Steam Games Recommender System

ECE461 - Machine Learning for Data Science and Analytics - Final Project Paper - Team C

Thomas Katraouras
Department of Electical and Computer Engineering
University of Thessaly
Volos, GR 383 34
tkatraouras@uth.gr

Vasileios Giosis
Department of Electical and Computer Engineering
University of Thessaly
Volos, GR 383 34
vgiosis@uth.gr

Konstantinos Kokkalis
Department of Electical and Computer Engineering
University of Thessaly
Volos, GR 383 34
kokokkalis@uth.gr

Abstract

This paper presents the Steam Games Recommender System, a solution designed to enhance the experience of Steam users by providing personalized game recommendations. The system tackles the challenge of selecting suitable games from the vast array of options available on the Steam platform. Utilizing two extensive datasets from Kaggle, the system employs a blend of collaborative and content-based filtering methods to generate recommendations. These methods include using game tags, publisher information, and user playtime data. The system uniquely combines techniques such as Singular Value Decomposition (SVD), cosine similarity, and Neural Networks with embeddings to achieve high-quality, relevant suggestions. A user-friendly graphical interface facilitates easy interaction with the recommendation system. The paper goes into detail about the methodologies and datasets used, the algorithmic approaches, and the system's performance evaluation, showcasing the effectiveness and user-centric approach of the Steam Games Recommender System in navigating the abundant choices on Steam and significantly enhancing user satisfaction.

1 Introduction

The video game industry has changed the way we search for and play video games through platforms such as Steam. There are so many games nowadays that it is not easy to find the right one among all this multitude. Finding a suitable game in digital stores is a challenge. Our project, the Steam Games Recommender System, aims to solve this by giving personalized game suggestions to Steam users. Our project aims at assisting Steam users to cope with the abundance of information and facilitate the choice of games. We want to ensure that users are satisfied with the use of our platform (Steam) through recommendation of games tailored specifically for their preferences. Our system

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

analyzes the behavior of gamers and game features from two large datasets from Kaggle. The first dataset, steam_200k [25], contains data related to the games that users have and how long they have played them. The second dataset, steam [8] (which will be referred to as steam_games from now on), has information about games on Steam such as game developers, publishers, tags, types and user reviews. The input for our algorithm is either a user or a game, we then use collaborative filtering and content-based filtering to recommend games. More specifically, there are three main methods: 1) recommending games based on how similar the game tags are to the games the user likes, 2) suggesting games from the same publisher as the user's favorite games, and 3) recommending popular games among other users. We also ensure that the recommendations are not only relevant but also of high quality by considering overall game ratings. We designed a user-friendly Graphical Interface through which the user can interact with our recommendation system. We have also attempted to use neural networks for suggesting games, which we discuss later in this paper. In the next sections, we will go into more detail about our methods, the datasets we used, how our recommendation algorithms work, and how well our system performs.

2 Literature Review

As part of our work on this project, we explored several key research papers that significantly contribute to the field. These include *AutoRec: Autoencoders Meet Collaborative Filtering* [21], *Wide & Deep Learning for Recommender Systems* [3], *Collaborative Filtering for Implicit Feedback Datasets* [10], *Matrix Factorization Techniques for Recommender Systems* [17], *Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model* [16], *Learning to rank: from pairwise approach to listwise approach* [2], and *Deep Neural Networks for YouTube Recommendations* [7]. Each of these studies offers unique perspectives and methodologies in recommender systems, ranging from deep learning approaches to collaborative filtering with implicit feedback, latent factor models, and innovative ranking techniques. We discuss the strengths and weaknesses of each approach, identify the most clever and state-of-the-art strategies among them and compare their relevance to our work on our Steam Games Recommender System.

2.1 Papers summary and strengths/weaknesses

We can categorize the 7 papers into 4 broader categories: Deep Learning Approaches, Collaborative Filtering with Implicit Feedback, Latent Factor and Neighborhood Models, Learning to Rank.

2.1.1 Deep Learning Approaches

***AutoRec: Autoencoders Meet Collaborative Filtering* [21]** It utilizes an autoencoder framework for collaborative filtering, efficiently handling partially observed user-item rating matrices. It stands out for its compact modeling and efficient training, although it faces challenges in interpretability and sparse data handling.

***Wide & Deep Learning for Recommender Systems* [3]** It combines wide linear models with deep neural networks, excelling in handling sparse inputs and improving app acquisition rates. This method's complexity, however, necessitates careful tuning and is best suited for large-scale, diverse applications.

***Deep Neural Networks for YouTube Recommendations.* [7]** It details YouTube's deep learning recommendation system, highlighting its scalability and adeptness with large, dynamic datasets. However, its complexity, resource demands, potential for overfitting, and opaque nature, along with reliance on implicit user feedback, pose challenges to its adaptability and interpretability.

2.1.2 Collaborative Filtering with Implicit Feedback

***Collaborative Filtering for Implicit Feedback Datasets* [10]** It focuses on collaborative filtering using implicit feedback data, effectively handling the absence of explicit negative feedback and inherent noise. The approach demands careful tuning and may not directly apply to scenarios involving explicit feedback.

***Matrix Factorization Techniques for Recommender Systems* [17]** It employs matrix factorization techniques for implicit feedback datasets, emphasizing the weighting of observed interactions. This method captures user-item interactions well but might struggle with sparse or noisy data.

