choosing video games. Figure 2 (Class diagram) shows the generalization between the *VideoGameRecommendation* superclass and three subclasses *ImplicitCollaborativeRecommendation*, *CollaborativeRecommendationWithEMAndSVD* and *ContentBasedRecommendation.*

VideoGameRecommender. Class attributes: Recommender_name (private, type String). Class operations: VideoGameRecommender() – конструктор (public, attributes: recommender_name(String)); generateRecommendations() – (public, return type: List).

ImplicitCollaborativeRecommender. Class attributes: __model (private, type List); __m_user_item (private, type List); __m_item_user (private, type List). Class operations: ImplicitCollaborativeRecommender() – конструктор (public, attributes: data_path(String)); load_data() – (private, attributes: data_path(String)); load_model() – (private); similar_items() – (private, attributes: items(List), n_similar(Integer), return type: List); recommend() - (private, attributes: users(List), n_recommendation(Integer), return type: List).

CollaborativeRecommenderWithEMAndSVD. Class attributes: __prediction_matrix(private, type List). Class operations: CollaborativeRecommenderWithEMAndSVD() – конструктор (public, attributes: data_path(String)); load_data() – (private, attributes: data_path(String)); create_prediction_matrix() – (private); top() – (public, attributes: user(List), n(Integer), return type: List).

ContentBasedRecommender. Class attributes: __data_matrix (private, type List); __cosine_sim_matrix (private, type List). Class operations: ContentBasedRecommender() – конструктор (public, attributes: data_path(String)); load_data() – (private, attributes: data_path(String)); ger_recommendations() – (private, attributes: cosine_sim(List), title(String), return type: List); make_recommendation_for_user() - (public, attributes: user_Id(String), game_list(List), games_user_has(List), return type: List).

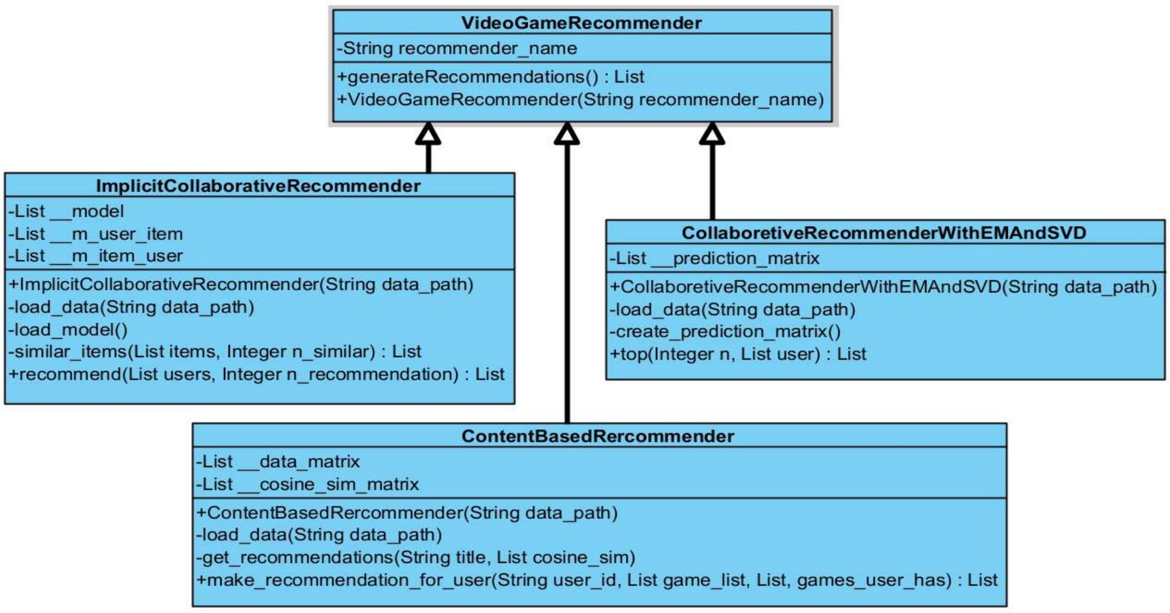

VideoGameRecommender
-String recommender_name
+generateRecommendations() : List
+VideoGameRecommender(String recommender_name)
ImplicitCollaborativeRecommender
-List __model
-List __m_user_item
-List __m_item_user
+ImplicitCollaborativeRecommender(String data_path)
-load_data(String data_path)
-load_model()
-similar_items(List items, Integer n_similar) : List
+recommend(List users, Integer n_recommendation) : List
CollaboretiveRecommenderWithEMAndSVD
-List __prediction_matrix
+CollaboretiveRecommenderWithEMAndSVD(String data_path)
-load_data(String data_path)
-create_prediction_matrix()
+top(Integer n, List user) : List
ContentBasedRercommender
-List __data_matrix
-List __cosine_sim_matrix
+ContentBasedRercommender(String data_path)
-load_data(String data_path)
-get_recommendations(String title, List cosine_sim)
+make_recommendation_for_user(String user_id, List game_list, List, games_user_has) : List

