UNIVERSITY OF INFORMATION TECHNOLOGY

CS114 - Machine Learning

Games Recommender System

Group: 11
Tran Xuan Thanh - 21520456
Mai Anh Quan - 21520411
Nguyen Ha Anh Vu - 21520531

Lecturer:
Pham Nguyen Truong An
Le Dinh Duy

UIT, Sunday 28th January, 2024

SUMMARY

Github: https://github.com/maqnitude/games-recommender

Collect data from Steam:

- Use `selenium` to collect `app_id` of the games.
- Use Steam Web API and the collected `app_id` to continue collecting the games data and user reviews data.

Build the recommender:

- Collaborative filtering using matrix factorization:
  - Create `pseudo_ratings` by combining different data from user review (e.g., `voted_up`, `playtime_forever`, etc.) to have something similar to `ratings` in the Movielens dataset, and create a feedback matrix using this `pseudo_ratings`.
  - The feedback matrix is decomposed into the the product of two lower-dimensional matrices: `user_embeddings` and `game_embeddings`. These embedding matrices represent the latent (hidden) features of users and games, and the number of laten features is determined by `embedding_dim`.
- Content-based filtering using a softmax model:
  - **Input**: The input to the model is a feature vector x, representing the list of games a user has reviewed. The selected features include `app_id`, `categories`, `genres`, `year`.
  - **Model training**: The model learns to create game embeddings by minimizing the softmax cross-entropy loss across all users. During training, the model chooses a random game from the list of reviewed games of a user as a label.
  - **Output**: The output of the model is a set of scores for each game, indicating the likelihood that a user will be interested in that game. The model is trained to predict this based on the user's history of reviewed games.
  - **Recommendations**: Recommendations are made by computing similarity scores between all games with the user history of games, then suggesting the games with highest aggregated scores.

We didn't evaluate our game recommender system because we didn't know the appropriate way to do it. Instead, we employed a qualitative approach of evaluation by gathering feedbacks from team members on the recommendations. We believe that whether a recommender system is good or bad depends on how the users respond to the recommendations. To understand this, we would need to see if the user clicks and views the game, adds it to their wish-list, or buys it, etc. the kind of data that is impossible for us to collect. There are other ways to evaluate a recommender system like using predictive accuracy to see how close the estimated feedback is to the original, but we didn't think it was the appropriate way to evaluate a recommender system, so we didn't use it.

Contents

0 Updates

No updates.

1 Introduction

1.1 Motivation

In today's technological era, the use of recommendation systems has become increasingly prevalent, providing personalized and optimized experiences for users. In the realm of electronic entertainment, particularly electronic games, recommender systems play a crucial role in helping players discover new and suitable gaming experiences tailored to their individual preferences.

The games recommender system problem poses the challenge of accurately suggesting engaging games based on a player's gaming history, previous ratings, and other personal factors. This necessitates a flexible combination of machine learning techniques, data mining, and deep insights into the player's personality and preferences.

In this context, the games recommender system problem not only aims to optimize user experiences but also plays a significant role in enhancing the connection between players and entertainment content, creating a personalized and ever-expanding space for entertainment.

1.2 Input & Output

Input:

- Games data (name, categories, genres, developers, publisher, etc.)
- User explicit/implicit feedback data (like or dislike, ratings, playtime, etc.)
- User data (age, location, games owned, etc.). This is optional, because it is illegal to collect user data without their permission.

Output:

- Game recommendations: A ranked list of items recommended for a particular user. This list may be accompanied by confidence or similarity scores or explanations for why these items are chosen.

2 Data Collection

Our collected data can be found here.

We first started scraping games from the Steam website, specifically their `app_id`, by using `selenium` then saved them to a text file. We then utilized these `app_id`s and the Steam Web API (`https://partner.steamgames.com/doc/webapi_overview`) to collect the games data and user reviews data for each game.

To scrape the `app_id`s, we tried to scrape as many games as we could in the top rated section of each of these categories:

```
categories = [
    'action', 'arcade_rhythm', 'fighting_martial_arts', 'action_fps',
    'hack_and_slash', 'action_run_jump', 'action_tps', 'shmup',

    'rpg', 'rpg_action', 'rpg_jrpg', 'rpg_party_based',
    'rogue_like_rogue_lite', 'rpg_turn_based',

    'strategy', 'strategy_card_board', 'strategy_cities_settlements',
    'strategy_grand_4x', 'strategy_military', 'strategy_real_time',
    'tower_defense', 'strategy_turn_based',

    'adventure', 'adventure_rpg', 'casual', 'hidden_object', 'metroidvania',
    'puzzle_matching', 'story_rich', 'visual_novel',

    'simulation', 'sim_building_automation', 'sim_dating',
    'sim_farming_crafting', 'sim_hobby_sim', 'sim_life',
    'sim_physics_sandbox', 'sim_space_flight',

    'sports_and_racing', 'sports', 'sports_fishing_hunting',
    'sports_individual', 'racing', 'racing_sim', 'sports_sim', 'sports_team',
]
```

But, we only managed to scraped 4586 games after more than 6 hours. This could be due to unexpected connection issues while contacting the Steam server.

To collect the games data, we used this API, which fortunately does not require Steam API key:

`https://store.steampowered.com/api/appdetails?appids={app_id}&lang=en`

And created the `csv` file called `games.csv` with the following columns:

```
GAMES_COLUMNS = [
    'app_id', 'name', 'required_age', 'is_free', 'developers', 'publishers',
    'platforms_windows', 'platforms_mac', 'platforms_linux', 'metacritic',
    'categories', 'genres', 'recommendations', 'coming_soon', 'release_date',
    'review_score', 'review_score_desc', 'total_positive', 'total_negative',
    'total_reviews',
]
```

To collect reviews data, we used this API, which also does not require Steam API key:

`https://store.steampowered.com/appreviews/{app_id}?`
`json=1&language=all&num_per_page=100&cursor={cursor}`

As you can see, we collected reviews in batches and each batch has at most 100 reviews per game. Cursor is an indicator that points to the next batch of reviews. We only managed to collect about 260K reviews, given the fact that Steam has rate limiting and we were collecting using multiple processes so some data could not be fully retrieved. For an in-depth view of our dataset's statistics, you can refer to this notebook.

3 Background

3.1 Recommendation System Overview

A typical recommendation systems comprises the following components:

- Candidate generation: This initial stage selects a small subset of items from a large corpus based on the user query.
- Scoring and ranking: This stage evaluates and orders the candidates based on a precise model that can use additional features.
- Re-ranking: This final stage applies additional constraints and criteria to the ranking, such as diversity, freshness, and fairness.

