Machine Learning with Recommender Systems: Using the Steam Dataset

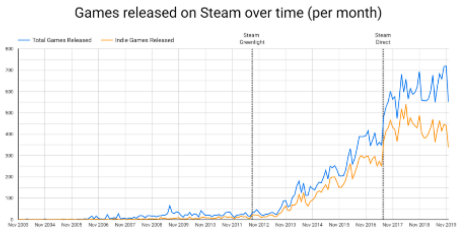
Lakshman Thillainathan[1]

Other group members:
Mahdi Rahman[2], **Yuan Surname3**[3]

Abstract. The internet as we know it is always expanding, with new media products to consume and more and more physical products being launched. We as humans are unable to look through every single item on the internet looking for what we want. What is a solution you may ask? Recommender Systems. These can be used to provide users with recommendations to products widing throughout many libraries. In this research project I will be going through the Steam webstore database in order to provide different algorithms that we may use in order to make the best for customer service.

1 Introduction

Recommender Systems are software tools and techniques that provide suggestions of items that are expected to be of the user's interests. The recommendations relate to various decision-making processes, such as what items to buy, what music to listen to, who you might match with(tinder) or even help you find jobs. Recommender Systems are usually dependant on the certain item you wish to summarise (CD, news, games, music), the core recommendation techniques are used to generate the recommendations and are customised depending on the type of item we are using. In my case I researched programs mainly focusing on the Steam library. Steam is a digital software distribution platform and is widely known for its video games. However, it has also recently, in 2008, made it possible for any software to be uploaded to the store. It was created with intentions of providing updates to Valve's games at the time, but eventually became the leading digital distribution platform thanks to its user-friendly GUI and the wide range of software available. Furthermore, the recommendation system is frequently updated for the user so they are able to always be provided with information of what they might enjoy.
Recommender Systems are useful in informing new users of what "everyone" enjoys and provide continuous users with suggestions of what they might like. It can be overwhelming when many of the platforms have several items that a user might like. In my case, I used the steam data set for this research, and there is millions of games and software on the site, that would be extremely difficult to navigate and find what you may like without the use of a recommendation system. Users who play games via steam are able to provide feedback for games. Compared to most rating based software which ask the user to rate the game from 1 to 10, Steam asks the user to write a review and a simple yes or no to whether you would recommend the game. If you browse games they show up as "Overwhelmingly positive reviews", "Very positive" and so on. Furthermore, the reviews are provided for more recent updates of the game too, depending on how long the game has been out. We call this data explicit feedback as it is based on user's ratings of the product. We can also take in data implicitly, by analysing the indirect behaviour towards a product, so how long a user has played the game and how often they view a game in the store page.



There are several ways in filtering the products for the user such as: Popularity-based modelling, Positive rate based modelling, content-based modelling, collaborative filtering modelling and spark ALS modelling. The main 2 ways I will be looking at are content based and collaborative for the steam data set. Collaborative Filtering systems analyse historical interactions alone, whereas Content-Based Filtering systems are based on profile attributes. Sometimes a Hybrid of both techniques can be used in order to provide suggestions, this is mostly used when there is a sparse amount of data and will be looked further into detail in this report. Recommender systems can also be used in real-world problems and is an active area of research, therefore being an important topic of Artificial Intelligence.

2 Background

Before the time of recommendation systems you would rely on other humans to recommend items for you. Having someone inform you of a movie they watched the night before and informing you of how good it was. Then you end up watching the movie and not liking it. This was the main issue of recommendations before the systems were invented. Recommender Systems are amazing in the fact they take in data from multiple users, and using the filtering techniques the machine will predict what you may like, not based on

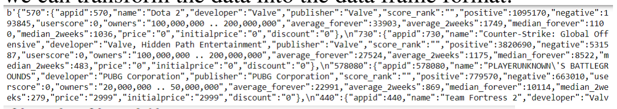
[1] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: lt4121g@gre.ac.uk
[2] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: mr2654m@gre.ac.uk
[3] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: yg1917c@gre.ac.uk