Figure 2: Class diagram

5. Study of the Functioning of the Proposed Methods for Generating Video Game Recommendations

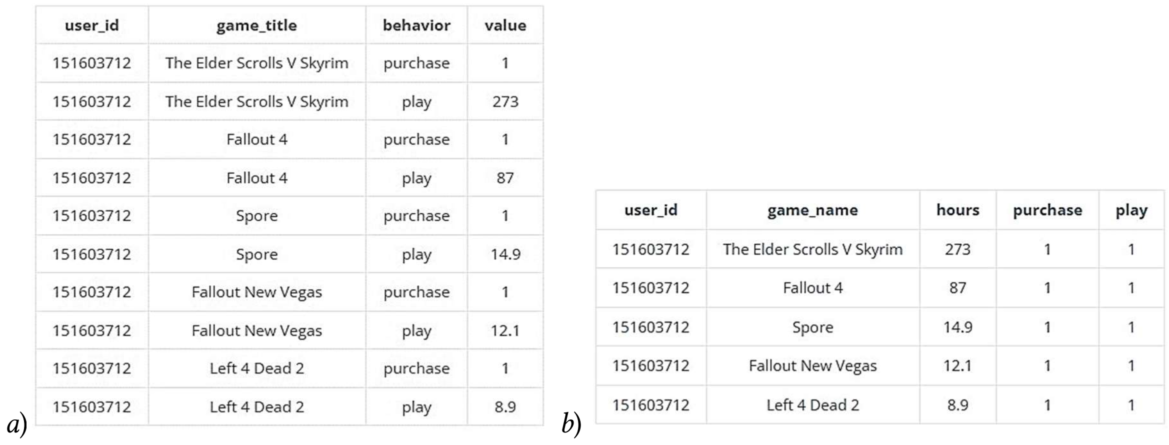
5.1. Formation of Data Sets for Testing Recommendation Algorithms

Two different data sets are used for this project. Both are available for free on Kaggle and contain data obtained from Steam. We used data for 2023 for the study.

5.1.1. User Data Set

The first dataset is the user dataset. It contains a user ID, a game name, a behavior ("purchase" or "play"), and a value associated with that behavior. Each dataset row represents the user's behavior towards the game - "play" or "purchase". If the behavior is "play", the value associated with it corresponds to the number of hours spent playing the game. If the behavior is "buy", the value associated with it is 1, which means that the user bought the game. In the case of this user data set, the value associated with "buy" is always 1.

Fig. 3 shows a part of the user data set. Column headings have been added for convenience based on the data description.



a)

user_id	game_title	behavior	value
151603712	The Elder Scrolls V Skyrim	purchase	1
151603712	The Elder Scrolls V Skyrim	play	273
151603712	Fallout 4	purchase	1
151603712	Fallout 4	play	87
151603712	Spore	purchase	1
151603712	Spore	play	14.9
151603712	Fallout New Vegas	purchase	1
151603712	Fallout New Vegas	play	12.1
151603712	Left 4 Dead 2	purchase	1
151603712	Left 4 Dead 2	play	8.9

b)

user_id	game_name	hours	purchase	play
151603712	The Elder Scrolls V Skyrim	273	1	1
151603712	Fallout 4	87	1	1
151603712	Spore	14.9	1	1
151603712	Fallout New Vegas	12.1	1	1
151603712	Left 4 Dead 2	8.9	1	1

Figure 3: View of the user data set: *a*) initial; *b*) reformatting

The user dataset contains a total of 200,000 rows, including 5,155 unique games and 12,393 unique users. For convenience, the structure of the user dataset has been reformatted (Figure 3,b) (the information stored in the "behavior" column is divided into two columns: "purchase" and "play"). For each row, the "play" column has a value of 1 if the user has actually played the game or 0 if the user has no record of hours spent. Each row in the reformatted user dataset represents a unique user-game interaction.

Using the reformatted user dataset, we began to explore and analyze the data stored in it. The main task is to assess whether the most purchased games correspond to the most played games. For each game, we calculated the total number of users and the total time spent in the game by all users.

We used a bar chart to better visualize the results for the top 20 games with the largest number of users (Figure 4).

Game titles are organized in descending order by the number of users. The color gradient reflects the total number of hours spent playing the game, from highest to lowest. We also counted only those users who actually played the games. Thus, users who purchased it but never played it were removed for each game.

To better understand user data distribution and gaming habits, we plotted the top 20 most popular games based on the total number of hours spent playing the game (Figure 5).

