Analysis of Top N recommenders

Neham Jain
ee19b084@smail.iitm.ac.in

Shreyas Kulkarni
me19b193@smail.iitm.ac.in

Tanay Dixit
ee19b123@smail.iitm.ac.in

Vibhhu Sharma
ee19b128@smail.iitm.ac.in

June 15, 2020

Abstract

A product recommender system based on product-review information and metadata history was implemented in our project. The primary goal for our recommender system is to recommend 'top N' products to a user based on his/her purchase history(explicit data). We used collaborative filtering models with both user-based and item-based strategies, matrix factorization/ Autoencoders and hybrid models to recommend products. The strengths and weaknesses of each model were analysed using targeted error metrics like Hit rate & Average Reciprocal Hit Rate so as to gain insight into making a better model.

Introduction

Recommender Systems are in wide use by companies working in diverse fields today. The focus in various spheres is shifting towards a more consumer-oriented approach and this requires a strategy that will keep him/her satisfied.

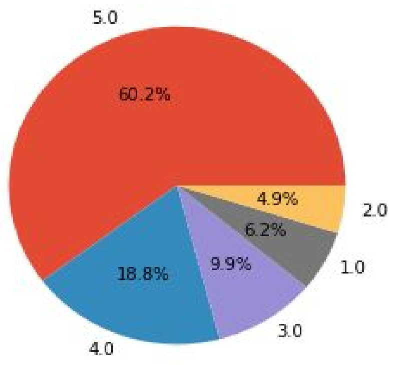
Recommender Systems aim to predict the preference/taste of a user so as to recommend products that he/she is likely to enjoy. Such an approach performs better than an approach based purely on popularity metrics and other such standard methods. The touch of personalization offered by a recommender system can generate a high conversion rate because of the relevant results generated by it.

Current recommender systems generally fall into two categories: *content-based filtering* and *collaborative filtering*. We experiment with both approaches in our project. For content-based filtering, we take metadata of Video Games such as title, description, features as inputs and use TF-IDF to calculate the similarity between video games. For collaborative filtering, the input to our algorithm is the observed users' movie rating, and we use K-nearest neighbours and matrix factorization methods to predict user's movie ratings. Metrics used to compare the models are Hit Rate, Average Reciprocal Hit Rate(ARHR), Novelty & Diversity each measured for a list of 1000 test users.

For our analysis, we used the Video Games dataset provided by Stanford Network Analysis group:(Julian McAuley, UCSD). We worked with explicit data which had 446k reviews provided by 55k users to 17k products. Along with this we also used product metadata that consists of a unique product identifier, title, category, description, timestamp(excluded in our study) and brand.

Basic exploratory analysis revealed that the reviews were highly skewed. Furthermore, only a small number of products were bought frequently(popular products), also the sparsity in the user-item matrix is 0.26%.

Fig-1 shows the skewed nature of ratings with 79% of ratings>=4 on a scale of 5



Rating Distribution (Fig-1)

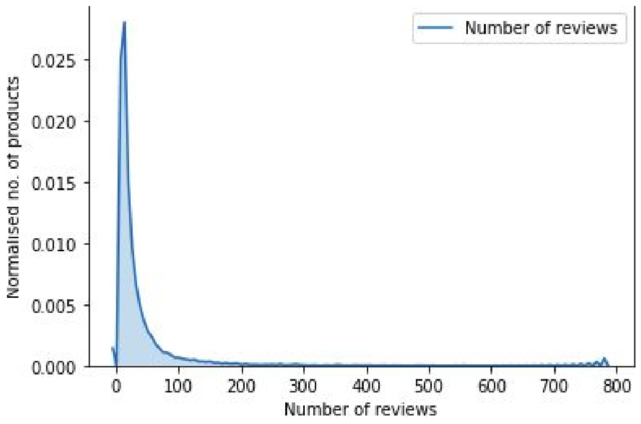
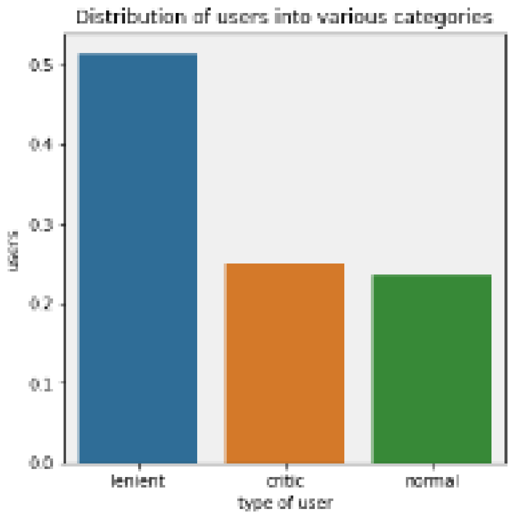


Fig 2: Long-tail plot

User analysis showed that many users purchased a large number of popular products, while some niche users had purchased a large number of non-popular products.

Fig-2 illustrates the long tail problem. Most products have very few ratings to their name. Only a few selected products have a large number of reviews(>500) while most are in the range between 0-50.