2.1.3 Latent Factor and Neighborhood Models

***Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model* [16]** It innovates by merging latent factor and neighborhood models, enhancing accuracy and exploiting both explicit and implicit feedback. Its complexity may pose scalability and interpretability challenges.

2.1.4 Learning to Rank

***Learning to rank: from pairwise approach to listwise approach* [2]** It introduces an innovative listwise approach for ranking tasks, providing a theoretical leap with its probabilistic method and empirical validation. However, its complexity, computational requirements, and focused experiment scope may limit its practical applicability and accessibility to non-specialists in the field.

2.2 Clever/Good Approaches and State-of-the-Art

Wide & Deep Learning for Recommender Systems [3], *AutoRec: Autoencoders Meet Collaborative Filtering* [21], and *Learning to rank: from pairwise approach to listwise approach* [2] represent the most innovative and state-of-the-art approaches. *Wide & Deep Learning for Recommender Systems* is particularly clever for its dual-model integration, *AutoRec: Autoencoders Meet Collaborative Filtering* for its novel use of autoencoders, and *Learning to rank: from pairwise approach to listwise approach* for shifting focus to listwise ranking challenges. The state-of-the-art in recommender systems is represented by approaches that effectively combine deep learning techniques with traditional collaborative filtering methods, as seen in *AutoRec: Autoencoders Meet Collaborative Filtering* and *Wide & Deep Learning for Recommender Systems*.

2.3 Comparison with Our Work

Our Steam Games Recommender System combines collaborative filtering and content-based filtering, using techniques like SVD, cosine similarity and Neural Networks with embeddings. This approach shares similarities with these methods in deep learning and collaborative filtering approaches, yet differs with a specific focus on embedding layers for users and games. It combines SVD-based collaborative filtering with content-based methods (cosine similarity), unlike the discussed papers.

3 Dataset and Features

3.1 Datasets Used

Our Steam Games Recommender System utilizes two main datasets from Kaggle:

steam_200k Dataset [25] This dataset contains information from approximately 200,000 Steam users, including the games they own and their playtime.

steam_games Dataset [8] This dataset includes a wide range of information about games available on Steam, such as game names, developers, publishers, tags, genre information, and SteamSpy tags (for categorization and research). It also encompasses data about positive and negative user reviews. Finally, it includes data regarding game price, game owners, release date, and more, which was not used in our application.

3.2 Data Preprocessing and Feature Extraction

3.2.1 steam_200k Dataset Preprocessing

Our initial handling involved adding column names that were missing by default and removing a useless column that was filled with zeros. Following this, we focused on data filtering, where we created a separate dataset, specifically for information about purchased games, while the main dataset

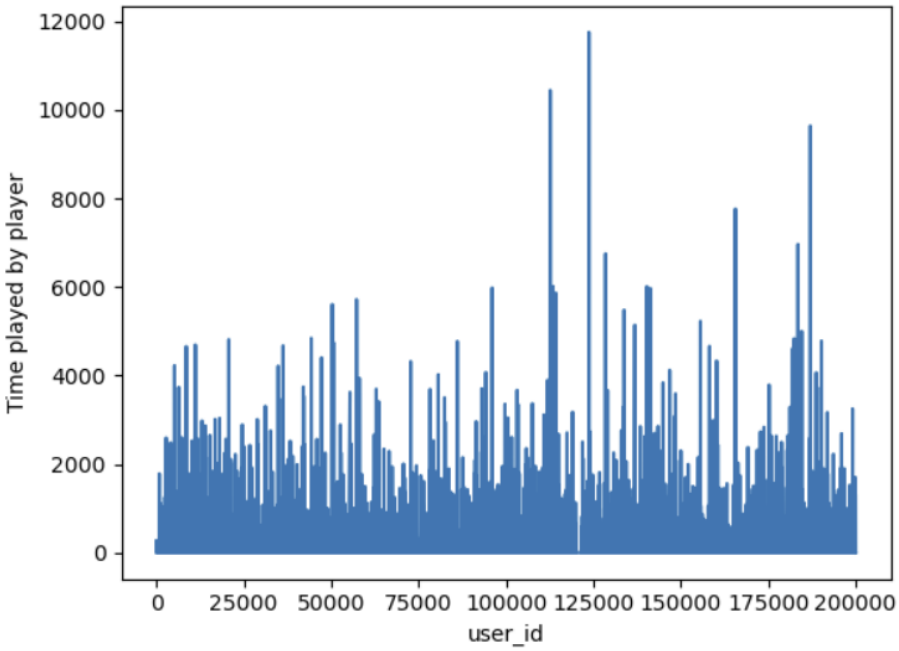

Figure 1: steam_200k [25] data before removing outliers

was filtered to include only playtime data. For outlier removal, we identified and removed data points representing unusually high playtime to ensure smoother data. We used the IQR (Interquartile Range) method. The IQR method is a statistical technique used to identify outliers in a dataset. The IQR is calculated as the difference between the 75th percentile (Q3) and the 25th percentile (Q1) of the data. Outliers are typically identified as observations that fall below Lower Bound $= Q1 - 1.5 \times$ IQR or above Upper Bound $= Q3 + 1.5 \times$ IQR [22]. We utilized the SciPy library [26] to apply IQR. Figure 1 demonstrates the data before removing outliers, and Figure 2 demonstrates it after removing outliers. Lastly, we applied normalization to the playtime data using Min-Max Scaling, aiming to standardize all numerical values to a consistent scale. Min-Max scaling normalization is a linear transformation technique that adjusts data to fit within a predefined boundary. The formula for Min-Max normalization is given as