Two common approaches to candidate generation are:

- Collaborative Filtering: Uses similarities between queries and items simultaneously to provide recommendations.
- Content-based Filtering: Uses similarity between items to recommend items similar to what the user likes.

In this final project, our primary focus is on candidate generation as we believe it is a crucial step in making recommendations. For the collaborative filtering method, we utilize Matrix Factorization. For the content-based approach, we implement a softmax model.

3.2 Matrix Factorization

Matrix Factorization is a class of collaborative filtering algorithms used in recommendation systems. The idea behind matrix factorization is to represent users and items in a lower dimensional latent space. The user-item matrix is then reconstructed by taking the dot product of these two matrices. The predicted ratings can be computed as the dot product of the user's latent factors and the item's latent factors.

Matrix factorization is a simple embedding model. Given the feedback matrix $A \in R^{m \times n}$, where m is the number of users and n is the number of items, the model learns:

- A user embedding matrix $U \in \mathbb{R}^{m \times d}$, where row i is the embedding for user i.
- An item embedding matrix $V \in \mathbb{R}^{n \times d}$, where row j is the embedding for item j.

The embeddings are learned such that the product UV^T is a good approximation of the feedback matrix A. Observe that the (i, j) entry of $U \cdot V^T$ is simply the dot product $\langle U_i, V_j \rangle$ of the embeddings of user i and item j, which you want to be close to $A_{i,j}$.

3.2.1 Objective Function

The objective function for matrix factorization can be defined as a squared distance objective. This objective function aims to minimize the sum of squared errors over all pairs of observed entries:

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2. \tag{1}$$

In this objective function, you only sum over observed pairs (i, j), that is, over non-zero values in the feedback matrix. However, only summing over values of one is not a good idea—a matrix

of all ones will have a minimal loss and produce a model that can't make effective recommendations and that generalizes poorly.

Instead, we can treat the unobserved values as zero, and sum over all entries in the matrix. This corresponding to minimizing the squared Frobenius distance between A and its approximation UV^T:

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} ||A - UV^T||_F^2. \tag{2}$$

We can solve this quadratic problem through **Singular value decomposition (SVD)** of the matrix. However, SVD is not a great solution either, because in real life applications, the matrix A can be very sparse. This can lead to poor generalization performance as the solution UV^T will likely be close to zero.

There is also a variant known as **Weighted matrix factorization**, which decomposes the objective into the following sums:

- A sum over observed entries
- A sum over unobserved entries (treated as zeros).

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} w_{i,j}(A_{ij} - \langle U_i, V_j \rangle)^2 + w_0 \sum_{(i,j) \notin \text{obs}} (\langle U_i, V_j \rangle)^2. \tag{3}$$

Here, $w_{i,j}$ is the weight for observed entries and w_0 is the weight for unobserved entries.

The model adds a regularization term to penalize large embeddings for unobserved entries, and weights the observed entries by their frequency. The weight $w_{i,j}$, a function of the frequency of query i and item j, prevents the dominance of frequent items or queries. While the hyperparameter w_0 balances the two terms in the objective function.

Common algorithms used to minimize these objective functions include Stochastic Gradient Descent (SGD) and Weighted Alternating Least Squares (WALS). These methods iteratively update the user and item embeddings to minimize the objective function.

3.2.2 Regularizaiton in Matrix Factorization

Regularization is crucial to prevent a phenomenon known as "folding", where the model fails to correctly position the embeddings of irrelevant items.

Two types of regularization are commonly employed:

1. **Model Parameter Regularization**: This involves a standard l_2 regularization term on the embedding matrices, mathematically represented as:

$$r(U, V) = \frac{1}{N} \sum_i ||U_i||^2 + \frac{1}{M} \sum_j ||V_j||^2. \tag{4}$$

2. **Gravity term**: This is a global prior that nudges the prediction of any pair towards zero. It is given by:

$$g(U, V) = \frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} \langle U_i, V_j \rangle^2. \tag{5}$$

The total loss function, which includes these regularization terms, is then given by:

$$\frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} (A_{ij} - \langle U_i, V_j \rangle)^2 + \lambda_r r(U, V) + \lambda_g g(U, V) \tag{6}$$

where λ_r and λ_g are regularization hyper-parameters.

These regularization techniques help in effectively managing the placement of embeddings, thereby enhancing the performance of the matrix factorization model in recommender systems.

Matrix factorization can be significantly more compact than learning the full matrix, especially in real-world recommendation systems where the matrix may be very sparse. It also allows for the use of additional features beyond the query ID/item ID, which can help capture the specific interests of a user and improve the relevance of recommendations.

3.3 Softmax model

Some limitations of matrix factorization include the difficulty of using side features and the relevance of recommendations. The Softmax model is a deep neural network (DNN) model that can address the limitations of matrix factorization. It treats the recommendation problem as a multiclass prediction problem in which:

- The input is user query
- The output is a probability vector with size equal to the number of items in the corpus, representing the probability to interact with each item

3.3.1 Input

The input to a softmax model can include both dense features and sparse features. Unlike the matrix factorization approach, we can add side features for user such as age, gender or country.

3.3.2 Model architecture

The model architecture determines the complexity and expressivity of the model. By adding hidden layers and non-linear activation functions (for example, ReLU), the model can capture more complex relationships in the data. However, increasing the number of parameters also typically makes the model harder to train and more expensive to serve.

3.3.3 Softmax Output

We can denote the input vector as x and the output of the last hidden layer by $\psi(x)$. The model maps the output of the last hidden layer, $\psi(x)$, through a softmax layer to a probability distribution $\hat{p} = h(\psi(x)V^T)$, where:

- h: $\mathbb{R}^n \longleftrightarrow \mathbb{R}^n$ is the softmax function, given by $h(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$
- $V \in \mathbb{R}^{n\times d}$ is the matrix of weights of the softmax layer.

The softmax layer maps a vector of scores $y \in \mathbb{R}^n$ to a probability distribution.