As such, understanding user behaviour was key to building a good recommender system so appropriate user bias had to be added when rating prediction models were implemented based on how critical or lenient the user was while rating a product.



Item analysis showed:

The meta-data of the products gave information about the category, brand, shape/size, title, description, etc, the most important being category and Brand. It was observed that only a few brand's items were bought a lot mainly because they provided good games or other factors(Fig3). Even analysing categories was key(Fig 4) in content-based filtering as users who like PC games may dislike console-based games. A product could belong to one or more than one category.

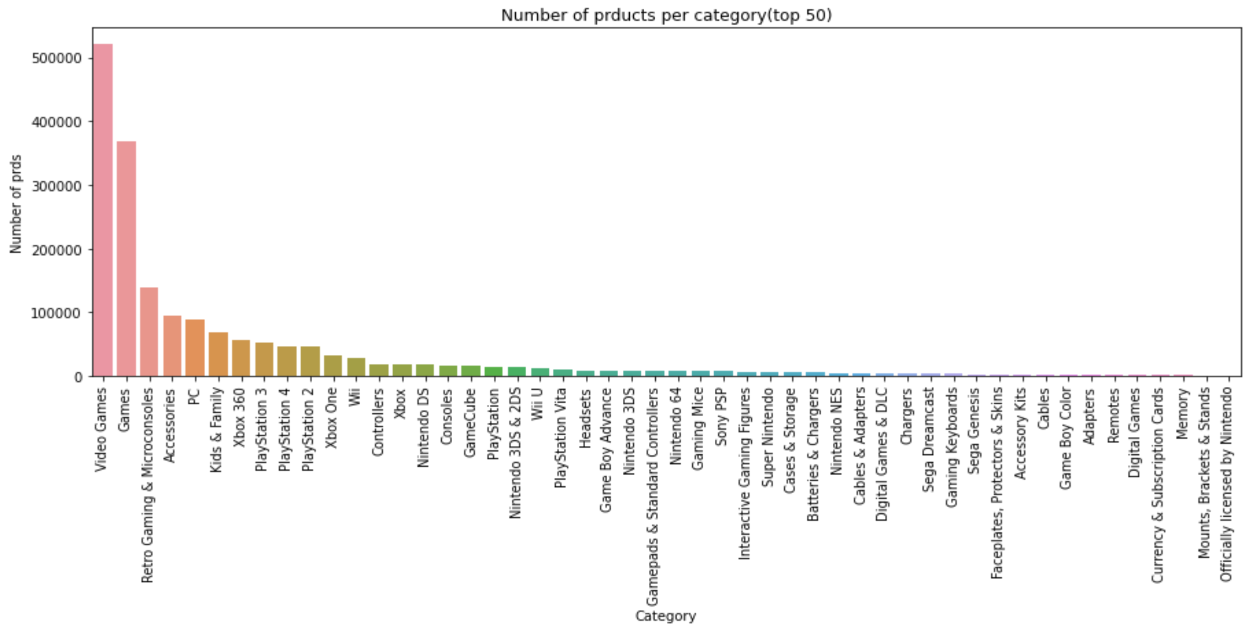


Fig3: This plot shows the number of products bought by users for each category

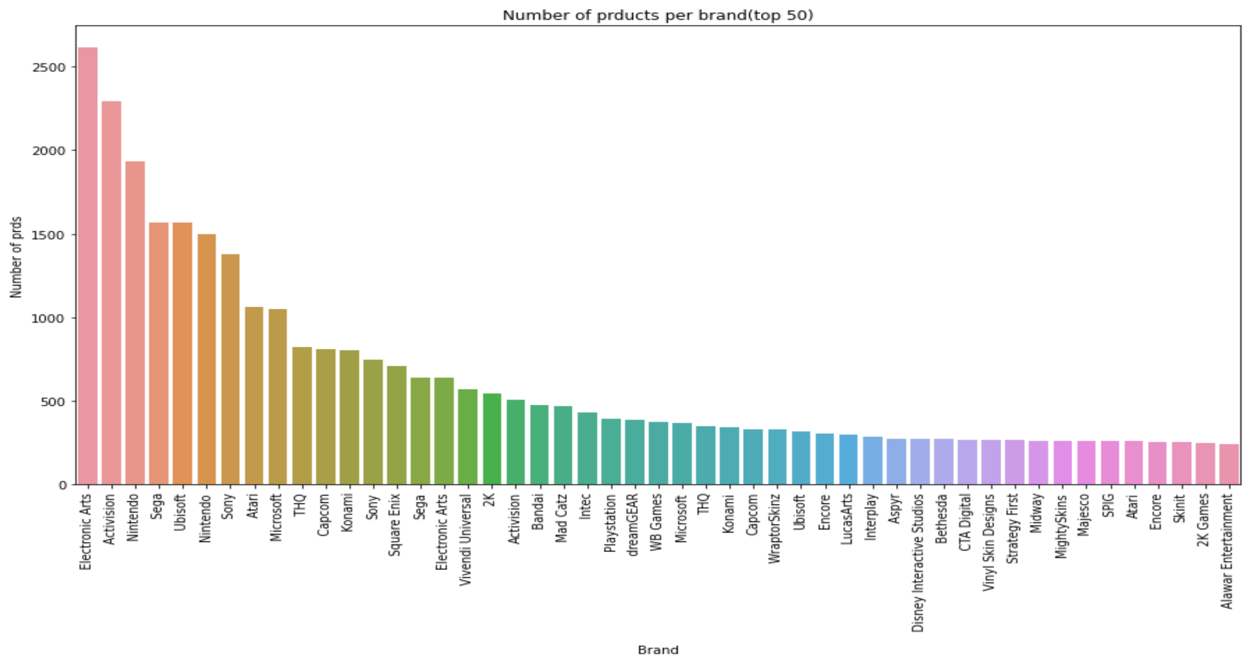


Fig4: This plot shows the number of products bought by users for each Brand

For each of the following methods, we analyse the top 10 recommendations for a particular user (UserId: A14XH33SGMTA7R). This user was selected because he had a clear liking for action/adventure video games as he had purchased games like X-Men, Street Fighter, Marvel Superheroes Set, Age of Empires and other PS games.

3)Methods

1)Memory-based

1.1)Content-based Collaborative Filtering

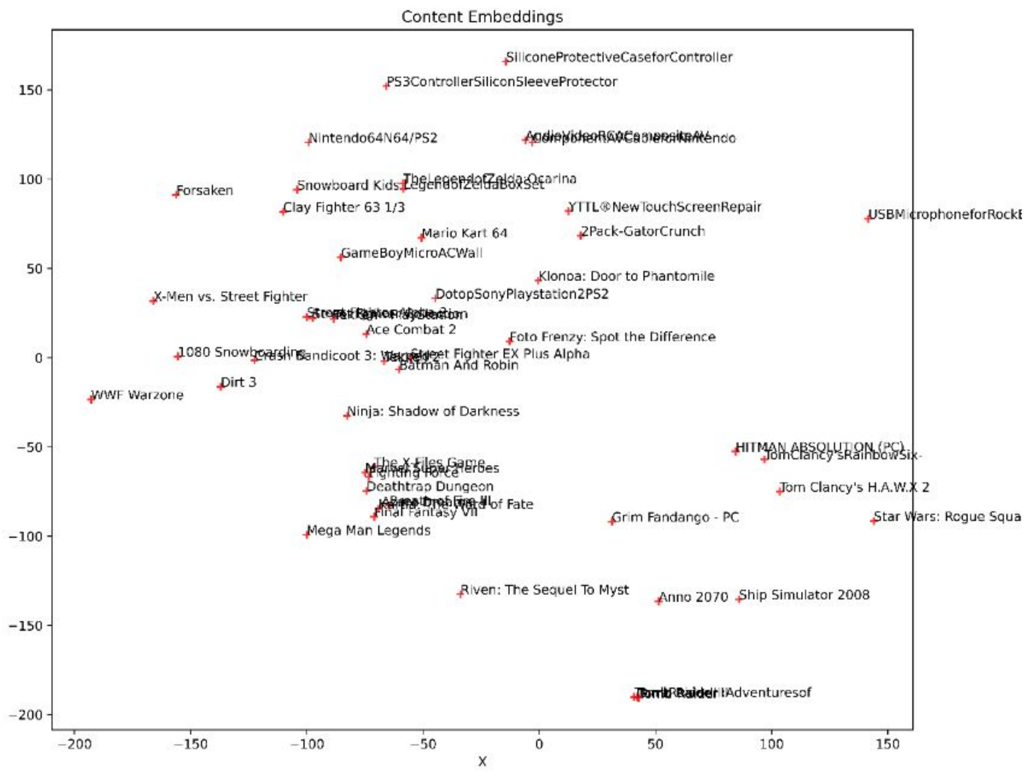
Content-based CF is a model based on recommending similar products on the basis of content. To find similar products we make use of the product title, features & category present in the meta-data file. We merge these columns for each product, apply cleaning algorithms ie: Stemming/lemmatization and remove stop words. We apply Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer. This will give a matrix where each column represents a word in the content vocabulary (all the words that appear in at least one document) and each row represents an item, as before. This is done to reduce the importance of words that occur frequently and therefore, their significance in computing the final similarity score.

To reduce the size of the TF-IDF matrix, Truncated SVD was implemented to reduce the dimensions to much more condensed space. We use cosine similarity on the TF-IDF vectors for each product in order to find similar products. A cosine similarity of 1 implies the two products are identical while a score of 0 means they are completely unrelated.

$$cosine\ similarity = cos(\Theta) = \frac{A.B}{|A||B|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

We identify the products which are similar to a user's purchase history and multiply each similarity score by the rating the user gave to the corresponding product, then we sort the products by their net score and recommend the top 20 products to a user.