$$A' = \frac{A - Amin}{Amax - Amin} \times (D - C) + C$$

where A is the original data, $Amin$ and $Amax$ are the minimum and maximum values of A respectively, C and D are the predefined boundaries (usually 0 and 1), and A' is the normalized data. This technique ensures that the normalized values fall within the specified range, maintaining the relationship among the original data values [19]. We utilized the Scikit-learn library [20] to apply Min-Max scaling.

3.2.2 steam_games Dataset Preprocessing

The dataset was already clean, with no duplicate or empty entries, so there was no need for further cleaning. There is no numerical data to check for outliers or normalize (tags are text features). Since positive and negative ratings are numerical but used as a weight for each game, normalization was deemed unnecessary. For the tags text feature, we employed TF-IDF Vectorization, which is suitable when we have natural language content, like descriptions or - in our case - tags. The tags column was tokenized (since tags were separated by ";") and then converted into text format for applying TF-IDF Vectorization. We utilized the Scikit-learn library [20] to apply it.

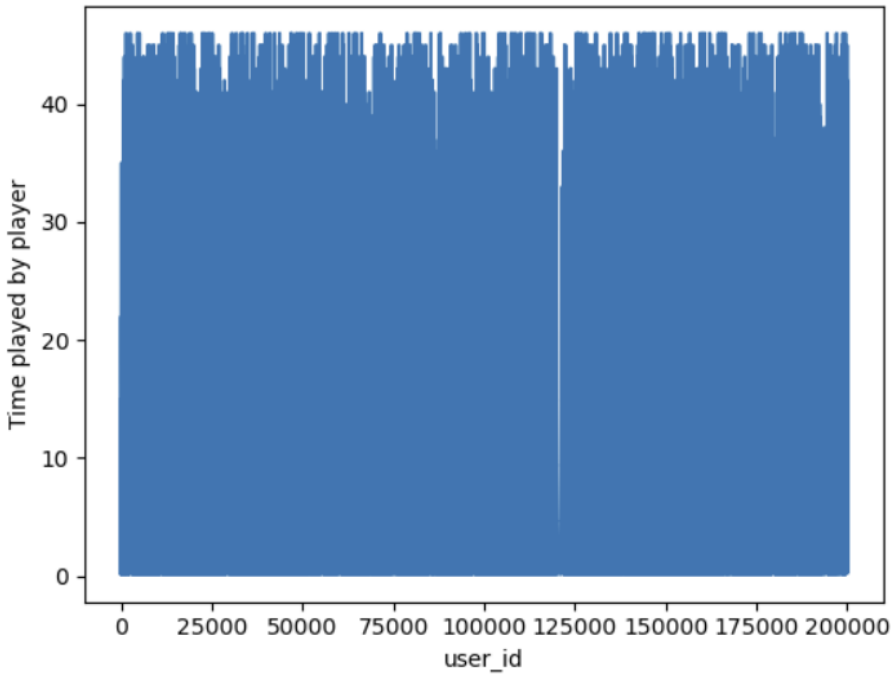

Figure 2: steam_200k [25] data after removing outliers

3.3 Features Used

From the steam_200k [25] dataset, we used user playtime as a critical feature for collaborative filtering and for recommending games based on what users have played the most. From the steam_games [8] dataset, we extracted the steamspy game tags and publisher information. These were used for content-based filtering, where recommendations are based on the similarity of game tags and publishers, or for collaborative filtering, combining user data from steam_200k. The tags associated with each game were transformed into a numerical form using TF-IDF Vectorization. TF-IDF Vectorization identifies important words in a document by combining Term Frequency (TF) (the count of a word in a document) with Inverse Document Frequency (IDF), which measures the rarity of the word across all documents. The TF-IDF score is calculated using the formula

$$\text{TF-IDF} = TF \times log(\frac{total\ number\ of\ documents}{number\ of\ documents\ containing\ the word\ +\ 1})$$

where TF is the frequency of the word in the document and IDF is the logarithm of the total number of documents divided by the number of documents containing the word, plus one for smoothing. This method emphasizes words that are common in individual documents but not often found in all the texts together [13]. This allowed us to quantify the similarity between different games based on their tags. Finally, we kept the negative and positive reviews columns, which were also utilized to introduce a weight to our system.

4 Methods

In our Recommender System, we have utilized three different approaches to offer personalized game suggestions for the users. The approaches use different aspects of the data provided and different algorithms to meet different user needs. These approaches are designed to guarantee that the suggestions focus on not only relevance but also the quality and type of games. In this section, every one of these approaches will be discussed in detail along with the algorithms used, their implementation and the reason why they were selected. The approaches are: 1) Recommending Games Based on What Other Users Play, 2) Recommending Games Based on Similarity and 3) Recommending Games from the Same Publisher.