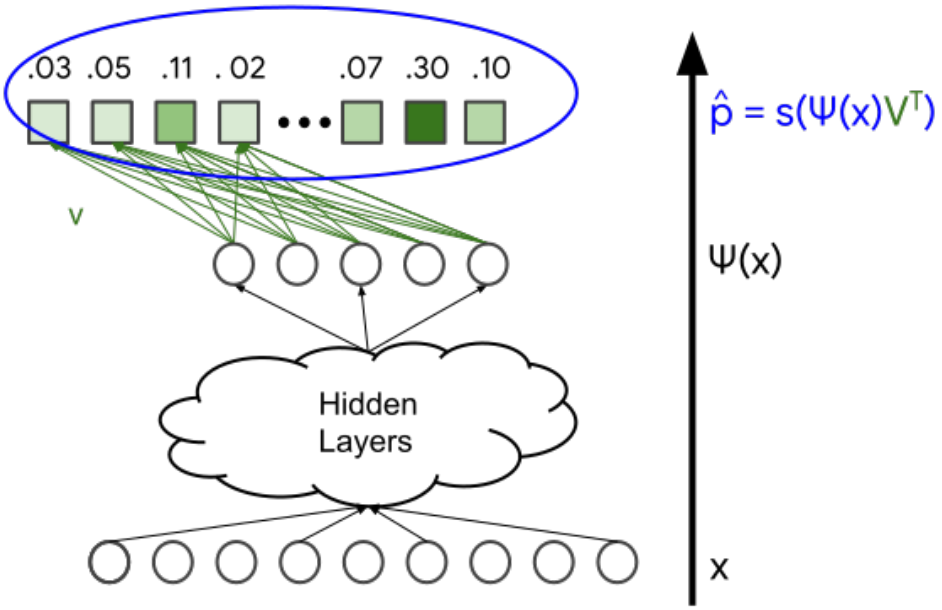


Figure 1: Visualization of the softmax model architecture

3.3.4 Loss function

A loss function aims to compare the following:

- $\hat{p}$, the output of the softmax layer (a probability distribution)
- p, the ground truth, representing the items the user has interacted with. This can be represented as a normalized multi-hot distribution (a probability vector).

The softmax model is trained using cross-entropy loss, which encourages the model to assign higher probabilities to items that the user interacts with.

3.3.5 Softmax Embeddings

The probability of item j is given by $\hat{p_j} = \frac{\exp(\langle \psi(x), V_j \rangle)}{Z}$, where Z is a normalization constant that does not depend on j. In other words, $\log(\hat{p_j}) = \langle \psi(x), V_j \rangle - \log(Z)$, so the log probability of an item j is (up to an additive constant) the dot product of two d-dimensional vectors, which can be interpreted as query and item embeddings:

- $\psi(x) \in \mathbb{R}^d$ is the output of the last hidden layer. We call it the embedding of the query x.
- $V_j \in \mathbb{R}^d$ is the vector of weights connecting the last hidden layer to output j. We call it the embedding of item j.

The model maps the output of the last layer through a softmax layer to a probability distribution. The softmax function normalizes the model predictions over all items as a probability distribution. It then maximizes the probability of the observed items as compared with that of the unobserved items.

The Softmax model can easily incorporate query features and item features due to the flexibility of the input layer of the network. This can help capture the specific interests of a user and improve the relevance of recommendations. It also allows for the use of additional features such as age or country.

4 Methodology

4.1 Collaborative Filtering with Matrix Factorization

4.1.1 Preprocessing

First, we merged the games dataframe (4586 rows × 20 columns) to the reviews data (261184 rows × 19 columns) on `app_id` to create one big reviews dataframe (261184 rows × 38 columns). Next, we applied min-max normalization to the following columns:

```
cols_to_normalize = [
    'num_games_owned', 'num_reviews',
    'playtime_forever', 'playtime_last_two_weeks', 'playtime_at_review',
    'votes_up', 'votes_funny', 'comment_count',
    'review_score', 'total_positive', 'total_negative', 'total_reviews'
]
```

After that, we applied one-hot encoding to the `categories` and `genres` so that later on we could utilize these features. We could also have applied one-hot encoding to the `developers` and `publishers` columns, but we did not have enough RAM.

The final step that we did was to map `steam_id` and `app_id` according to their row indices. This was to make it easier when working with the user embeddings matrix and the game embeddings matrix.

4.1.2 Pseudo-ratings

In the Movielens dataset, which is a dataset for movies recommendation, they have explicit user feedback data, that is, `ratings` ranging from 1 to 5. In our dataset, we do have the explicit feedback data but it's in binary, that is, positive or negative vote (represented by the boolean data `voted_up`). But, we also have other implicit data like the total playtime on the games the users reviewed.

Matrix factorization requires a feedback matrix, but using only the explicit feeback data `voted_up` is not sufficient because it's only binary. We needed something that is similar to the `ratings` in the Movielens dataset, and came to the solution of creating a new feedback data for our dataset called `pseudo_ratings`. The `pseudo_ratings` is calculated by combining other implicit feedback and interaction data as a weighted sum. We separated the features that we wanted to incorporate into the pseudo-ratings:

- The normal features with the weight distribution as follows:

```
weights = {
    'weighted_vote_score': 0.5,
    'playtime_forever': 0.2,
    'voted_up': 0.1,
    'num_reviews': 0.05,
```

```
    'total_reviews': 0.05,
    'total_positive': 0.05,
    'total_negative': 0.05,
}
```

- The interaction features, which are combinations of multiple normal features, with the weight distribution:

```
interaction_weights = {
    'voted_up_total_positive_negative_reviews': 0.25,
    'playtime_voted_up': 0.25,
    'playtime_category': 0.25,
    'playtime_genre': 0.25,
}
```

Each of these set of weights is accounted for 50% of the total weights used to calculate the pseudo-ratings.

4.1.3 Build the model

Because we collected more than 4500 games and over 200,000 unique users who reviewed the games, we would not have enough RAM to create the feedback matrix. Given the fact that most of the users only reviewed 1 games, the feedback matrix would be very sparse. That's why we used a special datatype called a `SparseTensor`.

Once we converted the dataframe into the a `SparseTensor` representing the feeback matrix, we could then use `TensorFlow` to build and train the model.

4.1.4 Train the model

We split the data to create the testing data by randomly sampling 10% of the data, the other 90% would be our training data. For the loss function, we used mean squared error (MSE), and stochastic gradient descent (SGD) as the optimizer.