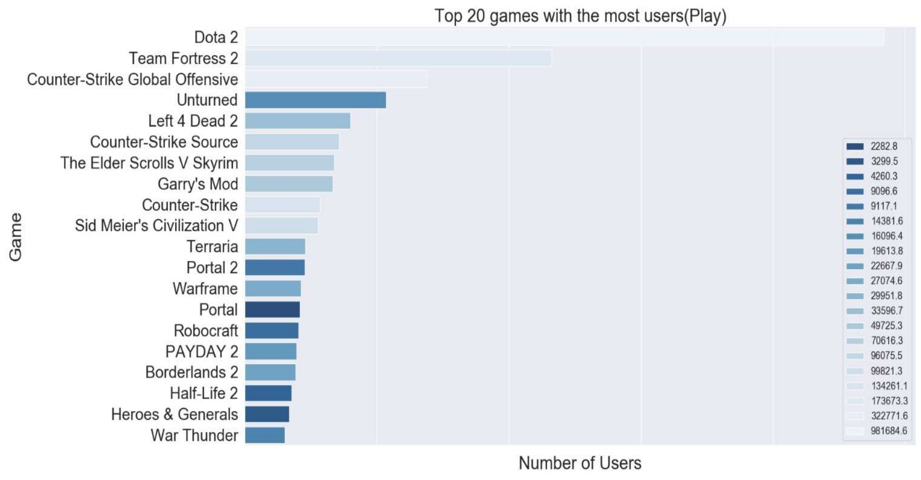


Figure 4: Top 20 games with the largest number of players

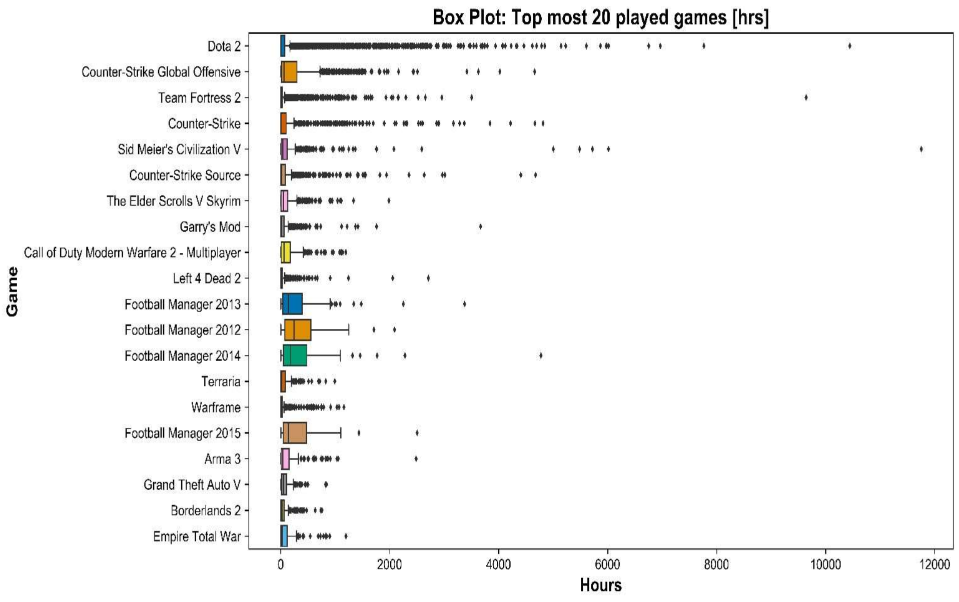


Figure 5: Top 20 games with the most hours played

As you can see, the distribution of data for each of the games considered is not symmetrical. Moreover, 75% of the data points for each game are in the range of hundreds of hours, with several games having very large outliers. For example, you can see that a user has played more than 10,000 hours of Dota 2. Another interesting example: a user has played almost 12,000 hours of Sid Meier's Civilization V.

5.1.2. Game Data Set

The second dataset is the games dataset. It contains a list of games, their descriptions, URL (pointing to the Steam store), package type (single game, bundle...), game title, short description, recent reviews, all reviews, release date, developer, publisher, popular tags (Action, Shooter, PvP...), game details (multiplayer, single player, full controller support...), languages, achievements, genre (Action, Adventure, RPG, Strategy...), game description, adult content description, minimum requirements to run the game, recommended requirements, original price and discount price. The dataset contains 51920 games in total.

5.2. Study of the Proposed Recommendation Generation Algorithms

All three algorithms generate recommendations for the same users, allowing us to compare their results and evaluate which approach best suits a given project.

5.2.1. Collaborative Filtering

Before implementing the algorithms that will be used for the collaborative recommendation filtering system, a training and test dataset is created from the reformatted user data set.

The reformatted version of the user dataset contains 128804 rows, each containing unique information about the user's interaction with the game. It was decided to remove 20% of all user interactions with the game (25761 rows) for the test dataset and leave the rest (103043 rows) for the training dataset.

The training dataset will be used to implement collaborative recommendation filtering models. Once completed, the models will be used to generate recommendations for all users listed in the test dataset.

Collaborative Recommender with ALS

Collaborative filtering requires no information about objects or users to make recommendations. It only uses the interaction between users and objects, expressed by a certain rating. The data used for this recommendation system is from the reformatted Steam user dataset. This data does not explicitly contain users' ratings or preferences for games but rather is expressed by the number of hours users have spent playing games.

The ALS algorithm, available through the "implicit" library, is a recommendation model based on performance-optimized algorithms [29,30]. The advantage of using the implicit library compared to manual algorithm implementation is the speed required to generate recommendations since the ALS model in the existing library uses Cython, which allows parallelizing the code between threads.

The ALS model uses two separate quantities (preferences and trust level) to express raw user observations. For each user interaction with an item in the data, it calculates a score that expresses whether the user likes or dislikes the item (i.e., preference) and combines this score with a confidence level directly related to the value of the raw implicit observations (higher confidence levels the more the user has played the game).

To create recommendations using the ALS algorithm described above, the ImplicitCollaborativeRecommender class is implemented in a Python script. The class was developed based on the recommendations from the sources mentioned above. The class performs all the necessary manipulations with DataFrame data to create the matrices required by the ALS algorithm. We used methods already implemented around the ALS algorithm from the "implicit" library to create recommendations.

The collaborative recommender model is created using the training user dataset and the ImplicitCollaborativeRecommender class.

After the model is successfully loaded, you can start generating recommendations for all users for whom the user-item interaction was hidden during data partitioning training and testing. For each user, 20 recommendations are generated. The recommendations are stored in a Pandas DataFrame, which is later output as a CSV file.

It should be noted that the model failed to provide recommendations for some users. This is due to the fact that many users have only one user-object interaction that was included in the test dataset. Therefore, since the model has no prior knowledge of the preferences of these users, it cannot provide any recommendations. For these cases, the output values are set to '-999'.

Collaborative Recommender with EM and SVD

EM algorithm: Distribution of playing time. To come up with a ranking system (since the user dataset contains implicit data), it is decided to use the distribution of hours played for each game using the EM algorithm rather than percentiles.

A ranking system is created based on the distribution of time spent in the game for each game available in the user data set. 5 groups (equivalent to a 5-star rating system) are used to determine the rating that users can give to the game they have played based on the time each user has spent in each game relative to the time spent by all others.