its opinion, but by the statistics surrounding the users and the item. Recommender Systems have evolved to fulfil the natural dual need for buyers and sellers by automating the generation of recommendations based on data analysis. [5] Tapestry was the first commercial recommender system to exist [6]. Upon reading [2] article, I have learnt that Tapestry was motivated by the huge stream of incoming documents via electronic mail at the time. Tapestry was known as the first collaborative filtering method used in a recommendation system. Collaborative filtering simply means that people collaborate to help one another perform filtering by recording their reactions to documents they had read. The reactions were stored as annotations in which they could be accessed by others' filters. The application proved useful and concluded that recommendation systems worked best when humans are involved in the filtering process.
There have been research conducted on an individuals traits and how they can be similar to other people. Evidence has shown that personality traits do not take effect on many aspects of behaviour, social interaction and mental and physical status. There was a study that showed that the personalities of a work team is related to the personality traits of the members within the team. There have been various new types of recommender systems built around personality traits over recent years, Roschinet al adopted a tool to construct the personality profile of a user through their written texts, but was not proven accurate. This sort of recommendation system has been used in dating software and music streaming software, but is yet to be applied on games.[1]

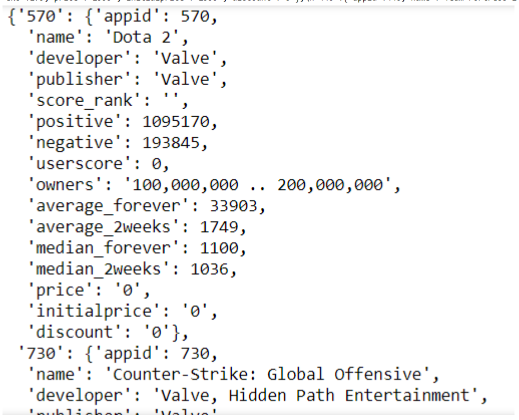
Collaborative filtering is based on the principle that similar people like similar things. We can put this in a matrix with one axes being the video games and the other axes being the users. We can then use the each of the users feedback on the game and provide it with a value from 1(disliked)and 5(liked). The matrix will then be left with some spaces where in which the recommender system will estimate what the user (looking for recommendations) would rate each of the games. So for example if a user has an identical taste to another user, we can infer that they will both rate a specific game highly. This can all be done using matrix factorisation of 2 smaller matrices.
Content-based filtering is based around splitting the user and games into 2 matrices. The users will be provided with categories such as age, country, gender etc. while the games will be provided with a different set of categories such as Horror, Platformer, Online etc. The 2 matrices are then linked depending on both the users similarities in taste of games and the games similarity to other games based on genre. Content-based filtering is better used in a hybrid situation.
Hybrid filtering is the process in which multiple filtering algorithms can be used to provide recommendations.[4] This is useful as collaborative filtering does not work for sparse data sets. So if a game has just been released or if a new user has registered into steam, it would cause an issue as there is not enough data of the user's preference to calculate recommendations. We could use content-based filtering until there is enough data to use collaborative filtering. We could also provide new users with games that are currently most popular.

3 Experiments and results

For us to test recommender systems and its capabilities we first need to select the correct packages to use with the data we provide. We used the NumPy, pandas, requests, sys to get the data that we need. To obtain the data I tried using the actual steam API, however the data format it provided gave a lot of unnecessary data and so I decided to use the SteamSpy data set as it was a better alternative. SteamSpy is a steams statistics service based on the web API provided by Valve. SteamSpy has to gather millions points of data daily to predict games sales and audience, therefore the data is not always accurate. Furthermore, the stats for some games with smaller samples are unreliable, such as games with any number below 30,000.
Our main objective is to recommend the user games according to the popularity, of the game, the similarity of the game description, the quality of the game, and the player's preference for the game, this will then give steam a higher degree of customer satisfaction.
The first steps is to gain access to the SteamSpy API, and creating a popularity-based recommendation algorithm. This will be useful for new users on the steam website, as they may not know what they want, so providing them with some of the most popular games owned by other previous users. Furthermore, these are the games that are mostly played throughout the whole of steam and therefore will provide a good whole time popular selection. We first need to extract the app id, game name, owners, and price, so we need to request the data then extract it into a JSON file. Then using panda we can transform the data into the data frame format:



```
b'{"570":{"appid":570,"name":"Dota 2","developer":"Valve","publisher":"Valve","score_rank":"","positive":1095170,"negative":1
93845,"userscore":0,"owners":"100,000,000 .. 200,000,000","average_forever":33903,"average_2weeks":1749,"median_forever":110
0,"median_2weeks":1036,"price":"0","initialprice":"0","discount":"0"},\n"730":{"appid":730,"name":"Counter-Strike: Global Off
ensive","developer":"Valve, Hidden Path Entertainment","publisher":"Valve","score_rank":"","positive":3820690,"negative":5315
87,"userscore":0,"owners":"100,000,000 .. 200,000,000","average_forever":27524,"average_2weeks":1175,"median_forever":8522,"m
edian_2weeks":483,"price":"0","initialprice":"0","discount":"0"},\n"578080":{"appid":578080,"name":"PLAYERUNKNOWN\'S BATTLEGR
OUNDS","developer":"PUBG Corporation","publisher":"PUBG Corporation","score_rank":"","positive":779570,"negative":663010,"use
rscore":0,"owners":"20,000,000 .. 50,000,000","average_forever":22991,"average_2weeks":869,"median_forever":10114,"median_2we
eks":279,"price":"2999","initialprice":"2999","discount":"0"},\n"440":{"appid":440,"name":"Team Fortress 2","developer":"Valv
```



```
{'570': {'appid': 570,
  'name': 'Dota 2',
  'developer': 'Valve',
  'publisher': 'Valve',
  'score_rank': '',
  'positive': 1095170,
  'negative': 193845,
  'userscore': 0,
  'owners': '100,000,000 .. 200,000,000',
  'average_forever': 33903,
  'average_2weeks': 1749,
  'median_forever': 1100,
  'median_2weeks': 1036,
  'price': '0',
  'initialprice': '0',
  'discount': '0'},
 '730': {'appid': 730,
  'name': 'Counter-Strike: Global Offensive',
  'developer': 'Valve, Hidden Path Entertainment',
  'publisher': 'Valve'
```