4.1 Recommending Games Based on What Other Users Play

4.1.1 Singular Value Decomposition (SVD) for Matrix Factorization

Singular Value Decomposition (SVD) represents a method to decompose a real (or complex) matrix, involving a sequence of a rotation, a scaling, and another rotation.This technique generalizes the concept of eigendecomposition of square normal matrices with orthonormal eigenbases, to any $m \times n$ matrix. SVD is also related to polar decomposition. Specifically, for an $m \times n$ real matrix A, the singular value decomposition takes the form

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

Here, U is an $m \times n$ real unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix containing non-negative real numbers on its diagonal, V is an $n \times n$ real unitary matrix, and V^T denotes the transpose of V [15]. In the context of our recommender system, A represents the user-item playtime matrix.

We chose not to split the dataset into training and testing sets, since SVD in our implementation is directly applied for matrix factorization, without being a part of a broader machine learning model requiring evaluation. To recommend games based on what other users have played the most, we constructed a user-item matrix from the playtime data. This matrix represents the hours each user has played each game, with NaN values for games not played by a user. We chose to fill this NaN data with zeros, since it represents that a user has never played or owned the game. We then applied SVD to it using the SciPy library [26] and, after decomposition, we reconstructed the matrix using the calculated U, Σ, and V^T matrices to approximate the original user-item interactions, allowing for game recommendations based on these interactions.

4.1.2 Neural Networks with Embeddings for Collaborative Filtering

We also made an attempt to use Neural Networks (NN) with embeddings to facilitate game recommendations based on what other users play. In neural networks, embeddings are a technique of representation learning that converts the categorical variables like words or entities into continuous vector spaces. Embeddings are especially very important when analyzing the discrete and also high-dimensional data, as they allow one to easily infer the semantic connections between different entities [9]. We utilized the clean and normalized steam_200k [25] dataset. The dataset was then converted to categorical codes for both user IDs and game titles. We split the data into training and testing sets, allocating 20% for testing purposes. We designed the model architecture with Embedding layers for both users and games, which we then flattened and concatenated. We employed a dense layer [11] followed by a dropout layer [24], followed by another dense layer, before the output layer. The model was compiled with the Adam optimizer and mean squared error loss function. The Adam optimizer is an efficient stochastic optimization method that requires only first-order gradients and has minimal memory requirements. It calculates adaptive learning rates for each parameter by estimating the first and second moments of the gradients. Adam combines features of AdaGrad, which is effective with sparse gradients, and RMSProp, suitable for online and non-stationary settings. It maintains update magnitudes invariant to gradient rescaling, has step sizes bounded by a hyperparameter, and works well with sparse gradients and non-stationary objectives [14]. The library we utilized for this approach is Tensorflow Keras [5].

4.2 Recommending Games Based on Similarity

4.2.1 Cosine Similarity for Item-Based Filtering

Cosine similarity is a measure of similarity between two non-zero vectors defined in an inner product space. Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. It is calculated as

$$\frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|}$$

making it a suitable metric for comparing the similarity of items based on their features, in our case comparing games based on tags [12]. We computed the cosine similarity between games based on their TF-IDF vectors of tags. This was done using the Scikit-learn library [20]. To refine

our recommendations, we calculated the ratio of positive to negative reviews for each game and downweighted games with a predominance of negative reviews. This ensured that the recommended games are both similar and of good quality.

4.3 Recommending Games from the Same Publisher

4.3.1 Cosine Similarity for Publisher-Based Recommendations

We applied TF-IDF Vectorization to the publishers' names and then calculated the cosine similarity between the resulting vectors. Like with game tags, we weighted the cosine similarity based on the ratio of positive to negative reviews, downweighting games with predominantly negative reviews. This way, the users get recommendations of good quality games from the publisher of the game they input.

5 Experiments/Results/Discussion

5.1 Setup and Hyperparameters

5.1.1 Singular Value Decomposition (SVD)

We chose to apply SVD with k=2 latent features to balance between capturing significant user-item interactions and maintaining computational efficiency. The decision was based on initial experiments showing that this number of features sufficiently captures the core interactions in the playtime data.

5.1.2 Cosine Similarity and TF-IDF Vectorization

We did not specify particular hyperparameters for Scikit-learn's TfidfVectorizer [20] beyond defaults, as the default settings provided a balanced representation of the tags. For weighting the cosine similarity based on game reviews, we set the ratio threshold at 1.5, meaning that games with more than 40% negative reviews are considered unfavorably. This was chosen based on our personal preference as to what we want the system to consider a "good" suggestion and based on what the Steam platform considers a "bad" suggestion. The ratings weight (which is applied to games under the ratio threshold) was set to 0.7 to provide a balance between similarity and quality of recommendations.

5.1.3 Combining Methods for User-Specific Recommendations

For user-specific recommendations, we combined all three methods (based on playtime, tags similarity, and publisher similarity). We selected the user's top X games (where X can be specified) by playtime and recommended Y games (where Y can also be specified) using these methods.