The model was trained in 2000 iterations, with the following settings:

- `embedding_dim = 70`.
- `init_stddev = 0.5`.
- `learning_rate = 10.0`

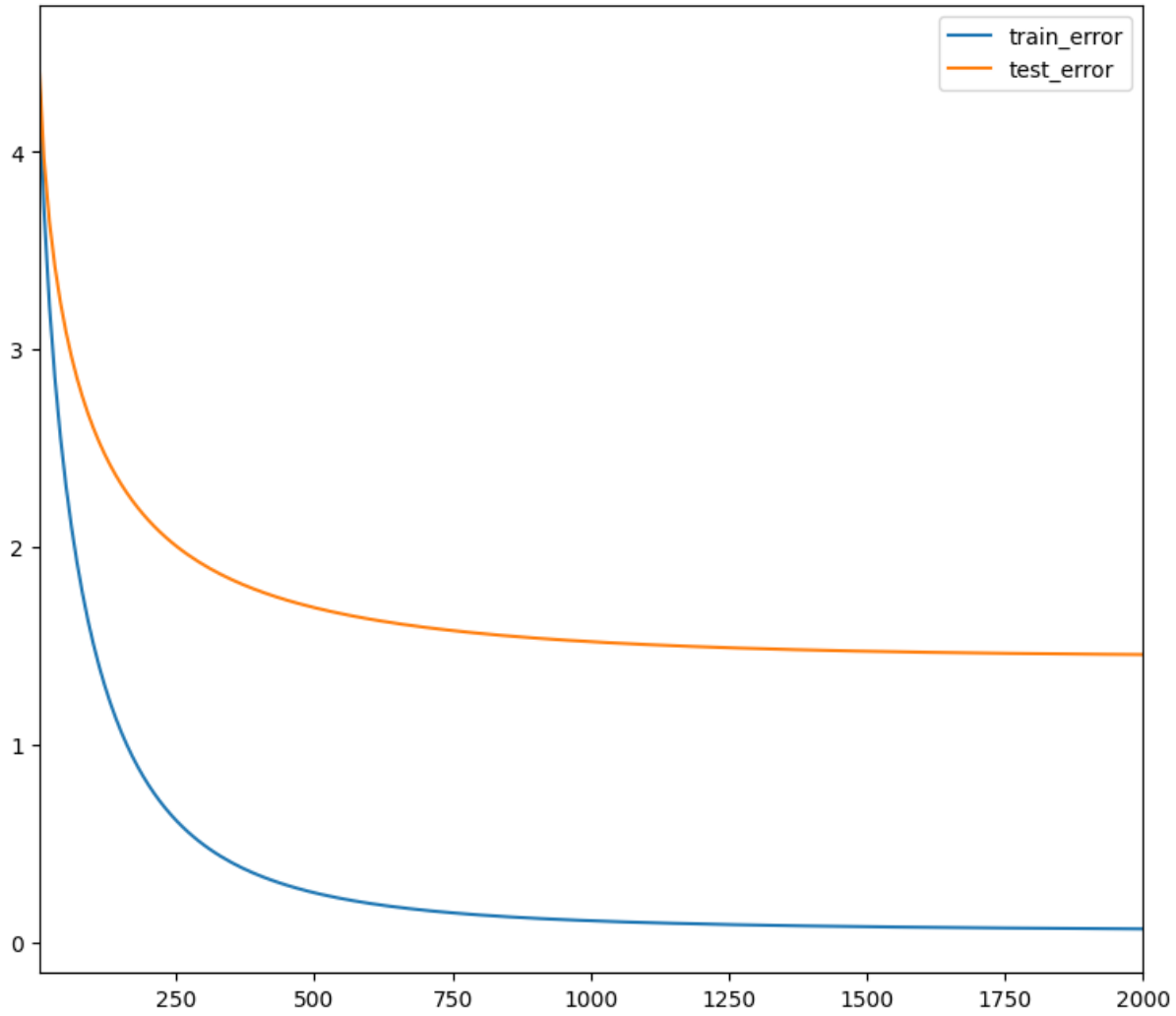


Figure 2: Unregularized: The train_error and test_error graph of the training process of matrix factorization model without regularization. The horizontal axis represents the number of iterations, and the vertical axis represents the value of the loss function.

4.1.5 Train the model with regularization

The regularized version of the model was also trained in 2000 iterations, with the following settings:

- `embedding_dim = 80.`
- `init_stddev = 0.5.`
- `learning_rate = 10.0.`

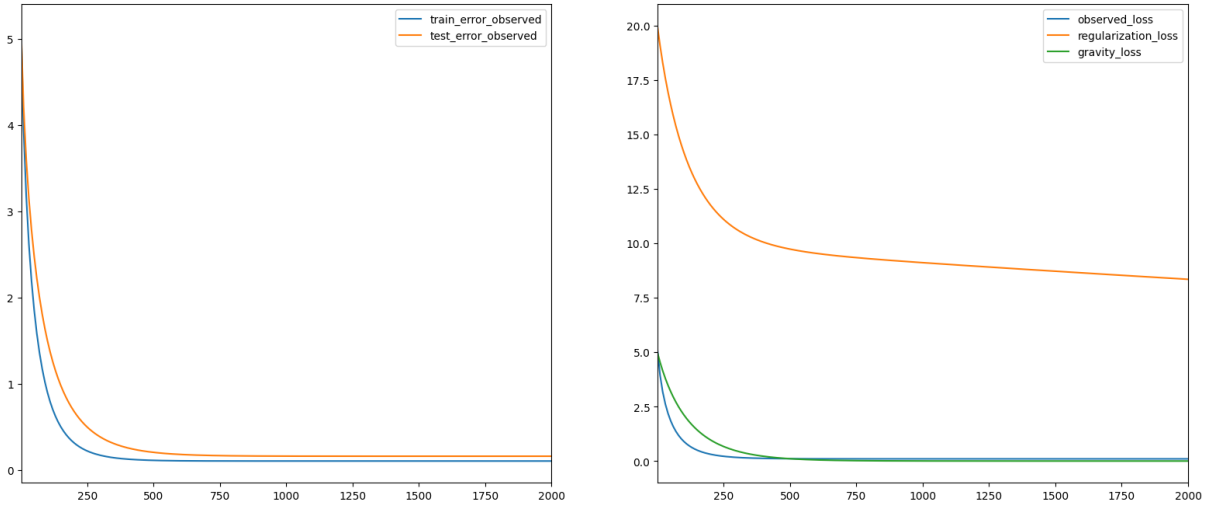


Figure 3: Regularized: The left graph shows the observed train_error and test_error through out the iterations. The right graph shows the observed_loss, the regularization_loss, and the gravity_loss through out the iterations.