We can then easily sort the games in descending order according to the number of owners, we can infer that the games with the most owners are the most popular. You are able to change the parameters to find the top 10, top 20 or any n value of games. As you can see in the part 1 of the code.

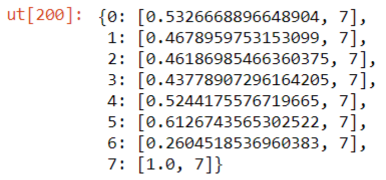
The reason as to why the Popularity-based recommendation is not the best is that it looks at the number of users that own a game, but does not look at the ratings of the game. This could very well mean that many users own a game that is heavily disliked, we wouldn't want to be recommending these games to new user's would we? To deal with this we can use the Quality-based recommendation algorithm. This algorithm as shown in part 2 allows the program to observe the number of positive reviews and negative reviews of each game. We can use simple maths by adding both values to find the total number of reviews, then dividing the number of positive comments by the total number of comments. For this bit of data we requested the top 100 games of all time from the API, this will provide accurate ratings. I also experimented with the code by looking for the most negative reviewed games by switching the parameter to negative reviews. However this only provides the worst of the top 100 and is simply a reverse of the positive rate model.

Part 3 is where it gets more interesting and we begin to use the more advanced algorithms that use AI. We need to analyse the descriptions of each game in order to get the content based recommendation system working. Unfortunately the SteamSpy API does not hold the data of the games descriptions, but it still is useful as we can use the appid to find the games on the steam website as each game has a unique games' IDs and have a common standard format on the website. We simply use the base HTTP format of the steam website (https://store.steampowered.com/app/ putting the appid number after the / in order to open the store page for the game. We then read in the descriptions and store them in a new DaraFrame Column. As mentioned previously the way in which content-based modelling works is that we look at say for example player A likes to play Call of Duty, and since CounterStrike Global Offensive is also a first person shooter game, we can assume that player A will like CounterStrike also. We then need to use the Beautiful Soup module to parse the HTML document in order to extract the description. We also need to strip the HTTP tags and formats as these are unnecessary bits of data, we use python regular expressions to combat this. We call this data cleaning. We will need to focus on the basic Natural Language Processing model TF-IDF in order to split the content texts and link to other descriptions.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