5.1.4 Neural Networks with Embeddings

In our model, we set the output dimension of the embeddings to be 100 factors. The input dimensions correspond to the number of users and games in the dataset. We specified 128 dimensions for the dense layer [11] and we used the Rectified Linear Unit (ReLU) activation function. This function is especially used in deep learning models. It is defined as

$$f(x) = \max(0, x)$$

which means it outputs the input directly if it is positive, otherwise, it outputs zero [18]. Furthermore, we added a second dense layer with 64 neurons (half the size of the first dense layer) and also used ReLU activation. A dropout layer [24] with a rate of 0.5 was introduced between the dense layers for regularization. We used linear activation [23] in the output layer. As discussed above, we selected Adam as the optimiser of the model with the Mean Squared Error (MSE) as the loss function. Adam was configured with a learning rate of 0.001. Training the model took 80 epochs with a batch size of 64. We selected 10% of the training set to be used for validation.

5.2 Primary Metrics and Experiments

The primary metric for evaluating the effectiveness of the recommender system was the relevance of the recommendations to the user's preferences, assessed qualitatively through user feedback and the

logical coherence of the recommendations. We assessed the system's performance through qualitative analysis, examining the recommendations for various user profiles and games. The focus was on ensuring that the recommendations were sensible, contextually relevant, of high quality and aligned with user preferences. We also utilized quantitative data to evaluate the performance of our Neural Networks implementation. In our evaluation of both the similarity-based and same publisher-based recommendation systems, we conducted an experiment. This involved the creation of a dummy version of the system, one that did not incorporate the weigh-by-reviews feature. Subsequently, we asked both the weighted and unweighted versions of the system to generate recommendations for the same game. The purpose of this process was to witness and compare the results that each version generated. This comparative analysis was essential in determining the effectiveness and accuracy of the systems, especially to judge if the addition of the weigh-by reviews element resulted in better game recommendation quality.

5.3 Results

5.3.1 SVD Recommendations Based on What Others Play

The SVD approach successfully recommended games based on what other users have played the most. For instance, a user with an ID 236188947 was recommended games like Team Fortress 2, Left 4 Dead 2, and Portal 2, which align with popular choices among Steam users in the dataset.

5.3.2 Cosine Similarity Recommendations Based on Tag Similarity

When recommending games based on tags, the system effectively identified games with similar features. The downweighting of games with negative reviews proved effective, as illustrated by the higher positioning of games with lots of positive reviews in the recommendations. In our experiment, the unweighted system recommended 'Games of Glory', demonstrating its capability to identify genre and style similarities, as it aligns with 'Dota 2', the input game. Both games are categorized as free-to-play, MOBA, and fiction. However, an examination of 'Games of Glory' on the Steam platform reveals that it possesses a substantial proportion of negative user ratings, exceeding 45%. This suggests that while the system successfully identified a similar game, the overall user experience might be negatively impacted due to the game's lower quality. A similar observation was made with 'Heroes of SoulCraft - Arcade MOBA', which, despite matching in similarity, has approximately 60% negative reviews. On the other hand, our weighted implementation did not recommend the two mentioned games and instead prioritized 'Bloodline Champions', a MOBA game with an overwhelmingly positive reception, evidenced by over 85% favorable ratings. Additionally, the system elevated 'Strife®' and 'SMITE®' in the recommendation list, both of which hold a reputable standing within the Steam community. Furthermore, the weighted system recommended two additional games with favorable ratings, reinforcing the effectiveness of our approach in integrating user review data to enhance the quality of game recommendations. This outcome confirms the efficacy of our weighted-by-reviews implementation in delivering not only similar but also highly-regarded game suggestions.

5.3.3 Cosine Similarity Recommendations Based on the Same Publisher

We carried out the same experiment in this case. Both the unweighted and weighted systems were tasked with suggesting games from the same publisher, in this instance, Ubisoft. However, a notable difference emerged as a result of incorporating user ratings into the recommendation process. The key distinction observed was that the unweighted system recommended 'Trials® Rising', a Ubisoft game which, upon review, exhibited only a mediocre rating. In contrast, our weighted system which factors in user reviews, did not include 'Trials® Rising' in its recommendations. Instead, it prioritized 'Beyond Good and Evil™', a game that has received exceptional ratings from users. The introduction of weighting by reviews once again highlighted the efficacy of this approach in delivering more user-satisfactory game suggestions.

5.3.4 Neural Networks Recommendations Based on What Others Play

The training and validation losses observed during the training phase are depicted in Figure 3. We applied the model to the test dataset and collected some metrics. More specifically, we calculated the Test Mean Squared Error (MSE), the Root Mean Squared Error (RMSE) and the R-squared (R^2).

The MSE is the average of the squares of the differences between predicted and actual values. The formula for the MSE is given as

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(X_i - Y_i)^2$$

where m is the number of data points, X_i are the predicted values and Y_i are the expected values [4]. The RMSE is the square root of the MSE and its formula is

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(X_i - Y_i)^2}$$

[4]. Finally, the R-squared (or coefficient of determination) indicates how well the variability of the dependent variable is explained by the independent variables, with a range from $-\infty$ to 1, where 1 indicates a perfect fit. Its formula is

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(X_i - Y_i)^2}{\sum_{i=1}^{m}(Y - Y_i)^2}$$

where Y is the mean of the actual values [4].