4.2 Softmax Model

4.2.1 Preprocessing

The model takes as input a feature vector x, representing the list of games a user has reviewed. Initially, we created a dataframe containing the user id (`steam_id`) and their corresponding reviewed games by grouping the reviews dataframe by `steam_id`. Subsequently, we reindexed the user and game ids to start from zero and ensure sequential values. To manage limited RAM resources, we reduced the dataset size by selecting only the top N users who interacted with the most games. We believe that users who interact with more games provide a richer resource for the model to learn from.

After preprocessing and selecting the top $N = 15000$ users based on their interactions with games, our final dataset for training the softmax model included 15000 users, 3660 games, and 42670 reviews.

4.2.2 Model Building

The feature chosen for the model are `app_id`, `categories`, `genres`, and `year`. These features were selected because they provide important information about the games. The `app_id` represents the unique identifier of a game, `categories` and `genres` provide information about the type of the game, and `year`, release year of the game, gives temporal information.

We then defined a function that generates batches of examples for training model. It takes a DataFrame of rated games and a batch size as input. The function first pads the features to ensure they have the same length, then creates a TensorFlow dataset from the padded features. This function is crucial for training the model efficiently, as feeding the model with batches of examples allows for parallel computation, which greatly speeds up the training process.

We built a softmax model that maps input features to user embeddings. The model takes a DataFrame of training examples, a dictionary mapping feature names to embedding column

objects and a list of the dimensions of the hidden layers. For each sparse feature, a bag-of-words embedding is created. The embeddings are then passed through a series of hidden layers to generate user embeddings. The item embeddings are learned as part of the `app_id` feature column.

4.2.3 Training

The model is trained by minimizing the softmax loss, which measures how well the model's predictions match the actual user-item interactions. The user embeddings and items embeddings are learned such that the embeddings of a user and the items they have interactions with are closed together in the embedding space.

For the evaluation of our model, we partitioned the data into a training set and a test set, with 90% of the data used for training and the remaining 10% held out for testing. We set the batch sizes for the training and testing data to be 200 and 100, respectively.

During training phase, we tuned the following hyperparameters:

- learning rate
- number of iterations.
- Input embedding dimensions (specified by the `input_dims` argument)
- Number and size of hidden layers (specified by the `hidden_dims` argument)

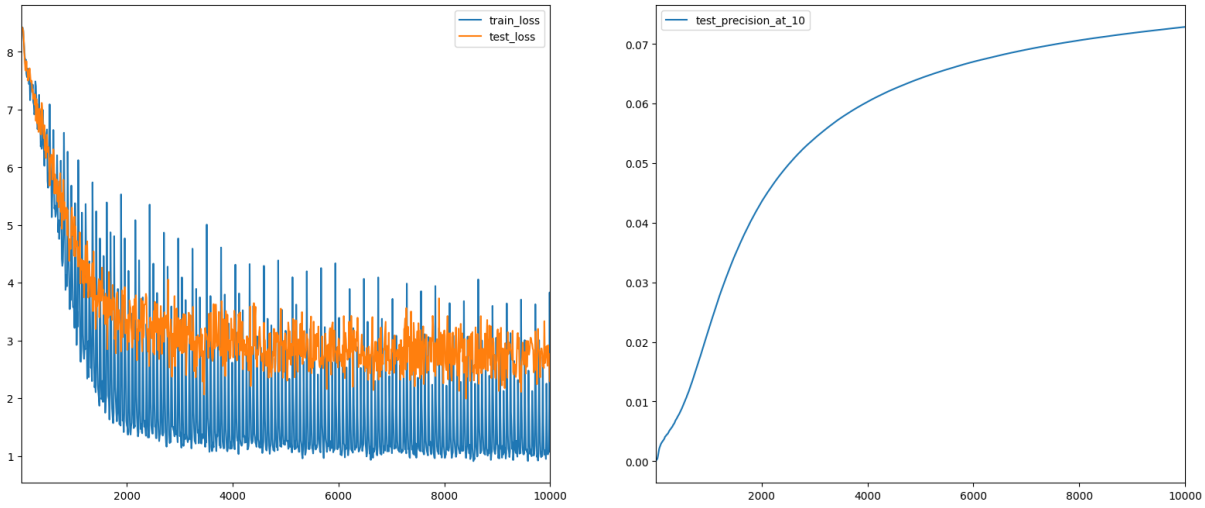


Figure 4: The left graph illustrates the training and testing loss over the iterations. The right graph illustrates the precision at 10 score for the test set. (learning rate = 3, iterations = 10000)

We experimented with various hyperparameter settings during model training and recorded the corresponding results.

Learning Rate	Iterations	Input embedding dim	No. & Size of Hidden Layers	Train Loss	Test Loss
5.	3000	40	[30]	1.414753	2.967012
8.	2000	40	[30]	1.456499	3.539693
3.	6000	40	[30]	1.011149	2.953995
8.	4000	45	[35]	1.905518	3.109182
3.	10000	40	[30]	1.871787	2.287838

Table 1: Some hyperparameter settings we've experimented

5 Testing the Models

5.1 Matrix Factorization Model

We trained the matrix factorization model with high embedding dimension (70, and 80 for the regularized version of the model) to capture as many latent features as we can without overfitting the model, so we thought it might be a good idea to apply a dimensionality reduction technique like principle component analysis (PCA) to reduce the embedding dimension of the user embeddings matrix and the game embeddings matrix by half and compare the results.

The following recommendation results are created using the matrix factorization model with regularization.

Nearest neighbors of "Counter-Striker 2" based on the dot and cosine similarity scores:

```
game = "Counter-Strike 2"
game_neighbors(reg_model.embeddings['app_id'], game, DOT)
game_neighbors(reg_model.embeddings['app_id'], game, COSINE)
```

```
--NORMAL--
```