Steam allows users to get a refund for games they have played for less than 2 hours. This fact was taken into account for the recommendation system. Thus, user interaction with an item lasting less than 2 hours is not taken into account. An example of the EM algorithm's timeline for a given game can be seen in Figure 6.

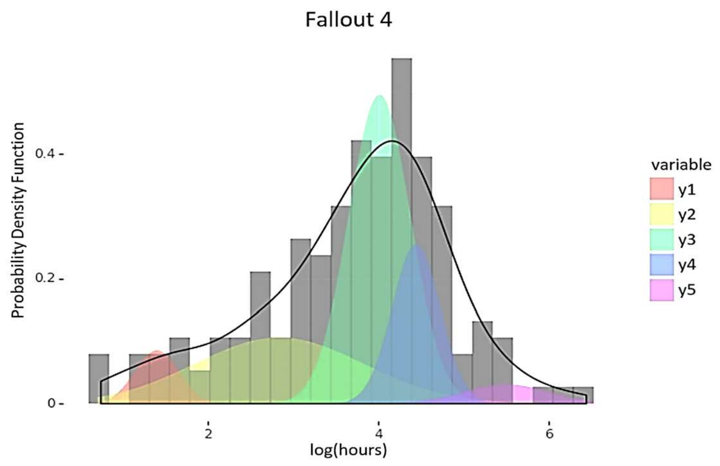


Figure 6: How the EM algorithm works for a given game

As you can see in the graph above for The Fallout 4, the EM algorithm does a great job of finding 5 groups of people with similar gaming habits who could potentially similarly rate the game. You can see that some users have played "The Fallout 4" for very few hours. It is possible that some of these users lost interest in the game shortly after they started playing it. For Groups 3 and 4, the distribution is tighter. This indicates that most users are interested in this game. Therefore, this game will have a high rating. A "user-items" matrix is created with users as rows and games as columns. The missing values are zero. The values stored in the matrix correspond to the number of hours for each user-game combination. Following the source's recommendations, the data used to create the user-game matrix only includes games with more than 50 users and users who have played the game for more than 2 hours.

It was decided to implement the *SVD algorithm* using the gradient descent method manually [31]. We set the learning rate to 0.001 and the number of iterations to 200, tracking the root mean square error (RMSE) (Figure 7). The U and V matrices are initialized with random values drawn from the normal distribution [0, 0.01]. The tracking function measures the RMSE between the actual and predicted values.

The graph shows that the SVD using gradient descent converges to zero on the training dataset, while the RMSE for the training dataset remains at around 0.68.

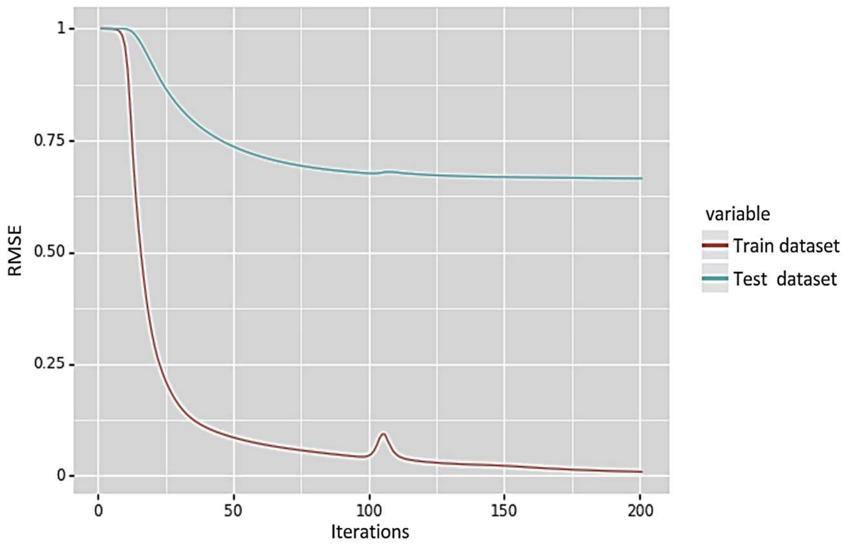


Figure 7: SVD accuracy on training and test datasets

Interestingly, after the 75th to 100th iteration, the accuracy on the test dataset stops improving (the RMSE remains at about the same level). The accuracy on the test data could be improved by using more leading components, but this would increase the time required for computation. Consequently, for the data used, one could stop the computation after 75 to 100 iterations because the accuracy on the test data set is no longer improving.

EM Algorithm: Post SVD via Gradient Descent. With the matrix of predicted user elements, the distribution of hours for the game "Fallout 4" was reviewed using the EM algorithm to find a reasonable rating from 1 to 5 stars.

The results of the EM algorithm for this game are plotted, this time using the predicted matrix of user elements obtained with the SVD algorithm using the gradient descent method (Figure 8).