The model exhibited a RMSE of 0.0206038, which signifies the standard deviation of the prediction errors. Additionally, we calculated the MSE to be 0.0004245. Finally, the model achieved an R-squared (R2) value of 0.027282. To gain additional insights into the model's performance, we visualized the latent embeddings of the users and games using PCA (Figure 4) and t-SNE (Figure 5), by utilizing the Scikit-learn library [20]. PCA is a linear technique that focuses on keeping the low-dimensional representations of dissimilar data points far apart. For high-dimensional data that lies on or near a low-dimensional nonlinear manifold, it is usually more important to keep the low-dimensional representations of very similar data points close together, which is typically not possible with a linear mapping like PCA. t-SNE is a nonlinear algorithm, which has become the de facto standard for data visualization. At a high level, t-SNE chooses two similarity measures between pairs of points—one for the high-dimensional data and one for the 2-dimensional embedding. It aims to construct a 2-dimensional embedding that minimizes the Kullback-Leibler (KL) divergence between the vector of similarities in the original dataset and the similarities in the embedding. This process involves a non-convex optimization problem, and t-SNE employs gradient descent with random initialization and other techniques like early exaggeration to find a reasonable solution [1].

The PCA plots demonstrated some degree of clustering among both user and game embeddings, suggesting the presence of underlying patterns. The t-SNE plots further highlighted these patterns, with more distinct clusters emerging, thereby providing a deeper understanding of the model's embedding space. This non-linear dimensionality reduction technique revealed more complex structures in the data, which could be leveraged to improve the recommendation system. Finally, we generated some predictions for a user to test functionality. The training and validation loss chart indicates good model learning with stable convergence. However, the test performance metrics reveal that the model's predictive accuracy is modest and explains a small fraction of the variance in the test data. The model may need future improvement to enhance its predictive capabilities. We shall adhere to our initial implementations for recommending games, unless additional time is allocated to improve the NN approach.

5.4 GUI Design and Functionality

To enhance the user experience within our project, we have developed a simple graphical interface that helps the user interact easily with our system, using the ipywidgets library [6]. The interface (Figure 6) presents the user with four distinct input options. Users are given the choice to input either a user ID or a game name to get tailored recommendations. If a game name is defined, the interface automatically disables the user ID input field. Additionally, users can choose the quantity of game recommendations they want across the three mentioned categories (what other users play, similarity, same developer), as well as the number of a user's favorite games that should be considered in generating these recommendations. The input for selecting the number of favorite games is disabled when a game name is provided. Upon clicking the 'Run Recommender' button, the system executes and delivers a list of suggested games to the user.

