UNIVERSITY OF MALAYA

SEMESTER 2, 2020/2021

WIH2001 DATA ANALYTICS
CASE STUDY

TOPIC : E-Commerce

(Recommendation System)

Prepared by

Soo Mun Chong

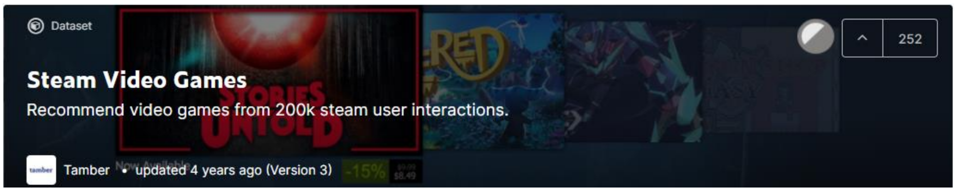
17204674

Lecturer

Dr. Hoo Wai Lam

Question 1

https://www.kaggle.com/tamber/steam-video-games



The case study that I chose is Case 2 : E-commerce. The challenge of e-commerce platforms that I chose to solve is ineffective target marketing via recommended choices.

Steam, one of the most successful E-Commerce platforms today, has over 6000 games and a community of millions of gamers. With the overwhelming number of choices of games, there is need to have a good recommendation system that helps users discover new games that they will most likely purchase.

Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. Mostly used in the digital domain, majority of today's E-Commerce sites like eBay, Alibaba etc make use of their proprietary recommendation algorithms in order to better serve the customers with the products they are bound to like (Doshi, 2019).

The dataset used for this case study is taken from a Kaggle page. The dataset contains 70k records of steam user id, the steam games they bought, and number of hours played per game for each steam user. The main reason why this dataset is useful for the purpose of product recommendation case study that I chose is it contains the information about the user/gamer, which Steam games that users bought, and the how many hours spent playing on the games for each user. These three information are needed to build a product recommendation system.

Question 2:

The dataset contains many attributes, but only a few that are of interests for this recommendation system study. I made some minor adjustments to the original dataset from Kaggle, such as deleting the columns I don't need in this case study and filtering rows that have behavior as play time, because I wanted to have playing time for a particular game as the label. Here is the descriptions of the attributes after adjustments applied to the original kaggle dataset. The modified dataset contains a list of users, the steam games they played, along with the number of hours played for that particular game.

The attributes of the dataset are as follows :

1. userId : Every user identified with a unique id (Structured)
2. steam_game : Every game is identified with a unique name (Unstructured)
3. hours_played : Number of Hours played for a game(Structured)

	userId	Steam_game	Behavior_name	behavior	unknown_column
0	151603712	The Elder Scrolls V Skyrim	purchase	1.0	0
1	151603712	The Elder Scrolls V Skyrim	play	273.0	0
2	151603712	Fallout 4	purchase	1.0	0
3	151603712	Fallout 4	play	87.0	0
4	151603712	Spore	purchase	1.0	0

Figure 1 : Original dataset

	userId	Steam_game	Hours_played
65428	5250	Portal 2	13.6
65430	5250	Alien Swarm	4.9
65432	5250	Team Fortress 2	0.8
65434	5250	Dota 2	0.2
65426	5250	Deus Ex Human Revolution	62.0

Figure 2 : Dataset after deleting unwanted columns

In this case study, the method used to build the recommendation system is called user-to-user collaborative filtering.