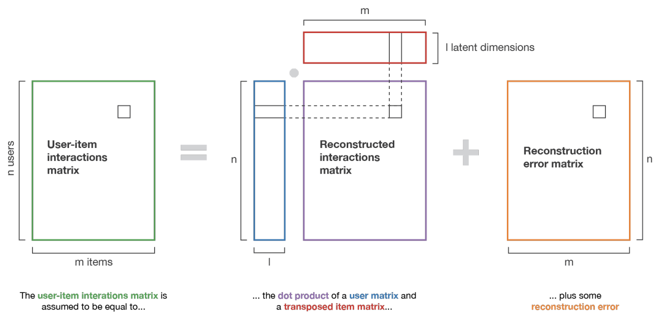
We calculate the weight by not only the frequency on this content text, but also the frequency of all the text set. We are then provided with TF-IDF values for each word in each game description and is stored in a matrix form. We then use linear-kernel to compute the games descriptions TF-IDF weights to other games' descriptions TF-IDF weights. We then multiply the vectors using linear-kernal. We then create a sequence of the results calculated, if the results are high then we can pair them up with the currently focused description. We then plot the top 10 recommendations for each game. We can see that the data puts itself as the first recommendation. Furthermore, with the example of "Portal", we can see that the 2nd recommendation is portal 2, which makes sense as they are sequels of eachother. Its an interesting phenomenon as there are relationships between certain games that are so strong the sequels are shown, as you can see by "Left 4 dead" and "Left 4 Dead 2".
Collaborative filtering recommendation system is the most accurate recommendation system I will be covering other than ALS. As you can see by part 4 of the code, this begins the same as others by requesting the data from the SteamSpy servers. However, an additional step is required as we need the data of users. Unfortunately the SteamSpy API does not provide an easy way to get this data, an alternative was to use the https://steamidfinder.com/ website. This website allows you to enter either the users' profiles SteamID or a random username. So in order to gain the data of random users' we had to enter in random steam names, aka entering random arbitrary names like "43". We then use the get function in the requests module to get the playing time information with 10 links. Since the steam review system works in a binary format (whether a user rates the game positive or negative), we do not have continuous ratings (1 to 5 stars). So in order to estimate the continuous rating, we can use the playtime of the users' to get an understanding to how much the user would rate a game. We can calculate the median data of playing time for each user and each game they play, we can then compare the median play time from the user with the median play time by all users. When we divide these values we should be left with a comparative median range from 1 to 5, which should help us fill in the empty spaces in the matrix. So to begin we get the data from the top 100 forever stats again, as they are the popular games and most of them have fairly high positive rates of over 90 percent. We can then assume that if a user's play time is bigger than the average play time for the game, then it is a good game, we can assume the continuous rating for the game is a 5, and if the users playing time is less than the median playing time but bigger than 0.8 multiply median playing game then the rating is assumed to be 4. We can also assume that if the users' playing time is less than 0.8 but bigger than 0.5, the rating is 3. If less than 0.5 but bigger than 0.1, the rating is assumed to be 2. If the users playing time is less than 0.1, we can assume that the playing time is 1. As the data provides a "no playing time" data, we can infer this to be a rating of



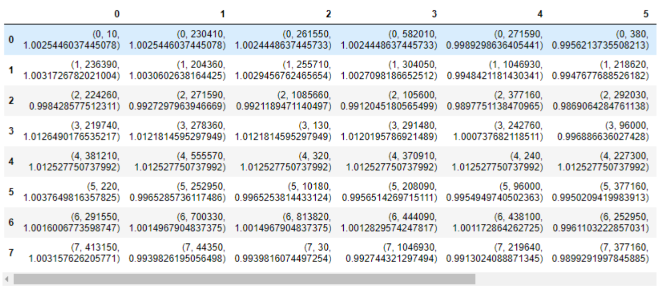
```
ut[200]: {0: [0.5326668896648904, 7],
          1: [0.4678959753153099, 7],
          2: [0.46186985466360375, 7],
          3: [0.43778907296164205, 7],
          4: [0.5244175576719665, 7],
          5: [0.6126743565302522, 7],
          6: [0.2604518536960383, 7],
          7: [1.0, 7]}
```

0.
We then use cosine regularity to get the results

To further push my study of recommender systems, I attempted to implement an ALS (Alternating Least Squares) recommendation model. This uses the matrix factorisation algorithm and the data we received from the collaborative based algorithm. The reason to use ALS is for a large scale database, as steam has a lot of users and games , we can use ALS efficiently.[3]


m
l latent dimensions
n users
User-item interactions matrix
m items
n
Reconstructed interactions matrix
l
Reconstruction error matrix
n
m
The user-item interations matrix is assumed to be equal to...
... the dot product of a user matrix and a transposed item matrix...
... plus some reconstruction error

For the results of the spark recommendations, we create an indivuals recommendation by the userid and the number of recommendations, then the part will give simulated ratings calculated by Spark ALS. We have to look through each public profile on the steam website by user ID one by one. This is an example of the output I received.