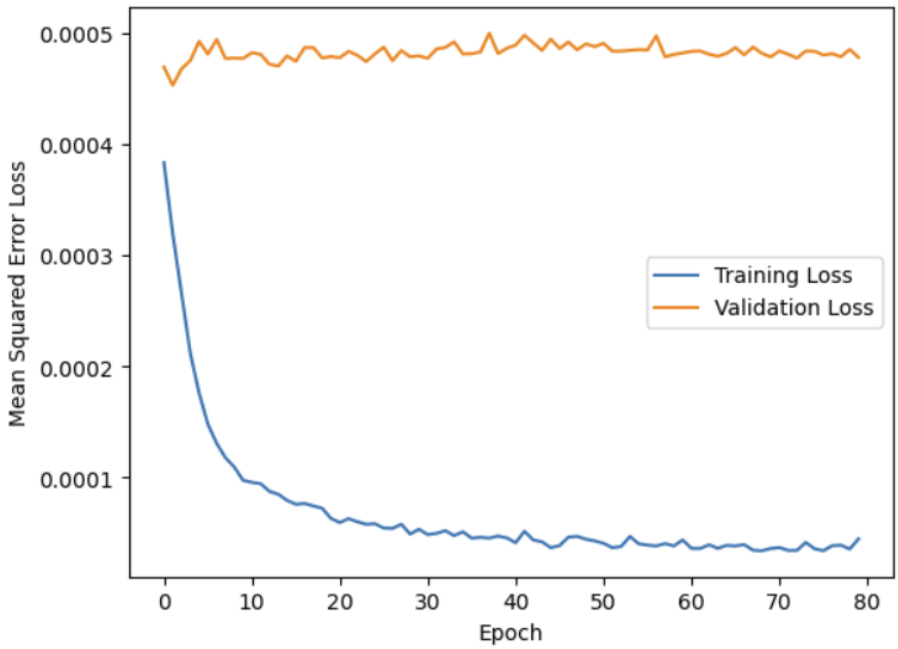

Figure 3: Neural Networks with Embeddings training and validation loss

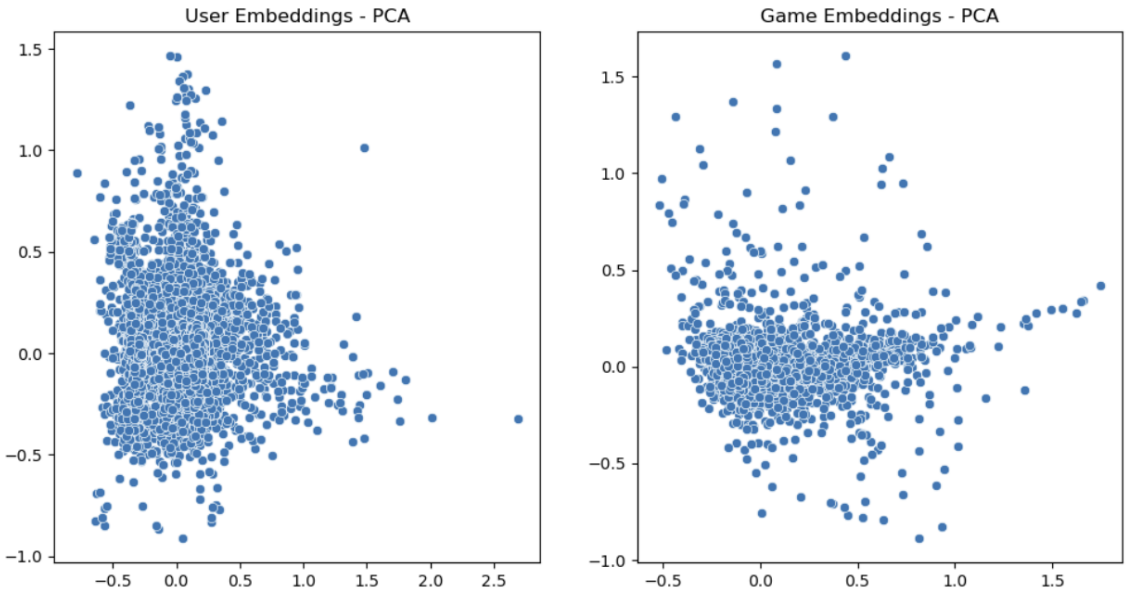

Figure 4: PCA plots of user and game embeddings

5.5 Discussion

The qualitative analysis demonstrated the system's capability to make sensible and relevant recommendations based on user playtime, game tags, and publishers. The inclusion of review-based weighting further refined the quality of recommendations, ensuring that highly-rated games were prioritized. There were no significant concerns about overfitting, as the system primarily relied on unsupervised methods like SVD and cosine similarity, which do not involve explicit training phases. The choice of hyperparameters, like the number of latent features in SVD and the threshold for reviews in cosine similarity, was guided by a balance between relevance and generalization.

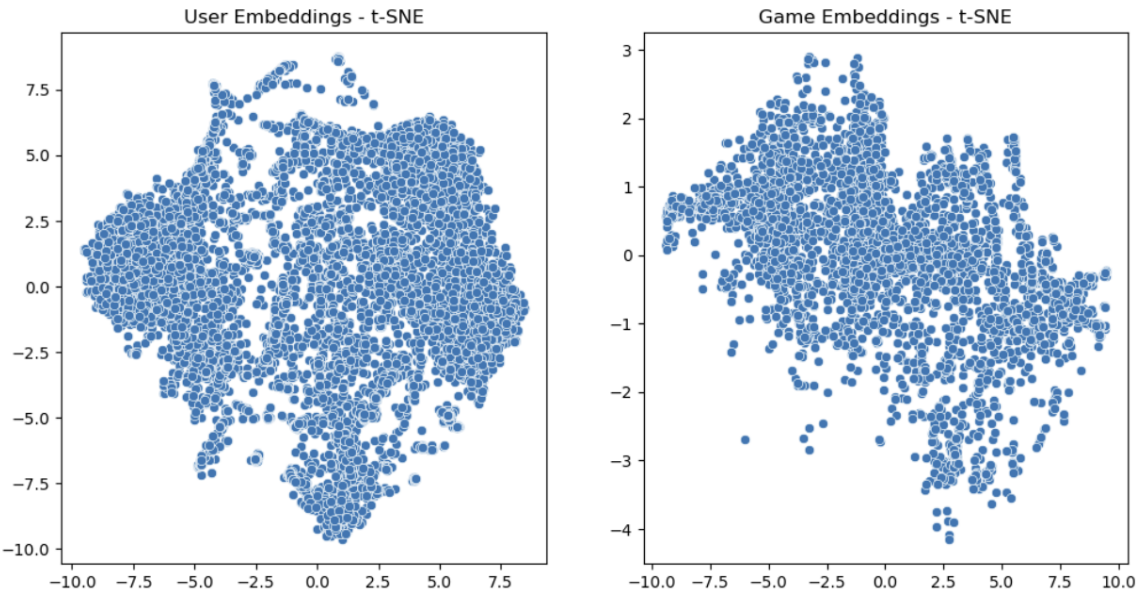

Figure 5: t-SNE plots of user and game embeddings

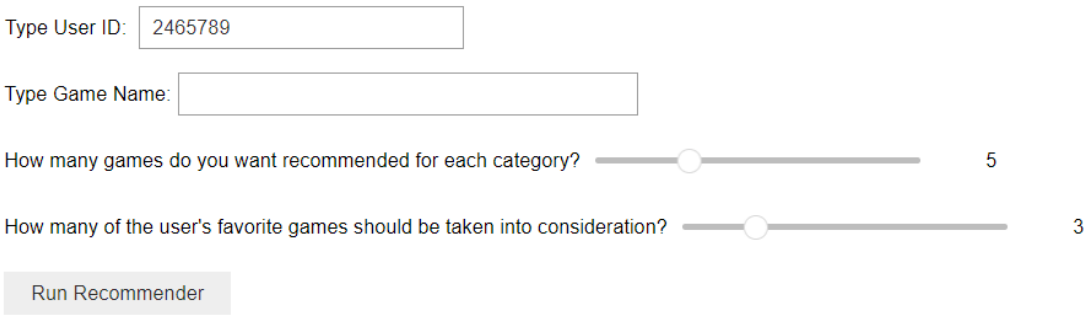

Figure 6: Graphical User Interface of the recommender system

Alongside these techniques, we explored the use of a Neural Networks (NN) approach with embeddings to recommend games by analyzing user behavior patterns. The quantitative analysis of this model suggested that while it learned and converged well, its predictive accuracy requires further enhancement. Further improvements could involve experimenting with different numbers of latent features in SVD, incorporating more sophisticated natural language processing techniques for tag analysis, and exploring additional user data, such as explicit game ratings, if available. Moreover, we aim to refine the NN approach, possibly by adjusting the architecture or incorporating additional data, to improve recommendation quality.