However, the structure shown in Figure 2 is not really suitable for building a collaborative filtering model. https://youtu.be/juU7m9rOAqo?t=2192

To provide structure to the data, and given the userId, Steam_game and the hours played for games, we are able to build a pivot table with the rows containing the userId, the columns containing the Steam_game, and the of the table contains the hours played for a particular game by the users. The dataframe is sparse because a steam user certainly does not play majority of the 6000 games in the inventory. This step is crucial to building algorithm later on in the model training phase.

```
steam_pivot = steam_df.pivot(values='Hours_played', index='userId', columns='Steam_game')
```

```
steam_pivot.head()
```

Steam_game	3DMark	4 Elements	7 Days to Die	9 Clues The Secret of Serpent Creek	A Valley Without Wind 2	A Virus Named TOM	A.R.E.S.	A.V.A - Alliance of Valiant Arms	ARK Survival Evolved	Abyss The Wraiths of Eden	...	Your Doodles Are Bugged!	Zack Zero	Zeno Clash	Zero Gear	Zombie Driver	Zombie Panic Source	Zon Sho
userId																		
5250	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
76767	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N
86540	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	N

Figure 2 : Pivot table

Question 3 :

Based on the case study, the potential issue that E-commerce platforms like Steam are facing is ineffective product recommendation for users. Many users are overwhelmed by the game gallery in Steam and therefore they need to be guided towards the most likely product they might purchase.

Recommender systems solve this problem by searching through large volume of dynamically generated information to provide users with personalized content and services. In E-Commerce setting, recommender systems enhance revenues, for the fact that they are effective means of selling more products to potential customers, because it supports users by allowing them to move beyond catalog searches.

Approach Used:

The data analytics techniques that I used are focused around collaborative filtering approach. There are two primary areas of collaborative filtering, which are neighborhood methods and latent factor models. Throughout this project, I am focusing on using latent factor model to build a video game recommendation system.

Broadly speaking, matrix factorization which is what latent factor model is based on, tries to explain the ratings by characterizing both items and users on, say 20 to 100 latent/hidden factors or features inferred from the ratings patterns. For games, the discovered factors might measure obvious dimensions/genres such as adventure, MMORPG, action or strategy-based; less well-defined dimensions such as quirkiness; or completely uninterpretable dimensions. For each steam user, each latent factor/"features" measures how much the user likes games that score high on the corresponding game factor. Matrix factorization breaks down matrix A into User matrix, $P \in R^{m \times f}$ and Item Matrix, $Q \in R^{f \times n}$, where m is number of users/rows and n is number of items/games in matrix A, and f is the dimensionality/number of latent factors as shown in Figure 3 and Figure 4. (Koren et al., 2009)

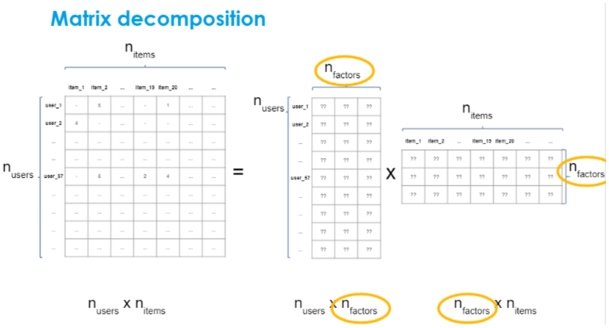


Figure 3 : Intuition of latent factor model. It decomposes the pivot matrix into two matrices, P and Q, with P representing User matrix and Q representing item matrix.

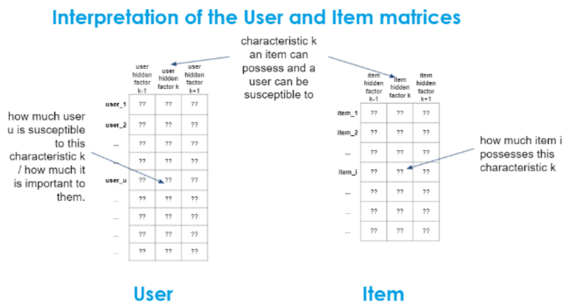


Figure 4 : Interpretation of User and Item Matrices

A Basic Matrix Factorization Model :

Basically, matrix factorization characterizes both items and users by vectors of factors inferred from item rating patterns. In this case, we characterize gamers and games by latent factors inferred from number of hours played per game patterns. The number of latent factors is a hyperparameter that has to be tuned. The more number of latent factors, the more complex it is and more prone to noise. It maps both gamers and steam_game to a joint latent factor space of dimensionality f, such that "user-item" interactions are modeled as inner products in that space. Similarity metrics such as cosine similarity is then used to measure how likely a gamer likes a particular steam_game. Geometric intuition is shown in Figure below.