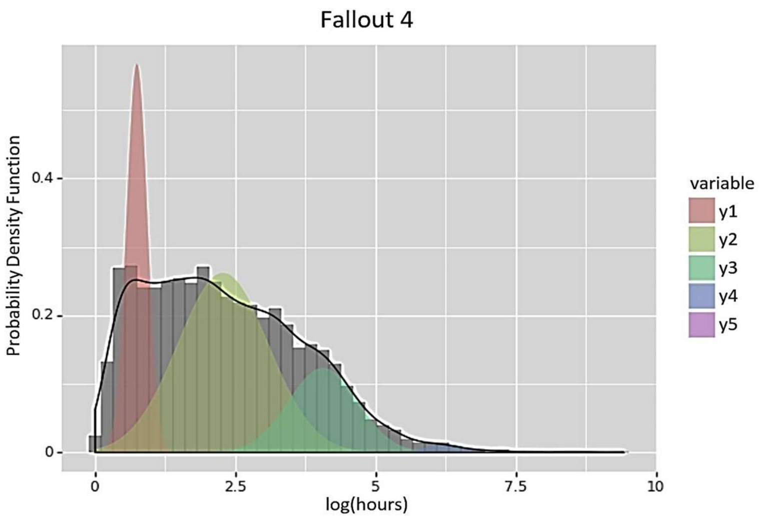


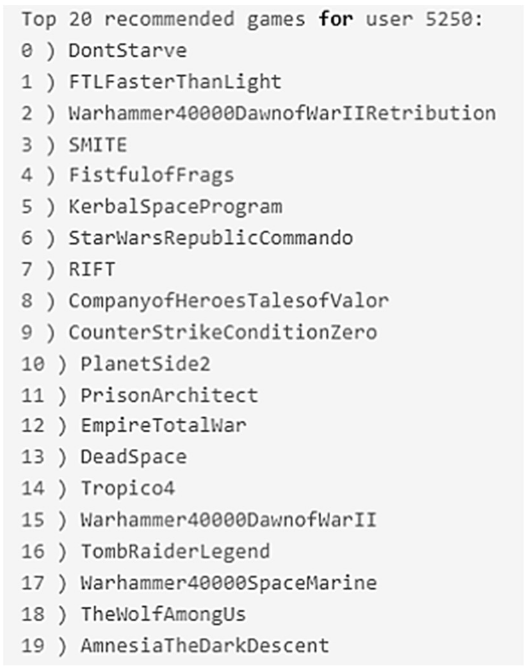
Figure 8: Operation of the modified EM algorithm for a given game

As shown in Figure 8, distributions 2-4 look like they fit the data quite well. However, this is not the case for distribution 1. On the other hand, distribution 5 is practically flat on the right-hand side.

To complete the study, we used the SVD algorithm via gradient descent to generate the top 20 game recommendations for each user from the test dataset.

Similar to the collaborative recommender with the ALS algorithm, the recommender does not make recommendations for users that exist only in the test dataset. Therefore, since the model has no prior knowledge of the observations of these users, it cannot provide any recommendations. For these cases, the output values are set to "0".

As an example, the top 20 game recommendations for user "5250" are shown in Figure 9.



```
Top 20 recommended games for user 5250:
0 ) DontStarve
1 ) FTLFasterThanLight
2 ) Warhammer40000DawnofWarIIRetribution
3 ) SMITE
4 ) FistfulofFrags
5 ) KerbalSpaceProgram
6 ) StarWarsRepublicCommando
7 ) RIFT
8 ) CompanyofHeroesTalesofValor
9 ) CounterStrikeConditionZero
10 ) PlanetSide2
11 ) PrisonArchitect
12 ) EmpireTotalWar
13 ) DeadSpace
14 ) Tropico4
15 ) Warhammer40000DawnofWarII
16 ) TombRaiderLegend
17 ) Warhammer40000SpaceMarine
18 ) TheWolfAmongUs
19 ) AmnesiaTheDarkDescent
```

Figure 9: 20 best gaming recommendations for a given user

5.2.2. Content-Based Recommendation

A content-based recommendation system makes recommendations based on the similarity between the game the user already has and other games.

To create a recommender system, you need to prepare data and build an algorithm. To do this, you first need to pre-process the game dataset with all the useful information to feed it to the recommendation algorithm, formatting it more simply. We implement a function that allows us to make recommendations for games that are similar to other games. Finally, we create recommendations for all users based on the games they already have.

To prepare the data for content-based recommendations, we started by selecting the information that would be most useful for finding similar games. Useful columns from the game dataset were read. It was decided to keep only those games that are present in both the game dataset and the user dataset, as many games in the game dataset have never been played or purchased by any user in the user dataset, so it makes no sense to include them in the recommendation system. In addition, the game dataset is too large to create a cosine similarity matrix because it takes up too much memory.

An identifier was created for each game to match games from both datasets, removing all non-alphabetic characters and spaces and changing all uppercase letters to lowercase. The same was done for the games from the custom dataset.

After that, we found all unique identifiers from the user dataset and used them to filter the rows in the game dataset, keeping those with identifiers that match those from the user dataset. This resulted in 3036 games from the game dataset that matched some of the 5152 games in the user dataset. Initially, without the ID approach, using only game titles, only 71 games from the game set were retrieved that matched games from the user dataset. Since there are fewer games in the new game set than in the user set, the recommender system will not be able to find recommendations for every game in the user set. This affects its performance.

In the new smaller game dataset, spaces have been removed from the useful columns that are decided to be used. This way, there is a guarantee that, for example, 'Steam Achievements' and 'Steam Cloud' will not get a match because they both contain 'Steam'. They will now have unique values of 'SteamAchievement' and 'SteamCloud'. Therefore, this feature has been applied to all columns in use.

Finally, some custom columns were created by combining several columns to find a combination of information that could give the best possible recommendation system.

Additional manipulations were made with the reviews column from the game dataset to extract the percentage and any other possible useful information. For this purpose, we used the fact that all reviews have the following format: "Predominantly positive (11481) - 74% of the user reviews for this game are positive".

We started by getting the percentage of positive reviews by using a regex to get the "- 74%" part of the review, leaving only the number. We also got the qualitative information about the reviews by separating them with a comma and keeping only the first element. Qualifications that contain the words "user reviews" are ignored, as this means that not enough users have reviewed the game and the format of the reviews is different.

To do this, we use a script that outputs the result to a CSV file. This CSV file is read by the content-based recommendation script to get feedback.