The following plot is of the Content Embeddings obtained from the item-Tfidf matrix. The Tfidf matrix was compressed to a 2-dimensional space with the help of t-SNE algorithm.



Products having similar features are closer to each other as compared to dissimilar products. As a simple example: all the gaming consoles can be seen on one side of the plot, all the video games can be seen on another side of the plot. So our content-based system looks at the user's purchase history and recommends products having similar features.

Top 10 recommendations for our sample user by Content-based CF :

1. Klonoa 2: Lunatea's Veil
2. PlayStation 2 SingStar Bundle - Ceramic White
3. Samurai Warriors 4 Empires - PlayStation 4
4. Age Of Mythology: Titans - PC
5. PlayStation 2 Console (Slim Line Version 1)
6. Skylanders Giants GAME ONLY for the PS3
7. Playstation 2 Console Slim - Ceramic White
8. Saints Row IV: Re-Elected + Gat out of Hell
9. Civilization II: Multiplayer (Gold Edition)
10. AmazonBasics Heavy-Duty Vault Case for PlayStation Vita and Vita Slim (Officially Licensed by Sony)

These recommendations are quite good as they are all action related games that the user hasn't bought. The best part of content-based recommendations is, it recommends items from the *long tail* as a result helping new items get discovered.

1.2) Item-based Collaborative Filtering

Item-based CF helps to find similar products based on all the user's purchase patterns. This algorithm was very popular in Amazon recommendations systems, Eg. When you buy any product on Amazon, you will find this line "Users who bought this item also bought...",
We use the item-user matrix which is a sparse matrix with items as the rows and users as the columns and the entries are the rating a user gave to a particular item. We compute the cosine similarity between two item vectors in the user dimension. Then we find similar products to the products a user purchased and multiply the similarity score by the rating the user gave to the corresponding product. After sorting we recommend the highest rated products.

In the beginning, we used NearestNeighbours algorithm to find similar items, but this approach was time-consuming as it took 7 sec per user, in order to improve this we decided to build it from scratch with the help of cosine similarity kernel(inbuilt python function) along with our own functions. This approach reduced the computational time to 0.1 sec, which is 70 times faster and didn't compromise Hit Rate at all.
Another plus point was that by removing KNN we were able to parallelise the computation for different users which further improved the computational time.

Top 10 recommendations for our sample user by Item-based CF

1. X-Men: Children of the Atom - PlayStation
2. Dragon Ball GT: Final Bout
3. Rival Schools
4. The Simpsons: Night of the Living Treehouse of Horror
5. Driver 2 Advance
6. Mega Man Anniversary Collection - PlayStation 2
7. Samurai Shodown - Sega Genesis
8. Street Fighter III 3rd Strike
9. Mortal Kombat 4
10. Puyo Pop Fever

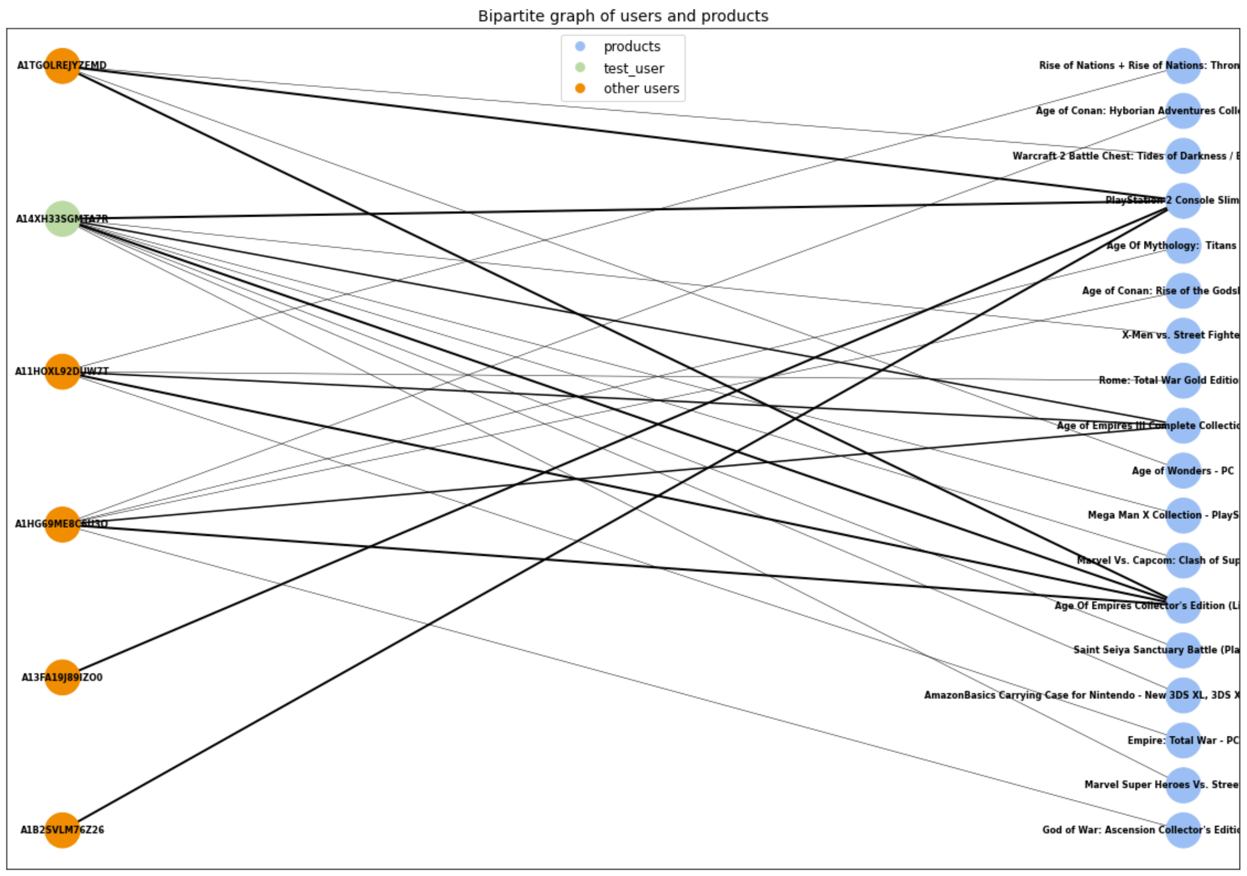
Item-based recommendations are arguably one of the best as they not only recommend new games but also the new versions of the games the user has played, ex Street Fighter 3. However, one important point is that all these games are popular and none are from the *long tail*.

