PROJECT REPORT

Steam Game Recommender System
CMPE 255: Data Mining

Submitted By:
Mithil Ashish Parmar
014726023

1. Introduction

Project Option: Application Track

Steam Game library consists of a huge number of games which the user can buy and play. Steam recommends games to you based on your previous bought games and doesn't take into consideration if you have liked the game or not. The recommendation is done based on the genre of the game and games with similar genre are recommended. To get more accurate and meaningful recommendations, we create a new recommendation system which makes the use of useful features like the number of hours spent on a particular game. This recommendation system will be implemented on a web application where a user can select the games they have played for and the approximate number of hours they have spent on the game to get the top 5 steam game recommendations.

2. Dataset Description

The dataset for our model is created by Tamber and is readily available on Kaggle. The dataset is comprised of four columns that consist of the user-id, game-name, behavior (play/purchase) and hours-played. For *purchase* behavior, the value of hours played is always 1. This is a huge dataset with unnecessary information for our model. We need to pre-process the data so that we just have the required information for our model.

During pre-processing, following checks and changes were made to the dataset:

- Changed *userid* to *string* type because it considered it as an *integer* and gave false information like the higher the number, the better.
- Checked for null values but found none.
- Checked the number of unique games which was 5155.
- Checked the number of unique users which was 12393.
- Removed the rows with *behavior* as *purchase*.
- Created and saved a new dataframe for *games* and assigned them *game_index*.
- Created and saved a new clean dataframe for the complete preprocessed dataset with *game_index* added as well as column *like* that has binary values(1 if played more than 40 hours and 0 otherwise).

```
df_clean.head()
```

	userid	game	behavior	hoursplayed	like	game_index
0	151603712	The Elder Scrolls V Skyrim	play	273.0	1	0
1	59945701	The Elder Scrolls V Skyrim	play	58.0	1	0
2	53875128	The Elder Scrolls V Skyrim	play	0.0	0	0
3	92107940	The Elder Scrolls V Skyrim	play	110.0	1	0
4	250006052	The Elder Scrolls V Skyrim	play	465.0	1	0

Fig 1. Cleaned data after pre-processing

3. Model

I have used a Restricted Boltzmann Machine (RBM) as the model for the recommender system. RBM is a generative stochastic neural network that can learn probability distribution over the set of inputs. They generally form a bipartite graph.

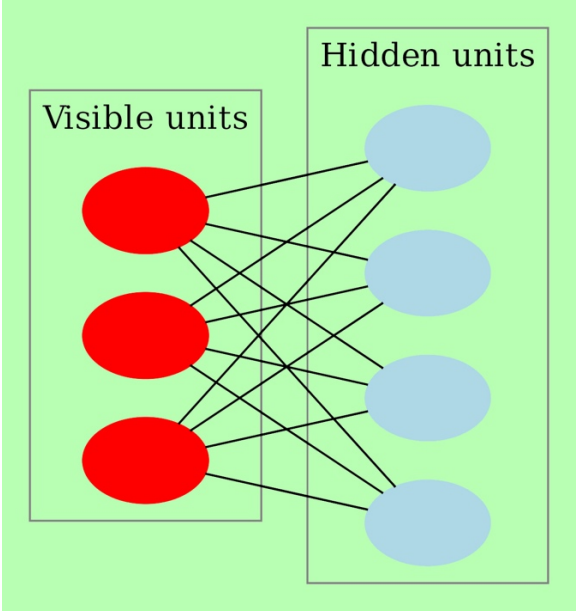

Fig 2. RBM as bipartite graph

RBMs are generative Deep Learning models with only 2 types of nodes: hidden and visible. There is no output node. RBMs from correlations, patterns, and parameters between layers on all data. There are two additional layers of bias: hidden bias and visible bias. Hidden bias produces activation during forward pass and visible bias helps reconstruct input during backward pass.