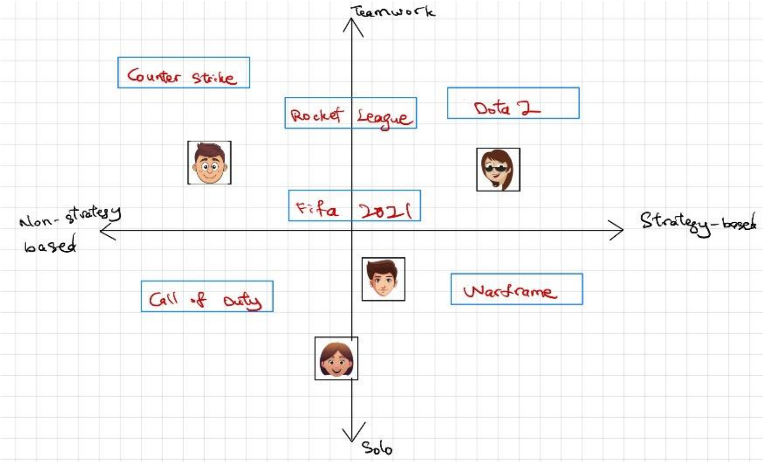


Figure 5: Self-drawn illustration of the latent factor approach, which characterizes both gamers and games using two latent factor vectors inferred as basis vectors (Example). Here, only 2 latent factor vectors are inferred(2-dimensional).

Accordingly, each item/game i is associated with a vector $q_i \in R^f$, and each user u is associated with a vector $p_u \in R^f$. For a given item i, the elements of q_i measure the extent to which the item/game possesses the latent factors/features. For a given user u, the elements of p_u measure the extent of interest the gamer has in games that are high on the corresponding factors. The resulting dot product, $q_i^T p_u$ captures the interaction between user u and game i – the user's overall interest in the game's characteristics (Koren et al., 2009). This approximates user u's playing hours of game i, which is donated by $r_{ui} = q_i^T p_u$.

Hence, the first step is to compute the mapping from each user/gamer and game to latent factor vectors q_i, $p_u \in R^f$. After the mapping, the resulting dot product $q_i^T p_u$ is the estimate of the number of hours that user/gamer will spend playing the item/game i.

Such a model is similar to Singular Value Decomposition (SVD) in the sense that both of the methods identify latent factors from a high rank matrix. However, in general recommendation pivot tables, the matrix is very sparse as a particular gamer certainly do not play all the games in the inventory. Conventional SVD requires a complete matrix for factorization. And some old-fashioned ways of fixing this problem is filling NaN values in the sparse matrix with 0, and then apply SVD to it, but then, this inaccurate assumption that all NaN values is 0 might distort the data considerably. As shown in Figure , conventional SVD decomposes the matrix into U, Σ and V^T (Steve Brunton, 2020).

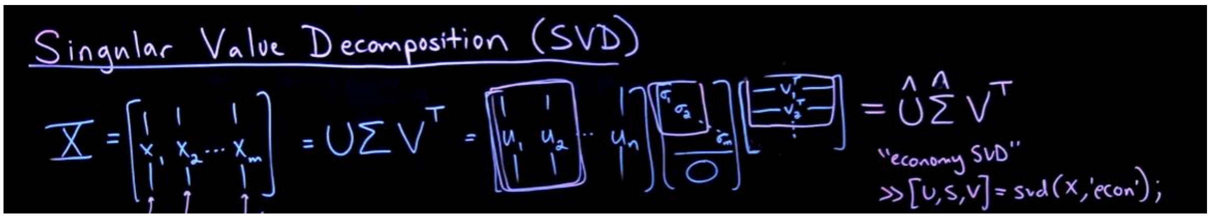


Figure 6 : Singular Value Decomposition(SVD), taken from Steve Brunton youtube channel (Steve Brunton, 2020).