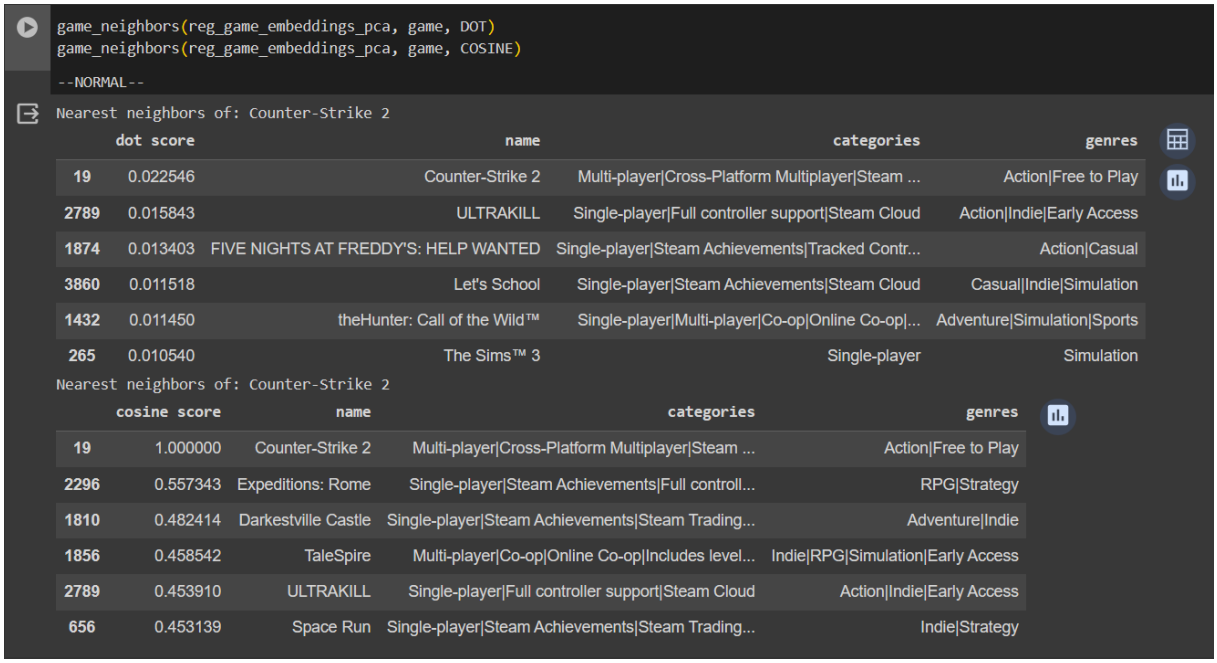
```
Nearest neighbors of: Counter-Strike 2
```

	dot score	name	categories	genres
19	0.037913	Counter-Strike 2	Multi-player|Cross-Platform Multiplayer|Steam ...	Action|Free to Play
1874	0.018450	FIVE NIGHTS AT FREDDY'S: HELP WANTED	Single-player|Steam Achievements|Tracked Contr...	Action|Casual
3788	0.016161	Wandering Sword	Single-player|Partial Controller Support|Steam...	Adventure|Indie|RPG|Strategy
2729	0.015894	Resident Evil Village	Single-player|Steam Achievements|Full controll...	Action
1432	0.015389	theHunter: Call of the Wild™	Single-player|Multi-player|Co-op|Online Co-op|...	Adventure|Simulation|Sports
2123	0.014105	Resident Evil 2	Single-player|Steam Achievements|Full controll...	Action

```
Nearest neighbors of: Counter-Strike 2
```

	cosine score	name	categories	genres
19	1.000000	Counter-Strike 2	Multi-player|Cross-Platform Multiplayer|Steam ...	Action|Free to Play
1874	0.350996	FIVE NIGHTS AT FREDDY'S: HELP WANTED	Single-player|Steam Achievements|Tracked Contr...	Action|Casual
602	0.342428	Crimsonland	Single-player|Multi-player|Shared/Split Screen...	Action|Indie|RPG
3788	0.337860	Wandering Sword	Single-player|Partial Controller Support|Steam...	Adventure|Indie|RPG|Strategy
2624	0.332537	Showgunners	Single-player|Steam Achievements|Full controll...	NaN
1479	0.318901	The Legend of Heroes: Trails of Cold Steel	Single-player|Steam Achievements|Steam Trading...	RPG

Figure 5: Nearest neighbors of "Counter-Strike 2" using the matrix factorization model with regularization.



```
game_neighbors(reg_game_embeddings_pca, game, DOT)
game_neighbors(reg_game_embeddings_pca, game, COSINE)
--NORMAL--
```

Nearest neighbors of: Counter-Strike 2

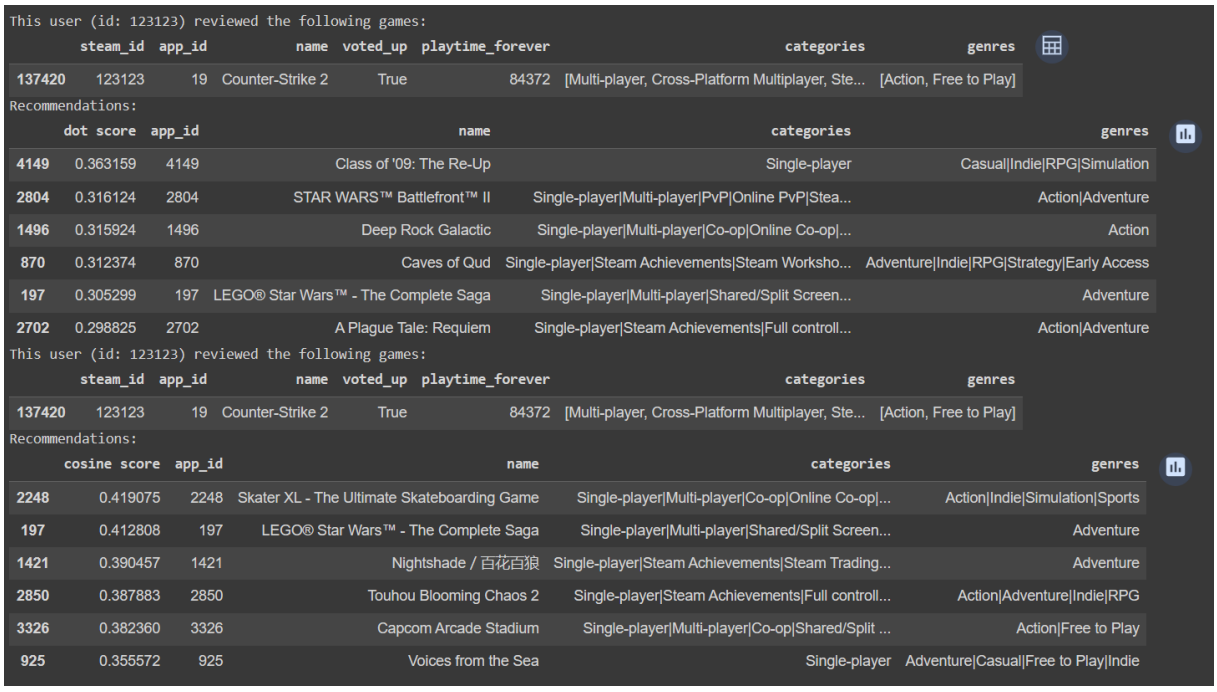
	dot score	name	categories	genres
19	0.022546	Counter-Strike 2	Multi-player|Cross-Platform Multiplayer|Steam ...	Action|Free to Play
2789	0.015843	ULTRAKILL	Single-player|Full controller support|Steam Cloud	Action|Indie|Early Access
1874	0.013403	FIVE NIGHTS AT FREDDY'S: HELP WANTED	Single-player|Steam Achievements|Tracked Contr...	Action|Casual
3860	0.011518	Let's School	Single-player|Steam Achievements|Steam Cloud	Casual|Indie|Simulation
1432	0.011450	theHunter: Call of the Wild™	Single-player|Multi-player|Co-op|Online Co-op|...	Adventure|Simulation|Sports
265	0.010540	The Sims™ 3	Single-player	Simulation