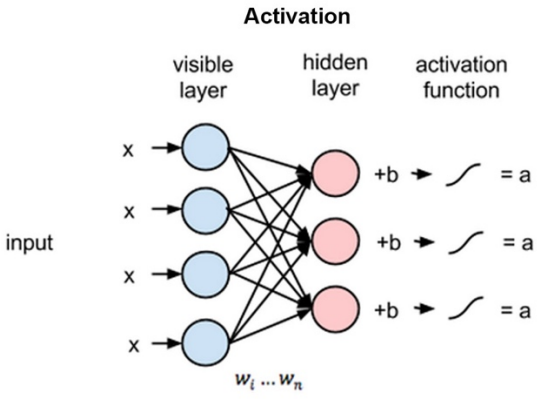

Fig 3. Activation process of RBM

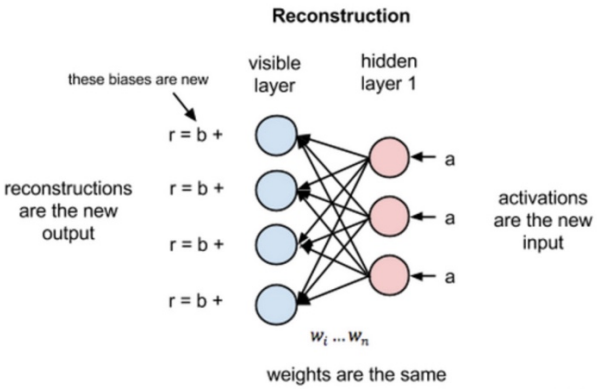

Fig 4. Reconstruction process of RBM

The first step in training RBM is to multiply inputs by weights and add them to the bias. The result is passed through a sigmoid activation function and this output decides if the hidden layer gets activated or not. Weights is a matrix with number of input nodes as number of rows and number of hidden nodes as the number of columns. All the data is in the form of vectors and matrices.

Mathematically, the different functions can be depicted as below:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{1 + e^{x}}$$

Eq. 1. Sigmoid Activation Function

$$h^{(1)} = S(v^{(0)T} W + a)$$

Eq. 2. Activation Phase

$$v^{(1)} = S(h^{(1)} W^{T} + b)$$

Eq. 3. Reconstruction phase

h and **v** are corresponding vectors for the hidden and visible layers with the superscript as the iteration. **W** is the weight vector. **a** is the hidden layer bias vector and **b** is the visible bias vector. ($v^{(0)} - v^{(1)}$) is the reconstruction error which needs to be reduced in subsequent steps in the training process. The weights get adjusted in each iteration to reduce the error which forms the learning process. Finding the probabilities during forward pass and backward pass gives us the joint distribution of the inputs.

For the recommendation for a user, we use a vector with the hours played by the user for the games they have played and pass it to the RBM. We get the recommendation score between 0 and 1 which is the joint distribution of the inputs of the user. Scores are sorted in descending order to display the top 5 games recommended to the user.

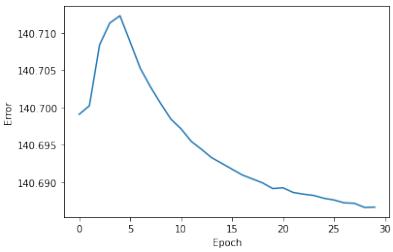

Fig 5. Error for each epoch during training

4. Evaluation

Previously, collaborative filtering was used to get recommendation for a user using item-item similarity matrix. This only recommended games that were similar to the games you have purchased irrespective of whether you liked it or not. This is not a very accurate method to get recommendations for the user. RBM solves this problem and gives customized recommendation to the users by utilizing the *hours_played* data for a particular game played by the user which shows the likeness for the game as well. The user will spend more time on a game they like than the games they don't.

To implement the recommendation system on a web application, Flask was used for backend to handle the API for the recommendation system. The frontend for the web application was done on React.js framework. Using the web application, the user can add as many games as he likes that he has played and also mention approximate hours played on each game to get the recommendations. Validation of data is done before sending it to the recommender system meaning that no duplicate games input is allowed, and it will alert the user to remove the duplication to get the recommendation.