In our latent model-based method, we inject Σ into U and V^T, thereby creating PQ^T, with P representing user/gamer matrix and Q representing item/game matrix (PyData, 2019). As shown in Figure 4 above in previous page.

Hence, a modern way of building a latent factor model on a recommender system is to model the observed valid values in the sparse pivot table/matrix, while ignoring the other non-recorded entry in the pivot table.

To learn the k latent factor vectors, where k is the low dimensionality in the joint latent factor space, the model tries to minimize the squared error on the set of known entries in the sparse pivot table (PyData, 2019).

$$\sum_{r_{ui} \in R} (r_{ui} - p_u \cdot q_i)^2 \quad (1)$$

Where p_u is elements of user latent vectors for user u, q_i is for item i, and r_{ui} is the known entry/hours played of user u and game i in the pivot table.

Applying partial derivative to the Equation 2 with respect to p and q gives (PyData, 2019):

$$\frac{\partial f_{ui}}{\partial p_u} = \frac{\partial}{\partial p_u}(r_{ui} - p_u \cdot q_i)^2 = -2q_i(r_{ui} - p_u \cdot q_i)$$
$$\frac{\partial f_{ui}}{\partial q_i} = \frac{\partial}{\partial q_i}(r_{ui} - p_u \cdot q_i)^2 = -2p_u(r_{ui} - p_u \cdot q_i) \quad (2)$$

The algorithm thus reads (PyData, 2019):

- initialize P and Q to random values
- for N passes/n epochs on the data
  - for all known entries in the sparse pivot table, r_{ui}, repeat:
    - update p_u and q_i with the following rule:

$$p_u \leftarrow p_u + \alpha \cdot q_i(r_{ui} - p_u \cdot q_i) \quad (3)$$
$$q_i \leftarrow q_i + \alpha \cdot p_u(r_{ui} - p_u \cdot q_i)$$

But, to avoid overfitting, a regularization term is introduced to the equation (1) (Koren et al., 2009).

$$\min_{q^*, p^*} \sum_{(u,i) \in \kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2) \quad (4)$$

where K is the set of (u, i) pairs for which r_{ui} is the known entry in the sparse pivot table.

Hence, the updating rule becomes (Koren et al., 2009):

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u) \quad (5)$$

Learning Algorithms (Stochastic Gradient Descent)

The approach I used to minimize the function in Equation (4) is called Stochastic Gradient Descent. The reason that it is called stochastic instead of normal gradient descent is that we update the parameters for each predicted r_{ui} instead of one iteration/epoch.

Essentially, what Stochastic Gradient Descent does is it loops through all the known entries in the sparse pivot table, as shown in Figure 7 below. For a given known training data point or known entry, the model predicts r_{ui} and the prediction error $e_{ui} = r_{ui} - q_i^T p$. Then it updates the p_u and q_i parameter as in equation (5) above.

Steam_game	3DMark	4 Elements	7 Days to Die	9 Clues The Secret of Serpent Creek	A Valley Without Wind 2	A Virus Named TOM	A.
userId							
5250	NaN	NaN	NaN	NaN	NaN	NaN	
76767	NaN	NaN	NaN	NaN	NaN	NaN	
86540	NaN	NaN	NaN	NaN	NaN	NaN	

Figure 7 : Sparse pivot table

This approach combines implementation ease with a relatively fast running time.

For evaluation purpose, if we used all the known entries in the sparse pivot table for training, then we are left with no known entries for evaluation purpose. Since this is also a supervised learning, there is need to have a separate evaluation set. Therefore, what I did was I kept some known entries in the pivot table as evaluation set and did not include it in the training phase. During the evaluation phase, I used the evaluation set that I kept earlier to test how well the model performs on unseen data (Valkov, 2019).