6 Contributions

Each team member contributed equally to the project, bringing their skills to develop an effective steam games recommender system. Our collaboration and combined efforts resulted in a system that can accurately recommend games to users based on a variety of factors, enhancing the user experience on the Steam platform. In the following sections we outline the contributions of each team member to the project.

6.1 Thomas Katraouras

Thomas mainly worked on developing the recommendation algorithms. He worked on implementing the methods to recommend games based on user preferences, similarities in game features, and what other users are playing. More specifically, he developed the reviews-weighted cosine similarity solution and deployed the SVD solution, along with Vasilis. Thomas also contributed to the integration of these algorithms into the recommender system, ensuring that they work effectively and efficiently to provide accurate game recommendations. He also worked on the Neural Networks implementation of the recommender.

6.2 Vasileios Giosis

Vasilis focused on the dataset preprocessing and the initial setup of the recommender system. He was responsible for cleaning and organizing the data, ensuring that it was suitable for analysis. He developed the code for handling and manipulating the datasets, which included removing unnecessary data, normalizing the data, and preparing it for the recommendation algorithms. He also assisted in deploying SVD for the first dataset and in deploying the Neural Networks implementation.

6.3 Konstantinos Kokkalis

Konstantinos was tasked with researching on the many relevant academic papers. He had to review these papers and combine their content into a simple interpretation for the benefit of his fellow team members. Konstantinos was also responsible for the user interface and user experience design of the recommender system. He developed an interactive, user-friendly GUI that allows users to interact with the system and receive game recommendations. Additionally, Konstantinos worked on the backend integration, ensuring that the GUI communicates correctly with the recommendation algorithms and displays the results in an intuitive and accessible manner. Lastly, he assisted Thomas and Vasilis in assessing the efficacy of the recommendation algorithms by providing feedback on the relevance of the recommendations to users, based on his user experience design.

References

[1] Sanjeev Arora, Wei Hu, and Pravesh K. Kothari. An Analysis of the t-SNE Algorithm for Data Visualization, June 2018. arXiv:1803.01768 [cs].

[2] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, Corvalis Oregon USA, June 2007. ACM.

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10, Boston MA USA, September 2016. ACM.

[4] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Computer Science*, 7:e623, July 2021.

[5] François Chollet and others. Keras, 2015.

[6] Jupyter widgets community. ipywidgets, a GitHub repository. 2015.

[7] Paul Covington, Jay Adams, and Emre Sargin. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198, Boston Massachusetts USA, September 2016. ACM.

[8] NIK DAVIS. Steam Store Games (Clean dataset).

[9] Cheng Guo and Felix Berkhahn. Entity Embeddings of Categorical Variables, April 2016. arXiv:1604.06737 [cs].

[10] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Pisa, Italy, December 2008. IEEE.

[11] Gao Huang, Zhuang Liu, Geoff Pleiss, Laurens van der Maaten, and Kilian Q. Weinberger. Convolutional Networks with Dense Connectivity, January 2020. arXiv:2001.02394 [cs, stat].

[12] Inderprastha Engineering College, AKTU, Ramni Harbir Singh, Sargam Maurya, Inderprastha Engineering College, AKTU, Tanisha Tripathi, Inderprastha Engineering College, AKTU, Tushar Narula, Inderprastha Engineering College, AKTU, Gaurav Srivastav, and Inderprastha Engineering College, AKTU. Movie Recommendation System using Cosine Similarity and KNN. *International Journal of Engineering and Advanced Technology*, 9(5):556–559, June 2020.

[13] Donghwa Kim, Deokseong Seo, Suhyoun Cho, and Pilsung Kang. Multi-co-training for document classification using various document representations: TF–IDF, LDA, and Doc2Vec. *Information Sciences*, 477:15–29, March 2019.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[15] V. Klema and A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control*, 25(2):164–176, April 1980.

[16] Yehuda Koren. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, August 2009.

[18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation Functions: Comparison of trends in Practice and Research for Deep Learning, November 2018. arXiv:1811.03378 [cs].

[19] S.Gopal Krishna Patro and Kishore Kumar Sahu. Normalization: A Preprocessing Stage. *IARJSET*, pages 20–22, March 2015.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. AutoRec: Autoencoders Meet Collaborative Filtering. In *Proceedings of the 24th International Conference on World Wide Web*, pages 111–112, Florence Italy, May 2015. ACM.

[22] Songwon Seo. A Review and Comparison of Methods for Detecting Outliers in Univariate Data Sets.

[23] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology*, 04(12):310–316, May 2020.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

[25] TAMBER. Steam Video Games.

[26] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.