Nearest neighbors of: Counter-Strike 2

	cosine score	name	categories	genres
19	1.000000	Counter-Strike 2	Multi-player|Cross-Platform Multiplayer|Steam ...	Action|Free to Play
2296	0.557343	Expeditions: Rome	Single-player|Steam Achievements|Full controll...	RPG|Strategy
1810	0.482414	Darkestville Castle	Single-player|Steam Achievements|Steam Trading...	Adventure|Indie
1856	0.458542	TaleSpire	Multi-player|Co-op|Online Co-op|Includes level...	Indie|RPG|Simulation|Early Access
2789	0.453910	ULTRAKILL	Single-player|Full controller support|Steam Cloud	Action|Indie|Early Access
656	0.453139	Space Run	Single-player|Steam Achievements|Steam Trading...	Indie|Strategy

Figure 6: Nearest neighbors of "Counter-Strike 2" from the matrix factorization model with regularization after applying PCA to the game embeddings matrix.

Recommend games for user with `id = 123123` based on the dot and cosine similarity scores:



This user (id: 123123) reviewed the following games:

	steam_id	app_id	name	voted_up	playtime_forever	categories	genres
137420	123123	19	Counter-Strike 2	True	84372	[Multi-player, Cross-Platform Multiplayer, Ste...	[Action, Free to Play]

Recommendations:

	dot score	app_id	name	categories	genres
4149	0.363159	4149	Class of '09: The Re-Up	Single-player	Casual|Indie|RPG|Simulation
2804	0.316124	2804	STAR WARS™ Battlefront™ II	Single-player|Multi-player|PvP|Online PvP|Stea...	Action|Adventure
1496	0.315924	1496	Deep Rock Galactic	Single-player|Multi-player|Co-op|Online Co-op|...	Action
870	0.312374	870	Caves of Qud	Single-player|Steam Achievements|Steam Worksho...	Adventure|Indie|RPG|Strategy|Early Access
197	0.305299	197	LEGO® Star Wars™ - The Complete Saga	Single-player|Multi-player|Shared/Split Screen...	Adventure
2702	0.298825	2702	A Plague Tale: Requiem	Single-player|Steam Achievements|Full controll...	Action|Adventure

This user (id: 123123) reviewed the following games:

	steam_id	app_id	name	voted_up	playtime_forever	categories	genres
137420	123123	19	Counter-Strike 2	True	84372	[Multi-player, Cross-Platform Multiplayer, Ste...	[Action, Free to Play]

Recommendations:

	cosine score	app_id	name	categories	genres
2248	0.419075	2248	Skater XL - The Ultimate Skateboarding Game	Single-player|Multi-player|Co-op|Online Co-op|...	Action|Indie|Simulation|Sports
197	0.412808	197	LEGO® Star Wars™ - The Complete Saga	Single-player|Multi-player|Shared/Split Screen...	Adventure
1421	0.390457	1421	Nightshade / 百花百狼	Single-player|Steam Achievements|Steam Trading...	Adventure
2850	0.387883	2850	Touhou Blooming Chaos 2	Single-player|Steam Achievements|Full controll...	Action|Adventure|Indie|RPG
3326	0.382360	3326	Capcom Arcade Stadium	Single-player|Multi-player|Co-op|Shared/Split ...	Action|Free to Play
925	0.355572	925	Voices from the Sea	Single-player	Adventure|Casual|Free to Play|Indie

Figure 7: Recommendations for user using the matrix factorization model with regularization.