1.3)User-based Collaborative Filtering

User-based CF involves the basic idea that users with similar past purchase histories/interests are likely to continue to show interest in similar products.
This model uses a "user-item" matrix-a sparse matrix with each user as a separate row and each item as a column. The matrix is populated by the ratings given by a user to a corresponding item. Each row of this matrix now serves as a vector which is then used to compute cosine similarity with the vector representing a different user.

The top 30 most similar users to the user under consideration are found and their respective ratings to each item are scaled by the similarity score for the user. Ultimately, each item gets a final rating by summing over the values obtained for the 30 users. They are sorted according to this rating and the highest rated products are recommended.

The following plot depicts the interaction between users and items.The green user is our test user and the remaining are users are who have similar buying history (shown in dark lines)


Bipartite graph of users and products
products
test_user
other users
A1TGOLREJYZEMD
A14XH33SGMTA7R
A11HOXL92DUW7T
A1HG69ME8CEU3O
A13FA19J89IZ00
A1B2SVLM76Z26
Rise of Nations + Rise of Nations: Thron
Age of Conan: Hyborian Adventures Coll
Warcraft 2 Battle Chest: Tides of Darkness / E
PlayStation 2 Console Slim
Age Of Mythology: Titans
Age of Conan: Rise of the Godsl
X-Men vs. Street Fighte
Rome: Total War Gold Editio
Age of Empires III Complete Collectio
Age of Wonders - PC
Mega Man X Collection - PlayS
Marvel Vs. Capcom: Clash of Sup
Age Of Empires Collector's Edition (Li
Saint Seiya Sanctuary Battle (Pla
AmazonBasics Carrying Case for Nintendo - New 3DS XL, 3DS X
Empire: Total War - PC
Marvel Super Heroes Vs. Stree
God of War: Ascension Collector's Editio

Top 10 recommendations for our sample user by user-based CF

1. Age Of Mythology: Titans - PC
2. Age of Wonders - PC
3. Warcraft 2 Battle Chest: Tides of Darkness / Beyond the Dark Portal
4. Kingdom Hearts
5. Rome: Total War Gold Edition - PC
6. Rise of Nations + Rise of Nations: Thrones & Patriots - PC
7. Empire: Total War - PC
8. Age of Conan: Hyborian Adventures Collector's Edition - PC
9. Age of Conan: Rise of the Godslayer - PC
10. God of War: Ascension Collector's Edition - Playstation 3

These recommendations are a little off as they're all PC games and the user showed no specific inclination towards PC games, though the genre of all games is as per his interest. This is one flaw of user-based, it is very difficult to find similar users based on purchase pattern unless one has a really big dataset to work with

One more problem in user-based collaborative approaches is that it doesn't treat likes and dislikes separately, i.e., it may say two users are similar if they hate the common products also, but they may not necessarily like the same things which matters the most while providing recommendations.

2)Model-based

2.1) Matrix Factorization

Matrix factorization decomposes the user-item matrix into the product of two smaller matrices: the first one has a row for each user, while the second has a column for each item. These matrices represent user-latent and item-latent matrices respectively. Matrix factorization forces the model to learn 'k' latent features for each user and product. By reducing dimensionality, it allows the model to focus on the factors most relevant to rating prediction.