A cosine similarity matrix is generated for the recommendation system. First, a frequency matrix is created for each word in the selected column (column_name) and for each game. Then, using the frequency matrix, the cosine similarity matrix is created.

To generate recommendations for each game, the get_recommendations function is used, as shown below. The input to the function is the name of the game in the form of a string and the cosine similarity matrix created just before. The output is a list of recommended games ordered by similarity.

The similarity score for each recommended game is obtained from the cosine similarity matrix to order them from most similar to least similar. Finally, we get the number of recommendations we need and return them in the form of a list. The variable 'n_recommendation', set to 20, determines the number of recommendations to be generated.

To get recommendations for each user, the 'make_recommendation_for_user' function is implemented. This function combines the recommendations created for each game using the 'get_recommendations' function, keeping the recommendations with the best reviews (extracted from the game dataset). This function takes three inputs: a user ID, a list of recommendations for the user (the 'get_recommendations' function described earlier is applied to all games the user already has, the results are returned as a list with all recommendations), and a list of all games the user already has. The function returns a Pandas DataFrame containing the user's ID in the first column, followed by 20 columns with the best recommendations.

If the list of recommendations is empty (this can happen if none of the games that the user already has are included in the game data set) or invalid, a DataFrame is returned without recommendations. If the recommendation list is valid, the returned DataFrame contains the recommended game titles with the corresponding rating (percentage of positive reviews). Games that the user already owns are removed (there is no need to recommend a game that the user has already purchased).

The recommendations are then organized according to their reviews, from best to worst. It was decided to do this because it is the easiest way to organize the recommendations, especially since it is not possible to create recommendations for every game the user owns, as the games in both datasets do not match perfectly, as mentioned earlier. If this were not due to a mismatch, consideration was given to taking into account the proportion of playing time each game has relative to other games played by the user to recommend similar games to those played most often. Using reviews to sort recommendations still ensures that the recommended games are considered good overall by all users.

If the number of recommendations is less than the desired number, the remaining columns are filled with spaces. All the DataFrame rows created by the 'make_recommendation_for_user' function are merged and then output as a CSV file.

6. Analysis of the obtained results

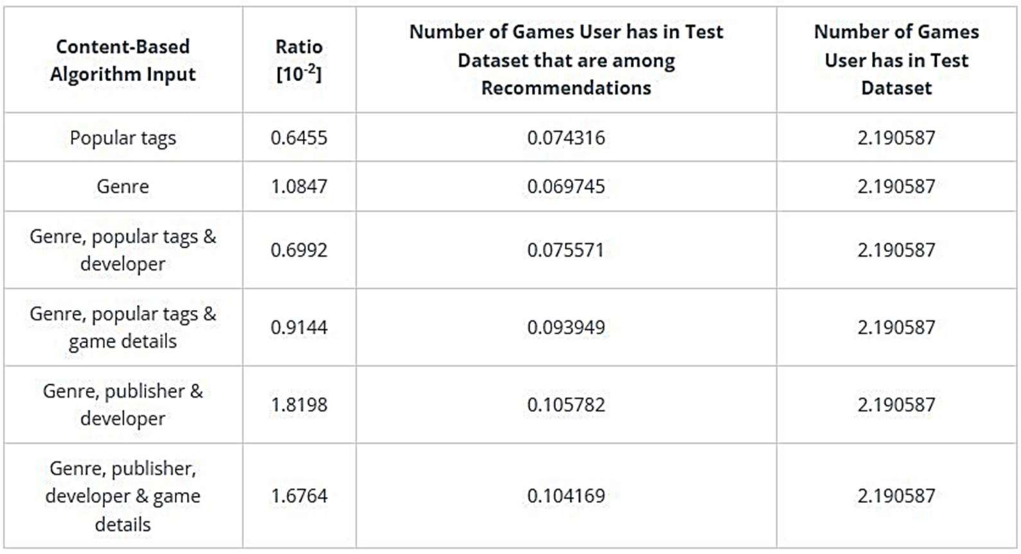
To compare the different algorithms used to generate recommendations, we created an algorithm that calculates for each user the ratio of the number of games in the user's test dataset $N_{ItemUserRecomTest}$ that are in the top 20 recommendations to the total number of games in the user's test dataset $N_{ItemUserTest}$ (7):

$$Ratio = \frac{N_{ItemUserRecomTest}}{N_{ItemUserTest}}, \tag{7}$$

where $N_{ItemUserRecomTest}$ – the number of games in the user's test dataset that are in the top 20 recommendations; $N_{ItemUserTest}$– the total number of games in the user's test dataset.

The average value of the ratio for all users is then calculated. This ratio is somewhat underestimated because if a recommendation cannot be created for a particular user, the calculated ratio is 0.

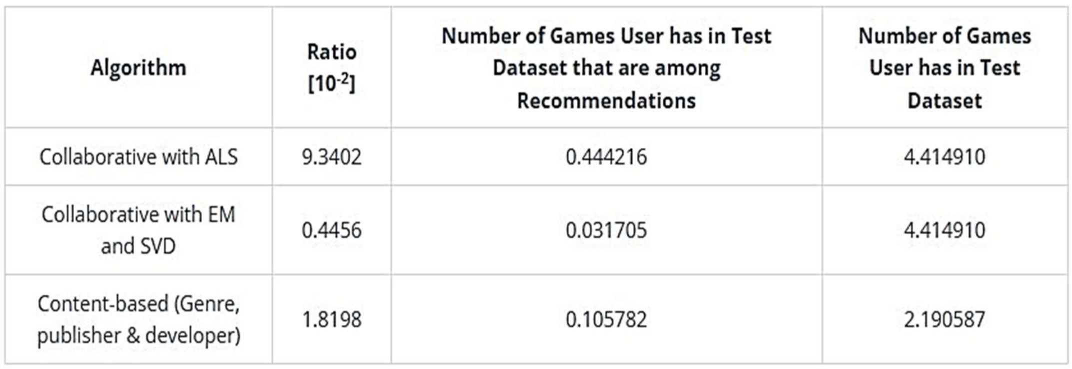
First of all, we created a comparison of the content-based recommendation algorithm with different input data. This is either a column from the original dataset or a combination of different columns. Figure 10 shows the ratio calculated using different inputs. Here, the best result corresponds to using a combination of columns: genre, publisher, and developer. This implementation of content-based recommendations is used in the comparison with the other two collaborative filtering recommendations.