In the training phase, I used GridSearch for tuning the hyperparameters, number of latent factors, gamma and lambda.

```
n_factors = [10, 20, 30]
gammas = [0.005, 0.01]
lambda_values = [0.001, 0.1, 0.5]
for factor in n_factors:
    for gamma in gammas:
        for reg in lambda_values:
            p, q, train_error, val_error = SGD(steam_df, n_factors = factor, gamma = gamma, lambda_val = reg)
            print(f'\nHyperparameters : n_factors = {factor}, gamma = {gamma}, lambda = {reg} : \n \
                  Train Error : {train_error}, Validation Error : {val_error}')
```

Figure 8: Hyperparameter tuning using GridSearch

For measuring the error, one of the most popular metrics used to evaluate the accuracy of Recommender Systems is Root Mean Squared Error (Valkov, 2019).

$$RMSE = \sqrt{\frac{1}{N}\sum(y_{ui} - \hat{y}_{ui}^{2})}$$

(6)

Where y_{ui} is the actual number of hours played for a game I by user u, y_{ui} hat is the predicted one, and N is training size.

Question 4

During the training phase, there are many hyperparameters to be tuned, and the results of hyperparameter tuning using GridSearch is shown in Appendix A. Then, I chose the best set of hyperparameters for training the collaborative filtering latent factor model.

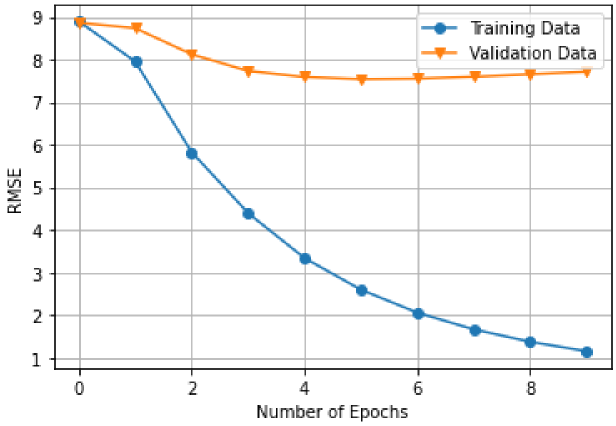


Figure 9: The root mean squared error(RMSE) plot of training data and validation data.

For a more intuitive way of seeing how accurate the trained model is, I tested the collaborative filtering model built by predicting the most favourite games of a user in terms of playing hours in the original dataset. Here is the result :

```
steam_data.iloc[10].sort_values(ascending=False)[:10]
```

```
Steam_game
Wolfenstein                        26.0
Left 4 Dead 2                      26.0
Battlefield Bad Company 2          20.0
Call of Duty Modern Warfare 2      17.5
Metro 2033                         16.9
Half-Life 2                        16.6
Killing Floor                      14.3
Duke Nukem Forever                 14.3
Sniper Elite V2                    14.2
Mass Effect                        13.7
Name: 561758, dtype: float64
```

```
df_user_item_filled.iloc[10].sort_values(ascending=False)[:10]
```

```
Wolfenstein                        25.747800
Left 4 Dead 2                      23.496496
Far Cry 2                          22.974824
Battlefield Bad Company 2          21.988369
Call of Duty Modern Warfare 2      20.429848
FINAL FANTASY XIV A Realm Reborn   20.384014
Wasteland 2                        19.443644
DiRT 3                             19.209970
Half-Life 2                        18.525060
Rust                               18.354479
Name: 561758, dtype: float64
```

Figure : Actual playing hours of games for User 561758

Figure : Predicted playing hours of games for User 561758

As we can see from the results, the steam user likes to play adventure, first person shooting genres of game, and the predicted results show some accurate predictions and some recommended games that are similar to the ones he/she played before that the user will most likely buy.