5. Results

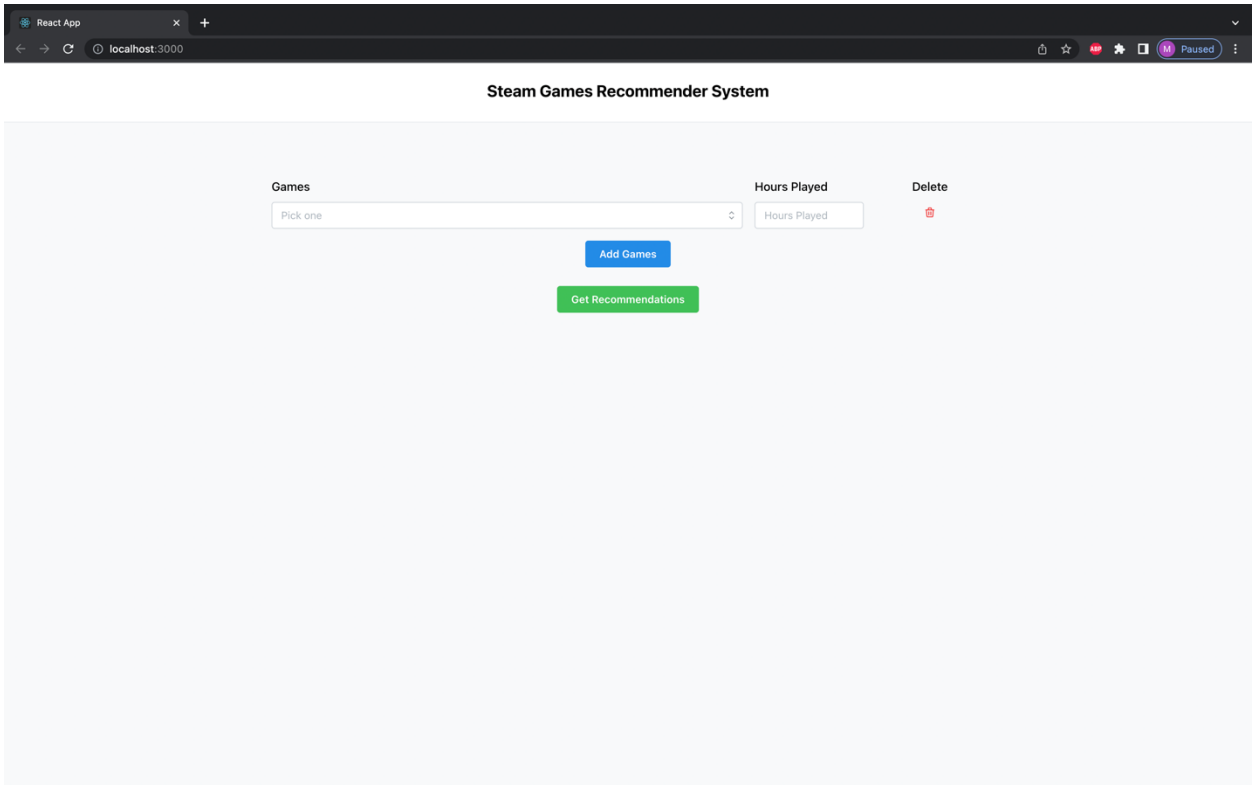

Fig 6. Home Screen for Web Application

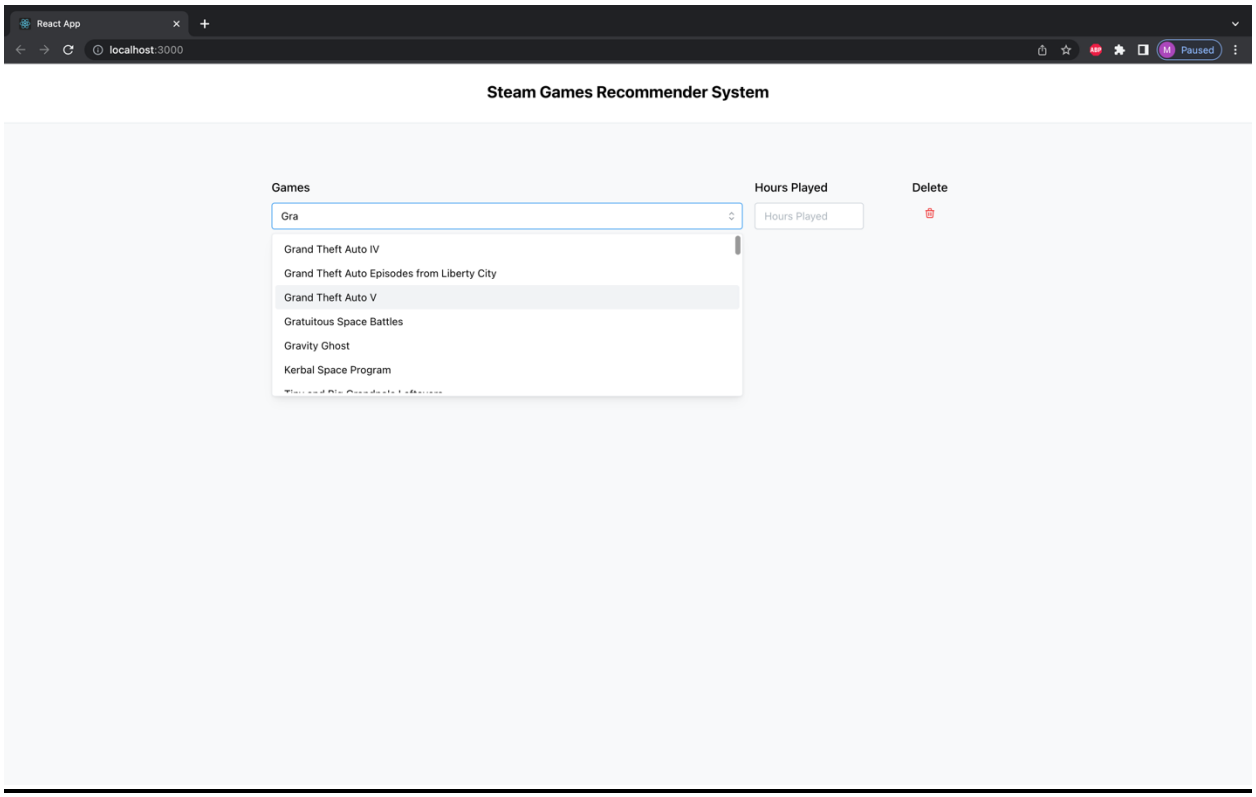

Fig 7. Dropdown searchable list for games

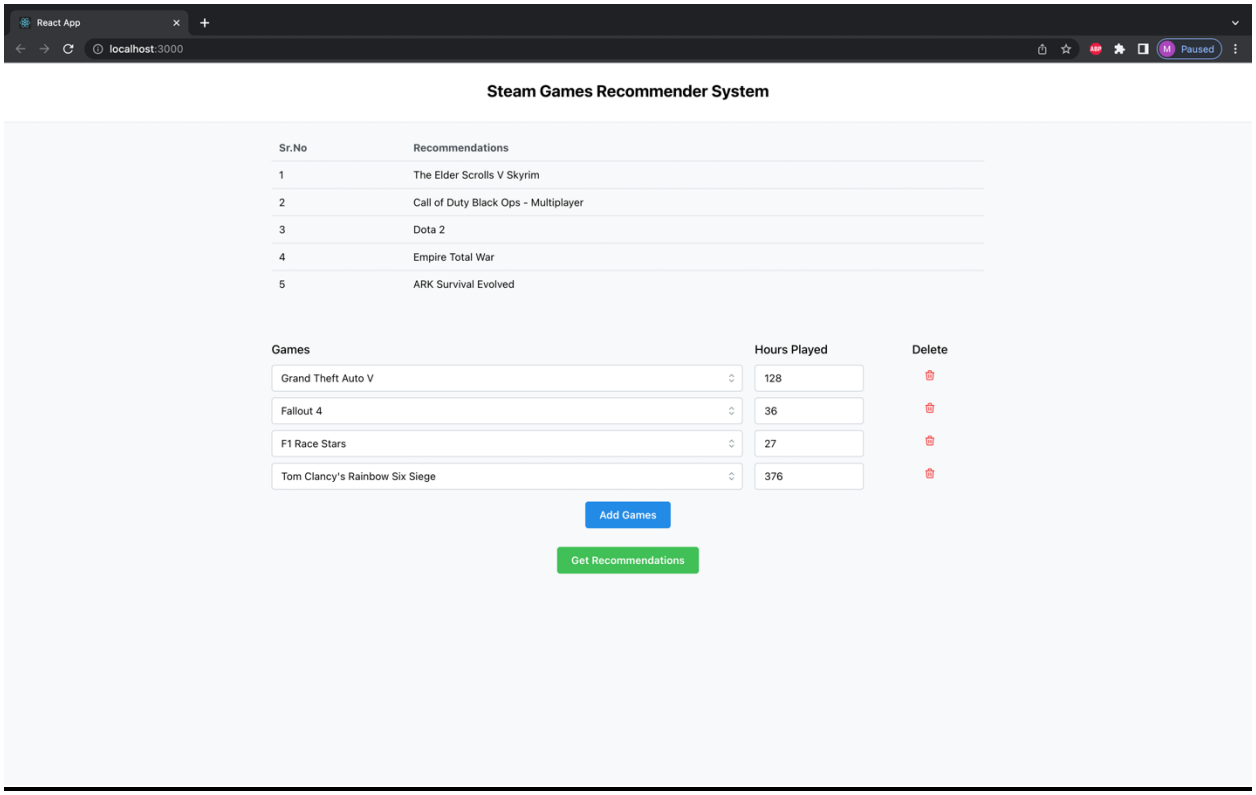

Fig 8. Final Recommendations after giving games played by user as input

6. References

Code samples and understanding of a few concepts were done using the following links:

1. https://rubikscode.net/2018/10/22/implementing-restricted-boltzmann-machine-with-python-and-tensorflow/
2. https://medium.com/machine-learning-researcher/boltzmann-machine-c2ce76d94da5
3. https://wiki.pathmind.com/restricted-boltzmann-machine
4. https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine
5. https://www.latentview.com/blog/restricted-boltzmann-machine-and-its-application/
6. https://adityashrm21.github.io/Restricted-Boltzmann-Machines/