Content-Based Algorithm Input	Ratio [10^{-2}]	Number of Games User has in Test Dataset that are among Recommendations	Number of Games User has in Test Dataset
Popular tags	0.6455	0.074316	2.190587
Genre	1.0847	0.069745	2.190587
Genre, popular tags & developer	0.6992	0.075571	2.190587
Genre, popular tags & game details	0.9144	0.093949	2.190587
Genre, publisher & developer	1.8198	0.105782	2.190587
Genre, publisher, developer & game details	1.6764	0.104169	2.190587

Figure 10: The impact of combinations of characteristics on the result

We calculated the ratios for both collaborative filtering recommendations in the same way as described earlier. Their results, as well as the results of the retained content-based recommender, are shown in Figure 11.



Algorithm	Ratio [10^{-2}]	Number of Games User has in Test Dataset that are among Recommendations	Number of Games User has in Test Dataset
Collaborative with ALS	9.3402	0.444216	4.414910
Collaborative with EM and SVD	0.4456	0.031705	4.414910
Content-based (Genre, publisher & developer)	1.8198	0.105782	2.190587

Figure 11: Comparison of the results of the three algorithms

As you can see from the table, the best recommender is a collaborative recommender with ALS. The performance of the collaborative recommender system with EM and SVD, as well as the content-based recommender system, lags far behind it.

7. Conclusions

As a result of the research, an overview of the latest and most well-known methods, tools, algorithms, and approaches to solving the problem of recommendation generation is made, such as: neural collaborative filtering, variational autoencoder, contextual sequence learning, wide and deep neural networks, and DLRM.

The most important goal of recommender systems is to facilitate the process of choosing video games and provide players with games that best match their interests and preferences. To analyze user preferences and generate personalized video game recommendations, we proposed three algorithms: a collaborative recommender with ALS, a collaborative recommender with EM and SVD, and a content-based recommender, and analyzed their performance results. We used two different datasets for testing, available for free on Kaggle and containing data obtained from Steam. The first dataset is the user dataset, which contains a total of 200,000 rows, including 5,155 unique games and 12,393 unique users. The second dataset is the game dataset, with a total of 51920 game characteristics. As a result of the study, the collaborative recommender with ALS performed best. It was implemented in a prototype video game recommender system. After selecting and evaluating 5 games from the list, the user will be able to receive 5 video game recommendations generated by the system based on his/her choice. This can significantly improve the gaming experience of users, making the process of choosing games more personalized and efficient.

Creating this project helped us better understand how a collaborative filtering system works. It doesn't use any information about objects but relies entirely on user interaction with objects and matrix operations to generate recommendations. We needed to find an approach to work with the dataset (only user data for the collaborative recommender), as it contains only implicit data. Both approaches handle implicit data differently to generate recommendations. For example, Singular Value Decomposition (SVD) is used to remove some noise data as a dimensionality reduction method to make it easier to work with a large dataset.

On the other hand, the content-based approach requires a description of the elements to generate recommendations. Some problems were found when implementing the content-based recommender because it uses two different datasets. When the project started using two datasets (user and game), it was expected to find all the games available in the user dataset in the game dataset since they both come from Steam. However, as the project progressed, it was concluded that this was not the case. Among the 5152 game titles available in the user dataset, only 3036 game titles were found in the game dataset. This poses a serious problem for a content-based recommender because it relies on the assumption that all games available in the user dataset have information in the game dataset. Because of this, it is impossible to create recommendations for every game that a user has purchased.

Moreover, this problem made it impossible to create recommendations for multiple users. This probably affects the performance of the content-based algorithm.

Overall, the results are satisfactory. Moreover, it has provided a better understanding of recommender systems' complexity, stringent requirements, and importance in real life. Of course, there are ways to improve the performance of this model by using more complex algorithms or training it on a better dataset. You can also deepen the system itself by integrating it with a user's Steam account or the history of previous generations.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] L. Liling, Summary of recommendation system development, J. Phys: Conf. Ser. 1187(5) (2019) 052044. DOI:10.1088/1742-6596/1187/5/052044

[2] O. Veres, P. Ilchuk, O. Kots, Information System for Analysis of Hardware Computer Components, International Conference on Computer Science and Information Technologies (2023) 1-4.

[3] L. Halkiv, I. Kulyniak, N. Shevchuk, L. Kucher, T. Horbenko, Information and Technological Support of Enterprise Management: Diagnostics of Crisis Situations, Proceedings - International Conference on Advanced Computer Information Technologies, ACIT (2021) 309–312. DOI:10.1109/ACIT52158.2021.9548354.

[4] O. Veres, P. Ilchuk, O. Kots, Y. Levus, O. Vlasenko, Recommendation System for Leisure Time-Management in Quarantine Conditions, CEUR Workshop Proceedings. 3312 (2022) 263–282.