Reference :

Doshi, S. (2019, December 17). *Brief on Recommender Systems - Towards Data Science*. Medium. https://towardsdatascience.com/brief-on-recommender-systems-b86a1068a4dd

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer, 42*(8), 30–37. https://doi.org/10.1109/mc.2009.263

PyData. (2019, August 29). *Hands on - Build a Recommender system: Camille Couturier | PyData Amsterdam 2019* [Video]. YouTube. https://www.youtube.com/watch?v=juU7m9rOAqo

Steve Brunton. (2020, January 19). *Singular Value Decomposition (SVD): Mathematical Overview* [Video]. YouTube. https://www.youtube.com/watch?v=nbBvuuNVfco

Valkov, V. (2019, June 30). *Music artist Recommender System using Stochastic Gradient Descent | Machine Learning from Scratch (Part VII)*. Medium. https://towardsdatascience.com/music-artist-recommender-system-using-stochastic-gradient-descent-machine-learning-from-scratch-5f2f1aae972c

Appendix A

```
Hyperparameters : n_factors = 10, gamma = 0.005, lambda = 0.001 :
                   Train Error : 8.909745126680464, Validation Error : 9.274877706458964

Hyperparameters : n_factors = 10, gamma = 0.005, lambda = 0.1 :
                   Train Error : 8.926090031612224, Validation Error : 8.965961136473112

Hyperparameters : n_factors = 10, gamma = 0.005, lambda = 0.5 :
                   Train Error : 8.93844773328159, Validation Error : 9.085130930511804

Hyperparameters : n_factors = 10, gamma = 0.01, lambda = 0.001 :
                   Train Error : 8.537490415546785, Validation Error : 7.997279891980885

Hyperparameters : n_factors = 10, gamma = 0.01, lambda = 0.1 :
                   Train Error : 8.560260219834012, Validation Error : 7.385002676669853

Hyperparameters : n_factors = 10, gamma = 0.01, lambda = 0.5 :
                   Train Error : 8.791808373547834, Validation Error : 8.350071309153904

Hyperparameters : n_factors = 20, gamma = 0.005, lambda = 0.001 :
                   Train Error : 8.900729779308751, Validation Error : 8.520977194986738

Hyperparameters : n_factors = 20, gamma = 0.005, lambda = 0.1 :
                   Train Error : 8.922404536811925, Validation Error : 8.221686533147802

Hyperparameters : n_factors = 20, gamma = 0.005, lambda = 0.5 :
                   Train Error : 8.99464303572115, Validation Error : 6.919181728231755

Hyperparameters : n_factors = 20, gamma = 0.01, lambda = 0.001 :
                   Train Error : 8.069081293844826, Validation Error : 7.880334952899474

Hyperparameters : n_factors = 20, gamma = 0.01, lambda = 0.1 :
                   Train Error : 8.086897994695804, Validation Error : 6.706117534555613

Hyperparameters : n_factors = 20, gamma = 0.01, lambda = 0.5 :
                   Train Error : 8.594410345378195, Validation Error : 8.096083753717961


Hyperparameters : n_factors = 30, gamma = 0.005, lambda = 0.001 :
                   Train Error : 8.846252981457633, Validation Error : 8.306629581583696

Hyperparameters : n_factors = 30, gamma = 0.005, lambda = 0.1 :
                   Train Error : 8.883461316204802, Validation Error : 8.419471373179286

Hyperparameters : n_factors = 30, gamma = 0.005, lambda = 0.5 :
                   Train Error : 8.927398000429958, Validation Error : 8.423284032559224

Hyperparameters : n_factors = 30, gamma = 0.01, lambda = 0.001 :
                   Train Error : 7.685875852065938, Validation Error : 6.367249728962334

Hyperparameters : n_factors = 30, gamma = 0.01, lambda = 0.1 :
                   Train Error : 7.7025679191539345, Validation Error : 7.405452721556388

Hyperparameters : n_factors = 30, gamma = 0.01, lambda = 0.5 :
                   Train Error : 8.602992366069913, Validation Error : 7.68275279095803
```