The predictions are further improved by adding a user bias and item bias, unique for each user/item, as well as the global rating mean. The global ratings mean was 4.2329. This allows the model to better differentiate between users who are more inclined to always give good ratings and those who generally rate negative products.

$$\overline{r_{ui}} = \mu + b_i + b_u + q_i p_u^T$$

The predicted rating which a user (u) will give an item (i) is given by the sum of μ (global ratings mean), b_i the item bias, b_u the user bias and the Σ term obtained by taking the dot product of a row from item latent matrix (q_i)with a column from user latent matrix(p_u^T)

$$\text{Loss} = \sum_{u\ \varepsilon\ \#users} \sum_{i\ \varepsilon\ \#items} \left(r_{ui} - \overline{r_{ui}}\right)^2$$

Note: Mean squared error is computed on already known ratings and the objective function aims to minimise this error. Stochastic Gradient Descent(SGD) outperformed Alternating Least Squares(ALS) in RMSE score hence we used SGD to minimise the loss function.

The first implementation was done without Keras/TensorFlow with no in-built SGD. SGD on the loss function was performed using NumPy functions. As a result, it was computationally expensive. To make use of in-build SGD we used Keras with a basic architecture of input, user embeddings, item embeddings and user/item bias. These layers were added and the dot product was fed to the output layer(ratings). It was much faster than the previous method and even RMSE score improved by a margin of (0.1).

To make it even faster and improve accuracy/scalability we decided to use Apple's open-sourced library Turi create. This library was chosen because it had many methods which were very helpful in providing Top-N recommenders ie; RankingFactorizationRecommedner

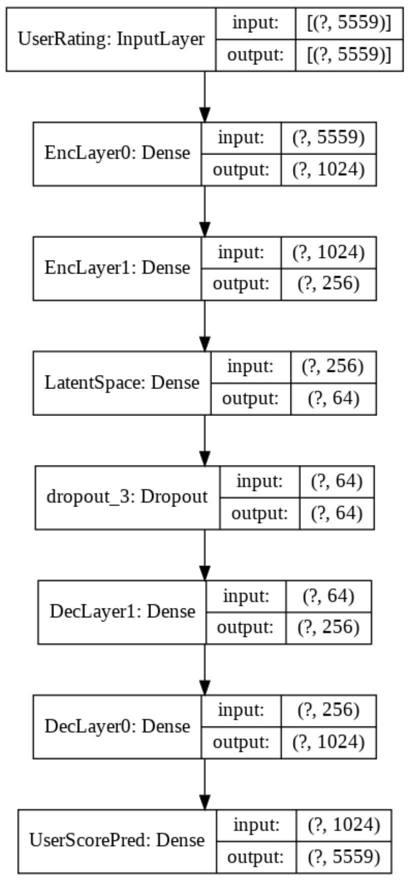
Top 10 recommendations by using Matrix Factorization for our sample user :

1. Xbox One Play and Charge Kit
2. Uncharted: The Nathan Drake Collection -PS4
3. The Legend Of Zelda
4. Tomb Raider: Definitive Edition: PS4
5. Tomb Raider
6. Pokemon Y
7. PlayStation 2 Dualshock Controller Black
8. Minecraft - PlayStation 3
9. Gears of wars 3
10. Xbox One Wireless Controller (Without 3.5-millimeter headset jack)

This model is doing quite well even though its hit rate is relatively low as it is able to understand the fact the user likes action games. Furthermore, it has also figured out that the user has not purchased any additional accessories and is recommending them accordingly too.

2.2)AutoRec

Although the matrix factorization model achieves decent performance on the rating prediction task, it is essentially a linear model. Thus, such models are not capable of capturing complex nonlinear and intricate relationships that may be predictive of users' preferences.



We introduce a nonlinear neural network collaborative filtering model, which essentially performs collaborative filtering (CF) with an autoencoder architecture and aims to integrate nonlinear transformations into CF on the basis of explicit feedback. Neural networks have been proven to be capable of approximating any continuous function, making it suitable to address the limitation of matrix factorization and enrich the expressiveness of matrix factorization.

On one hand, AutoRec has the same structure as an autoencoder which consists of an input layer, a hidden layer, and a reconstruction (output) layer. An autoencoder is a neural network that learns to copy its input to its output in order to code the inputs into the hidden (and usually low-dimensional) representations. In AutoRec, instead of explicitly embedding users/items into low-dimensional space, it uses the column/row of the interaction matrix as the input, then reconstructs the interaction matrix in the output layer.

AutoRec doesn’t drastically improve the performance of ‘top N’ recommenders, thus implying that there aren’t many non-linear features to extract from the data.