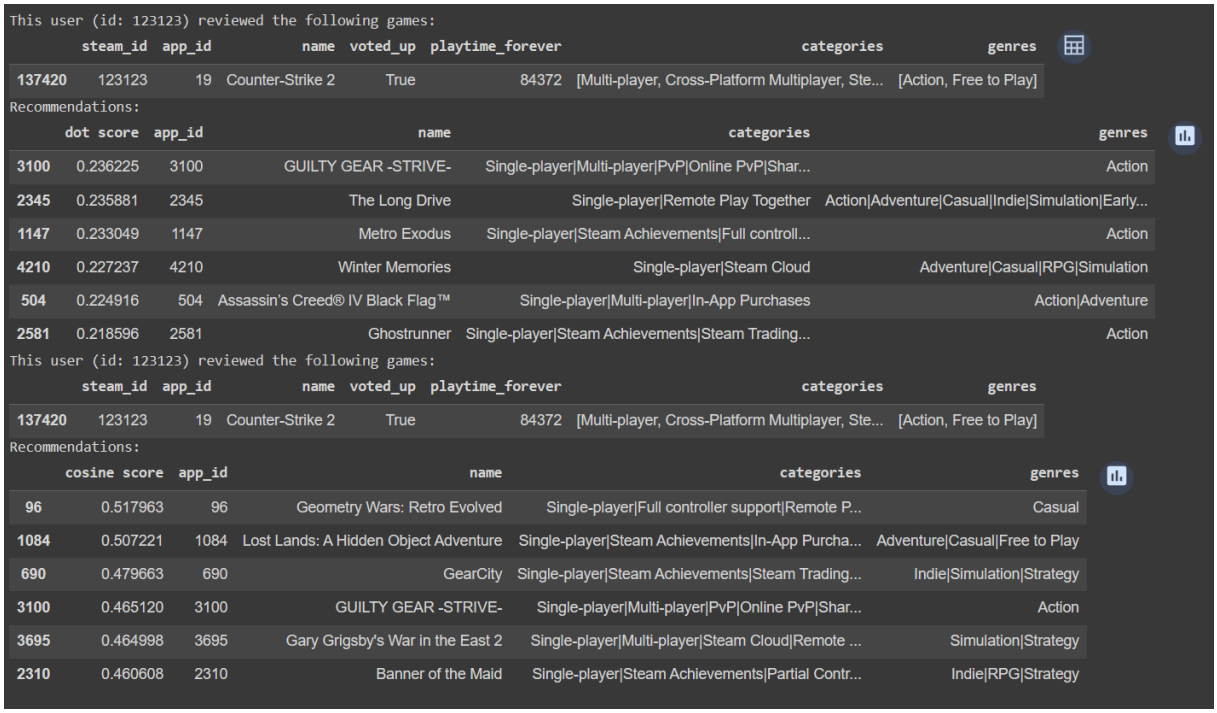


Figure 8: Recommendations for user using the matrix factorization model with regularization after applying PCA to the user embeddings matrix and the game embeddings matrix.

5.2 Softmax Model

To find games that are similar or "neighbors" to a given game, we can compute a similarity measure between the embedding of the given game and the embeddings of all other games. Then take the top ones with highest similarity scores.

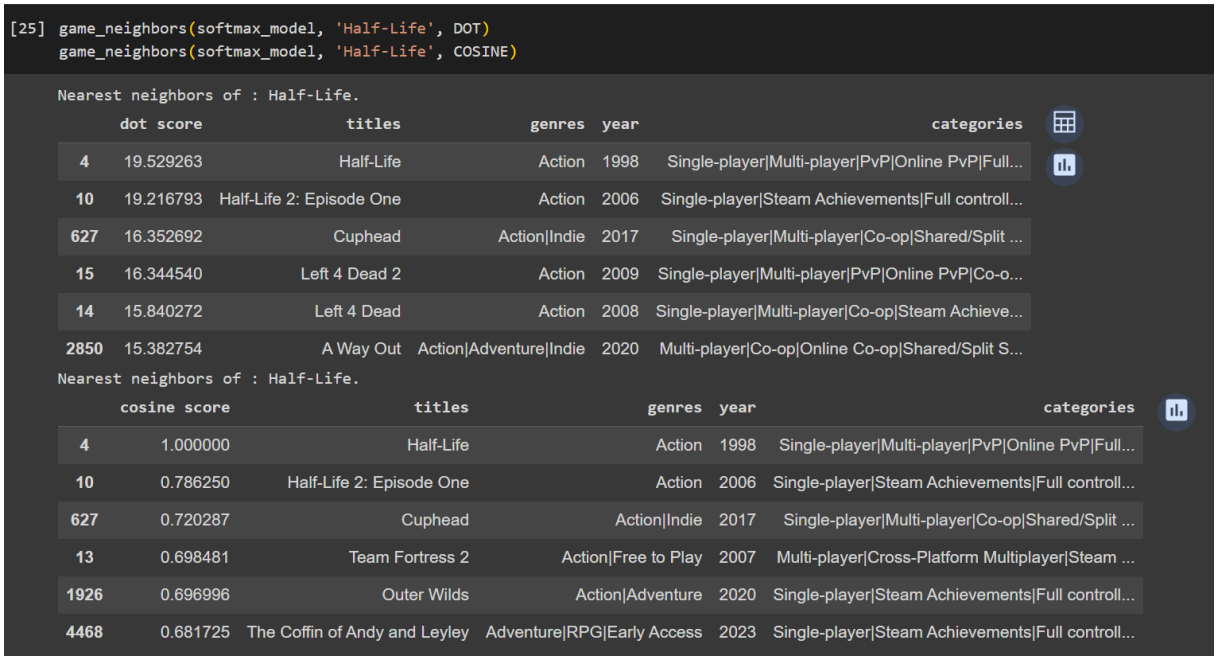


Figure 9: Nearest game neighbors of "Half Life" using the softmax model

To generate personalized recommendations for a specific user, we employ a method that involves iterating through the list of games they have reviewed. For each game in this list, we compute a similarity score in relation to all other games within our dataset. The final score assigned to each game is calculated as the average of these similarity scores. Subsequently, we present the top 'k' games, ranked by their respective scores, as our recommendations.

Additionally, we can refine our recommendation process by excluding certain games from consideration. For instance, we can disregard games from the user's reviewed list that have been voted down or not recommended by the user. Similarly, games with low playtime can also be excluded. Consequently, our recommendations would only include games that are similar to those remaining in the user's reviewed list.

```
[29] user_recommendations(softmax_model, '20263', DOT, exclude_rated=True)
```

Games that user 20263 has reviewed:

	app_id	name	categories	genres	year
3059	3059	Cult of the Lamb	Single-player|Steam Achievements|Full controll...	Action|Adventure|Indie|Strategy	2022
3104	3104	The Cosmic Wheel Sisterhood	Single-player|Steam Achievements|Full controll...	Adventure|Indie	2023
3154	3154	The Room 4: Old Sins	Single-player|Steam Achievements|Steam Trading...	Adventure	2021
3307	3307	Sun Haven	Single-player|Multi-player|Co-op|Online Co-op|...	Adventure|Casual|Indie|RPG|Simulation	2023

Recommendations:

	dot score	app_id	name	categories	genres	year
1151	13.747642	1151	Stardew Valley	Single-player|Multi-player|Co-op|Online Co-op|...	Indie|RPG|Simulation	2016
2109	13.293730	2109	Little Nightmares II	Single-player|Steam Achievements|Full controll...	Adventure	2021
4143	13.037323	4143	Jusant	Single-player|Steam Achievements|Full controll...	Action|Adventure|Indie	2023
4128	12.723573	4128	A Building Full of Cats	Single-player|Steam Achievements|Steam Trading...	Casual	2022
1347	12.392584	1347	Human Fall Flat	Single-player|Multi-player|PvP|Online PvP|Co-o...	Adventure|Casual|Indie|Simulation	2016
1292	12.129722	1292	The Room Three	Single-player|Steam Achievements|Steam Trading...	Adventure|Indie	2018

Figure 10: Recommendations for user with steam_id 20263 using the softmax model

6 References

- https://developers.google.com/machine-learning/recommendation
- https://github.com/recommenders-team/recommenders
- https://www.tensorflow.org/api_docs/python/tf/compat/v1