[5] O. Veres, P. Ilchuk, O. Kots, Methods of recommendations for analysis of computer components, CEUR Workshop Proceedings. 3426 (2023) 385–405.

[6] S. Kulkarni, S. F. Rodd, Context Aware Recommendation Systems: A review of the state of the art techniques, Computer Science Review. 37 (2020) 100255. DOI:10.1016/j.cosrev.2020.100255.

[7] H. Ko, S. Lee, Y. Park, A. Choi, A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields, Electronics. 11(1) (2022) 141. DOI:10.3390/electronics11010141.

[8] Recommender Systems: Behind the Scenes of Machine Learning-Based Personalization, Altexsoft (2021). URL: https://www.altexsoft.com/blog/recommender-system-personalization/.

[9] M. Jalili, S. Ahmadian, M. Izadi, P. Moradi, M. Salehi, Evaluating collaborative filtering recommender algorithms: a survey, IEEE access. 6 (2018) 74003-74024.

[10] O. Bulut, D. C. Cormier, J. Shin, An intelligent recommender system for personalized test administration scheduling with computerized formative assessments, Frontiers in Education (2020). https://doi.org/10.3389/feduc.2020.572612.

[11] U. Javed, K. Shaukat, I. A. Hameed, F. Iqbal, T. M. Alam, S. Luo, A review of content-based and context-based recommendation systems, International Journal of Emerging Technologies in Learning (iJET). 16(3) (2021) 274-306.

[12] M. Nilashi, O. Ibrahim, K. Bagherifard, A recommender system based on collaborative filtering using ontology and dimensionality reduction techniques, Expert Systems with Applications. 92 (2018) 507-520.

[13] S. Sen, Recommender Systems - The Owl - medium. Medium, 2021. URL: https://medium.com/the-owl/recommender-systems-f62ad843f70c.

[14] Steam Search, (n.d.). URL: https://store.steampowered.com/search.

[15] Z. Fayyaz, M. Ebrahimian, D. Nawara, A. Ibrahim, R. Kashef, Recommendation systems: Algorithms, challenges, metrics, and business opportunities, Applied sciences (2020). DOI:10.3390/app10217748.

[16] F. O. Isinkaye, Matrix Factorization in Recommender Systems: Algorithms, Applications, and Peculiar Challenges, IETE Journal of Research. 69(9) (2021) 6087–6100. DOI:10.1080/03772063.2021.1997357.

[17] N. Li, Y. Xia, Movie recommendation based on ALS collaborative filtering recommendation algorithm with deep learning model, Entertainment Computing. 51 (2024) 100715. DOI:10.1016/j.entcom.2024.100715.

[18] S. Rendle, W. Krichene, L. Zhang, J. Anderson, Neural collaborative filtering vs. matrix factorization revisited, In Proceedings of the 14th ACM Conference on Recommender Systems (2020) 240-248.

[19] A. Seghiour, H. A. Abbas, A. Chouder, A. Rabhi, Deep learning method based on autoencoder neural network applied to faults detection and diagnosis of photovoltaic system, Simulation Modelling Practice and Theory. 23 (2023) 102704. DOI:10.1016/j.simpat.2022.102704.

[20] M. Srivastava, YouTube and Movie Recommendation System Using Machine Learning, Second International Conference on Electronics and Renewable Systems (ICEARS), Tuticorin, India (2023) 1352-1356. DOI:10.1109/ICEARS56392.2023.10084999.

[21] J. Nyandwi, The Transformer Blueprint: A holistic guide to the Transformer Neural Network architecture (2023). URL: https://deeprevision.github.io/posts/001-transformer/.

[22] Compare the different Sequence models (RNN, LSTM, GRU, and Transformers) (2024). URL: https://aiml.com/compare-the-different-sequence-models-rnn-lstm-gru-and-transformers/.

[23] S. Dong, P. Wang, K. Abbas, A survey on deep learning and its applications, Computer Science Review, 40 (2021) 100379. DOI:10.1016/j.cosrev.2021.100379.

[24] F. Ricci, L. Rokach, B. Shapira, Recommender Systems: Techniques, Applications, and Challenges, Recommender Systems Handbook. Springer, New York, NY (2022) 1-35.

[25] J. Stoll, Number of Netflix paid subscribers worldwide from 1st quarter 2013 to 3rd quarter 2023, Statista (2023). URL: https://www.statista.com/statistics/250934/quarterly-number-of-netflixstreaming-subscribers-worldwide/.

[26] How Netflix's Recommendations System Works, 2023. URL: https://help.netflix.com/en/node/1 00639.

[27] B. Team, Steam usage and catalog stats, Backlinko (2025). URL: https://backlinko.com/steam-users#steam-monthly-active-users.

[28] G. Geetha, M. Safa, C. Fancy, D. Saranya, A hybrid approach using collaborative filtering and content based filtering for recommender system, Journal of Physics: Conf. (2018). DOI:10.1088/1742-6596/1000/1/012101.

[29] Y. Koren, S. Rendle, R. Bell, Advances in Collaborative Filtering, Recommender Systems Handbook. Springer, New York, NY (2022). DOI:10.1007/978-1-0716-2197-4_3.

[30] J. Chen, J. Fang, W. Liu, T. Tang, C.Yang, clmf: A fine-grained and portable alternating least squares algorithm for parallel matrix factorization, Future Generation Computer Systems. 108 (2020) 1192-1205. DOI:10.1016/j.future.2018.04.071.

[31] M. H. Mohamed, M. H. Khafagy, M. H. Ibrahim, Two recommendation system algorithms used SVD and association rule on implicit and explicit data sets, International Journal of Scientific & Technology Research. 9(1) (2020) 17–24.