Top 10 recommendations for our sample user by AutoRec

1. Xbox 360 Wireless Controller - Glossy Black
2. Super Smash Bros. - Nintendo 3DS
3. Redragon M601 Wired Gaming Mouse, Ergonomic, Programmable 6 Buttons, 3200 DPI with Red LED Mouse for Windows PC Games - Black
4. Xbox One Play and Charge Kit
5. Turtle Beach - Ear Force PX22 Universal Amplified Gaming Headset - PS3, Xbox 360, PC
6. Turtle Beach - Ear Force PX22 - Universal Amplified Gaming Headset- PS3, Xbox 360, PC - FFP [Old Version]
7. Assassin's Creed - Ezio Trilogy Edition Xbox 360
8. Halo 4: Game of the Year Edition
9. Nintendo Wii U Pro Controller - Black
10. PlayStation 2 Dualshock Controller Black

3)Hybrid Recommenders

Combining models in order to learn from one other’s weakness is key to building a good recommender. The main drawback of Collaborative Filtering models is the cold start problem. The recommenders cannot learn enough features from new users/items, as they have very little interaction with the items/users, hence as a result performs poorly on them. Solving the cold start problem is key in attracting new users and incorporating new products into the system, it's a win-win situation for all.

In order to solve this issue, ensembling content-based recommenders with Collaborative Filtering models will help it learn more about new users/items. Content-based model is weak by itself since it only limits it’s recommendations to the same category/type of products, but if we incorporate item/user-based CF, along with content-based patterns it will also learn user patterns.

There are multiple ways to ensemble the two, simple weighted average or matrix stacking.
The weighted average approach involves taking the ‘weighted average’ of the rank of the items present in the list of recommendations provided by item-based CF and Content-based Recommendation for a user and then sorting it based on the new average rank generated.

In the second approach, we merge the item-user matrix with the item- TF-IDF matrix, column-wise to obtain a matrix in which each item vector is not only represented by every user vector but also by its

content vector. As a result, items which haven't been bought many times can be grouped with other products on the basis of their content vector space in the matrix.
Note now the size of the matrix can get really large so using sparse matrix representation can be very helpful (ex: csr_matrix kernel), various other approaches can also be used ie: subgrouping the matrix, or subgrouping the users.

Simple weighted average doesn't completely merge the two algorithms as it's solely based on the outputs, while matrix stacking adds completely new dimensions to our user-item space.
We tried both approaches and as expected matrix stacking was more promising.
Weighted Averaging helps to sort the recommendations better thus improving ARHR, while matrix stacking helps to generate better recommendations thus improving Hit Rate.
(Refer to Table 2)

Top 10 recommendations for our sample user by Hybrid Model

1. Riven: The Sequel To Myst
2. Tenchu 2
3. Street Fighter Ex 2 Plus
4. WWF Royal Rumble
5. Mega Man Anniversary Collection - PlayStation 2
6. Mischief Makers
7. Catwoman - Xbox
8. R-Type Command - Sony PSP
9. King of Fighters 2002/2003 - Xbox
10. Star Wars Episode I: Jedi Power Battles

Even though these recommendations are completely different from the rest, the user may like them as they are very similar to the games he has played in the past and some are even newer versions of the games he has played. The plus point is the diversity of the recommendations is good and there's no bias towards one category of games.

4) Metrics used for evaluating our models

The best way to rate an offline recommender is with Hit rate/ Average Reciprocal Hit Rate. If we are able to recommend an item which the user went and bought independently we can surely say we are recommending good items.

4.1)Hit Rate(HR):

$$\mathrm{HR} = \frac{\#hits}{\#users}$$

where *#hits* is the number of users for which the model was successfully able to recall the test item in the size-N recommendation list(ie: hit) and *#users* is the total number of test users.This metric tests the ability of a recommender system to give predictions that the user is *already known to be inclined towards*.

An HR value of 1.0 indicates that the algorithm is able to always recommend the hidden item, whereas an HR value of 0.0 denotes that the algorithm is not able to recommend any of the hidden items. A drawback of HR is that it treats all hits equally regardless of where they appear in the Top-N list. ARHR addresses it by rewarding each hit based on where it occurs in the Top-N list, which is defined as follows:

4.2)Average Reciprocal Hit Rate(ARHR):

$$ARHR = \frac{1}{\#users} \sum_{i=1}^{\#hits} \frac{1}{p_i}$$

where p_i is the position of the test item in the ranked Top-N list for the i-th hit. That is, hits that occur earlier in the ranked list are weighted higher than those occurring later and thus ARHR measures how strongly an item is recommended. The highest value of ARHR is equal to the hit rate and occurs when all the hits occur in the first position, whereas the lowest value is equal to HR/N when all the hits occur in the last position of the list.

We chose HR and ARHR as evaluation metrics since they directly measure the performance of the model on the ground truth data i.e., what users have already provided feedback for.

4.3)Diversity

$$\textit{Diversity} = 1\text{-}S$$

where S is the average similarity score between every possible pair of recommendations for a given user. This metric measures how 'diverse' the list of recommendations is. For example, if the final list contains a lot of products from the same brand, it represents low diversity. Ideally, a model should produce sufficiently diverse recommendations so that the user is introduced to newer avenues while maintaining a decent proportion of similar items as well.

4.4)Novelty:

Average popularity rank of recommended items. Higher means more novel. Helps us see if the model is only recommending "popular" items. A good model's novelty should neither be too high(always recommending low-popularity products) nor too low(only recommending highly popular products).
The maximum Novelty possible is 346 (because of the common number of ratings for items, many items have same "number of reviews", thus having "same popularity") which is when the model recommends items with only 1 rating while; Novelty of 1 is achieved when it recommends only the most popular item.