	0	1	2	3	4	5
0	(0, 10, 1.0025446037445078)	(0, 230410, 1.0025446037445078)	(0, 261550, 1.0024448637445733)	(0, 582010, 1.0024448637445733)	(0, 271590, 0.9989298636405441)	(0, 380, 0.9956213735508213)
1	(1, 236390, 1.0031726782021004)	(1, 204360, 1.0030602638164425)	(1, 255710, 1.0029456762465654)	(1, 304050, 1.0027098186652512)	(1, 1046930, 0.9948421181430341)	(1, 218620, 0.9947677688526182)
2	(2, 224260, 0.998428577512311)	(2, 271590, 0.9927297963946669)	(2, 1085660, 0.9921189471140497)	(2, 105600, 0.9912045180565499)	(2, 377160, 0.9897751138470965)	(2, 292030, 0.9869064284761138)
3	(3, 219740, 1.0126490176535217)	(3, 278360, 1.0121814595297949)	(3, 130, 1.0121814595297949)	(3, 291480, 1.0120195786921489)	(3, 242760, 1.000737682118511)	(3, 96000, 0.996886636027428)
4	(4, 381210, 1.012527750737992)	(4, 555570, 1.012527750737992)	(4, 320, 1.012527750737992)	(4, 370910, 1.012527750737992)	(4, 240, 1.012527750737992)	(4, 227300, 1.012527750737992)
5	(5, 220, 1.0037649816357825)	(5, 252950, 0.9965285736117486)	(5, 10180, 0.9965253814433124)	(5, 208090, 0.995651426971511)	(5, 96000, 0.9954949740502363)	(5, 377160, 0.9950209419983913)
6	(6, 291550, 1.0016006773598747)	(6, 700330, 1.0014967904837375)	(6, 813820, 1.0014967904837375)	(6, 444090, 1.0012829574247817)	(6, 438100, 1.001172864262725)	(6, 252950, 0.9961103222857031)
7	(7, 413150, 1.003157626205771)	(7, 44350, 0.9939826195056498)	(7, 30, 0.9939816074497254)	(7, 1046930, 0.992744321297494)	(7, 219640, 0.9913024088871345)	(7, 377160, 0.9899291997845885)

4 Discussion

I tried different solutions from the web with different codes, I also near the end experimented with the spotify database, as I realised the data received would be easier to handle. I had several issues using the official Steam API dataset as it would provide unnecessary data, however this does not mean that the SteamSpy dataset was the best of choices either. The steamSpy dataset was also difficult to work with as you could only request data once a day, this became problematic as I would be unable to retrieve data more than once in order to experiment. The solution I found was to use the Jupyter notebook, this proved efficient as you could run parts of the code separately. This meant I could retrieve data one day and still use the same data until I reset the data the next day. I also experimented with converting old data into text files, and then trying to read it in as a JSON file. My reason for not taking the next step and using the Steam API is that this was just an analysis of recommender systems and therefore did not need to go in depth with the Steam API. I also attempted using the steanidfinder website to create recommendations based off the user's friends list. I would take the steamids of the friends and use those ids instead of the random ids during the collaborative recommender systems.

5 Conclusion and future work

This project mainly only focused on less than 100 bits of data, and therefore would prove inefficient with large datasets, although I did go into the ALS recommender systems, I did not have too much luck getting it working. I think more research needs to be done on recommender systems as these can be proved fundamental not just for entertainment purposes. We could prescribe the correct medication to patients at pharmacies using recommender systems etc. Some more complicated algorithms such as PCA and SVD could be good additions to the created recommender system. Furthermore, we should consider the ratings based on binary ratings and continuous ratings. I also see steam implementing further recommendation systems based on the implicit feedback received by the user.

ACKNOWLEDGEMENTS

I would like acknowledge that most of the presentation was done by me, and also that the code may not work if attempted to run more than once as it is unable to request to the steamSPy servers more than once every 24 hours.

REFERENCES

[1] Paul Bertens, Anna Guitart, Pei Pei Chen, and África Periáñez, 'A machine-learning item recommendation system for video games', in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–4. IEEE, (2018).

[2] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry, 'Using collaborative filtering to weave an information tapestry', *Commun. ACM*, **35**(12), 61–70, (December 1992).

[3] Balázs Hidasi and Domonkos Tikk, 'Fast als-based tensor factorization for context-aware recommendation from implicit feedback', in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 67–82. Springer, (2012).

[4] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich, *Recommender systems: an introduction*, Cambridge University Press, 2010.

[5] Prem Melville and Vikas Sindhwani, 'Recommender systems.', *Encyclopedia of machine learning*, **1**, 829–838, (2010).

[6] Paul Resnick and Hal R Varian, 'Recommender systems', *Communications of the ACM*, **40**(3), 56–58, (1997).