5)Results & Conclusion:

All these metrics were measured for a list of 1000 test users.

The models are analysed based on their corresponding values to each metric.
A random baseline model which recommends a set of random products to a user has also been created for comparison purposes.

Model	HR (%)	ARHR (%)	Diversity (%)	Novelty
Random Recommender	0.00	0.00	99.28	321.05
Content-Based Collaborative Filtering	6.36	1.85	83.59	317.83
Item Based Collaborative Filtering	16.6	6.00	97.40	266.11
User-Based Collaborative Filtering	3.50	0.83	98.82	27.57
Matrix Factorization	4.08	0.61	97.15	60.00
AutoRec	4.12	0.93	N/A	N/A
Hybrid Model with matrix stacking	**20.35**	**6.32**	98.07	303.78

Table 1

Comparison of the hybrid models

Model	HR(%)
Hybrid model with weighted average	19
Hybrid model with matrix stacking	20.35

Table 2

Item-based CF is more accurate than user-based CF as the similarity between items always remains fixed with time while users' tastes may change. Also, the number of items will always be way less than the number of users who visit the site, so this approach is computationally cheaper. User-based Collaborative Filtering doesn't perform as well as other models mainly because the user's taste changes with time: it is possible that the user may not continue to like a product in the future. As we aren't taking timestamps into account in our study, it becomes difficult to group users who have similar interests.

Another problem in user-based Collaborative filtering is it doesn't treat likes and dislikes separately meaning it may say two users are similar if they hate the common products also, but they may not necessarily like the same things which matters the most while providing recommendations. Finding common interests is way more important than finding common hates.

Factorization methods work well when one has to predict what a particular user will watch or click on next given his history (ex: *Youtube recommendations*), but for movie/product recommendations '*ratings prediction*' models aren't solving the problem in hand which is recommending Top N items. Even if we improve our RMSE score by a margin, the top N recommendations may still remain the same implying improved accuracy doesn't always lead to better recommendations. On paper, they may have better results but the top N results can be obscure as they are trying to solve the wrong problem.

One can see that the hybrid model outperforms the rest mainly because it takes care of new users/ items as a result it provides way better recommendations to all. The Matrix Factorization models don't perform at par with the others, although it gave a really good RMSE, implying that they are good as '*ratings predictions*' models but not as 'Top N' recommenders.

All models perform better than our baseline model in all metrics except novelty & diversity, thus implying that these metrics aren't the sole condition to judge a model but instead should be used to analyse how diverse our recommendations are. A good model should have a decent novelty and diversity score, a model with very high diversity may imply its just recommending random items and a model with low novelty score implies it suffers badly from popularity bias.

Most of the models suffer from popularity bias, ie: the majority of the recommendations are the products which are popular, the sole reason being the sparsity of the user-item matrix. While some users may like being recommended popular products, there are some niche interest users who may want non-popular items in their recommendations. The best way to analyse this is with A/B testing.

6)Future Work

In the future, we plan to improve our system by incorporating the following changes:

Addition of timestamps will help sort our recommendations better by giving more weights to the recently bought products, also possible splitting of users into 3 groups based on the percentage of popular products purchased, and then analysing them separately will help to solve the popularity bias.
We would also like to implement Restricted Boltzmann Machines(RBMs), as a version of it is being currently used by Amazon.

A special thanks to Aniket Patil(Data analysts at Vrythms) for mentoring us throughout the project

7)References

1. Amazon Review and metadata for Video Games by JC Mcauley, UCSD: https://nijianmo.github.io/amazon/index.html.
2. Justifying recommendations using distantly-labelled reviews and fined-grained aspects Jianmo Ni, Jiacheng Li, Julian McAuley *Empirical Methods in Natural Language Processing (EMNLP)*, 2019
3. Sundog Education Recommender Systems: http://sundog-education.com/RecSys
4. Lazyprogrammer GitHub repo: https://github.com/lazyprogrammer/machine_learning_examples/tree/master/recommenders
5. Sedhain, Suvash & Menon, Aditya & Sanner, Scott & Xie, Lexing. (2015). AutoRec: Autoencoders Meet Collaborative Filtering. 111-112. 10.1145/2740908.2742726.
6. Tran, Dai & Hussain, Zawar & Zhang, Wei Emma & Nguyen, Khoa & Tran, Nguyen & Sheng, Quan. (2018). Deep Autoencoder for Recommender Systems: Parameter Influence Analysis.
7. https://towardsdatascience.com/creating-a-hybrid-content-collaborative-movie-recommender-using-deep-learning-cc8b431618af
8. Apple Turicreate documentation: https://apple.github.io/turicreate/docs/api/turicreate.toolkits.recommender.html
9. Abdollahpouri, Himan & Mansoury, Masoud & Burke, Robin & Mobasher, Bamshad. (2019). The Unfairness of Popularity Bias in Recommendation.