Capstone Project 2 Personalized Video Game Recommendation System for reddit user

Zongkai Wu

1. Introduction:

Video game industry is a boosting business, and there's large profit if you can successfully recommend the right game to the right person. Various recommendation systems for video games have been built for gaming platforms or gaming forums, but the problem is those platforms can only recommend games to gamers that already visited those platforms. If we can recommend games to a broader audience without them visiting a gaming specific platform, it will be largely beneficial for both gaming companies and gamers, as well as forum platform providers. For gamers, they can get recommendations for games they most likely will be interested in and get to play some awesome games they will like. For gaming companies, they can push the correct game to correct gamers in order to increase their profit, as well as analyze the market environment for specific gaming genres. For forum platforms, they can increase their ads efficiency by providing personalized games recommendations

In this project, we intend to use reddit as a platform for video game recommendation. Reddit as a board topic posting platform, has many different subreddit to post various posts to discuss certain topic with other viewers. There are many subreddit directly related to one certain game, one certain game series, or a genre of game. By analyzing user-subreddit interaction, we will be able to build information to analyze what reddit user like to discuss about games, then by combing a complicated user-subreddit matrix, we can calculate which game should be recommend to each user.

For data retraction, reddit has a very well-built API allowing us to retract user's comment and submission information to analyze user's information.

For game information, we use Rawg API to retract 394k game's information including name, genre, platform, tag and metacritic scores.

For the final product is to build a website that users can input their or other’s reddit user name, the website will analyze their comments and give a list of the user's current favorite game and recommended game for the user.

2. Data Wrangling:

The most unique feature of this recommendation engine is we predict user’s preference without the user putting any directive input regarding the game. The only information we can get for each user is the information of their comment and submission on the reddit platform. So, there are three key steps to build the connection between users and subreddit: 1. Collect game information 2. Connect subreddits that are related to those games. 2. Choose users from subreddits with enough information to make recommendations, and collect comments from those users. We will go over each step one by one.

2.1 Collect game information

This part of code can be viewed from links below:

https://github.com/zokwu/Capstone2_vgr_reddit/blob/master/Rawg_game_data.ipynb

https://github.com/zokwu/Capstone2_vgr_reddit/blob/master/Rawg_eda.ipynb

There are many websites that have rich game information. Due to the nature of this study, we’d like to use a website which provide detail games information for free with easy to use API to retract data. Rawg is a large video game database and they shared 350,000+ games, search and machine learning recommendations with the world. Here we used Rawg game API to retract a total of 394,321 entries of games with 28 columns including very detail gaming information such as name. genre, tags, platforms, release time, Metacritic score, and added user from Rawg.io.

Over 390k games is too much to process, and due to limit computational power, we have limit our games list to a much smaller range, at least for the scope of this project. After a few testing, we decide to select games with more than 1000 added and Metacritic socre higher than 85, which leave us 232 games. It seems to be a small number, but due to the large amount of reddit comments we can find associate to each game, the amount of data we need to process will scale up very fast.

2.2 **Connect subreddits to games**

This part of code can be viewed here:
https://github.com/zokwu/Capstone2_vgr_reddit/blob/master/Subreddit_matching.ipynb

This is one of the key steps of this project, and the most important step to differentiate this project from subreddit recommendation engines. By collecting game metadata from Rawg API, we were able to acquire rich background game information such as platform, genre, ratings and keywords. However, to connect each game to the user requires building connections between games and reddit comments. It's possible to connect games and each post on reddit, but it requires large compute power to finish the search and classification to get enough useful data. Another much easier approach is to link a game to a subreddit, then we assume that all the posts in this subreddit are about this game.

When searching subreddit with the game's name, we make a few adjustments to improve accuracy and find more relevant results: 1. Separate game's full title to main title and subtitle using the defined function sep_title, and search with both full title and main title. Without this step, some games with very long subtitles won't return any search result due to too many words, such as "The Witcher: Enhanced Edition Director's Cut". 2. If roman number is detected in title, replace roman numbers with Arabic numbers and search with both roman numeral title and Arabic numeral title. This is to avoid missing some subreddit which convert game title's number to Arabic numeral.

With the searching strategy above, we store top 4 search results selected by the reddit API search engine, then we build our own scoring mechanism to pick the best subreddit that can present the game. The scoring is based on Levenshtein distance, using fuzzywuzzy package. This grading was achieved with self-defined function grade_game_sub. Levenshtein distance comparison is simply comparing how much overlap the game title and subreddit titles are, but sometimes this main return false subreddit with high score. Example include the walking dead games, which if searched, will return r/walkingdead and r/walkingdeadgame, both result shows equally high score due to game title match the subreddit tile. But since we are interested in gaming subreddit, the second one is clearly the subreddit we are more interested in. Therefore, in the grade_game_sub function, we also define a whether a subreddit can receive bonus for including words such as 'games', 'video games', while provide a penalty for subreddit with words such as 'movies', 'tv shows'. With this additional adjustment, we were able to greatly improve the matching rate and return mostly game related subreddit for each game.

After considering penalty and bonus, another issue we need to figure out is subreddit for specific game or for a game series. For example, ‘The legend of Zelda: Breath of the wild’. By separating main title and subtitle, search and grading, we were able to get two winners based on matching alone, one is r/zelda, a subreddit for Zelda series. Another is z/botw, a subreddit delicate to discuss Breath of the Wild, this specific entry of game. However, There are much more posts on r/Zelda compared to r/botw, so although r/botw is a better match by itself, analyzing it won’t get us very rich information due to its low traffic, while r/Zelda while can discuss all sorts of games, it normally focused on the most recent entries or most popular entries, so if we match this game to r/Zelda, we are still getting relevant information about user’s opinion on this game, but also able to get more information. To help make those kind of decision, I manually tested a few criteria and come up with the following equation to grade a score for each game-sub pair. And the one with the highest score will be the matching result.

score = title_score * 1.7 + des_score * 1.2 + np.log(subscribers)*8 - penalty * penalty_point + bonus * bonus_point

title_score is title based matching score, des_score is description based matching score. From test result, subreddit for specific game normally will mention full subtitle in their description, which will lead to higher des_score. However, if this subreddit’s subscriber is too low, it will score low on the term np.log(subscribers)*8, which will end up still losing the final score. Last two terms are bonus and penalty we discussed above. Bonus_point or penalty_point are manual decided constant, while bonus or penalty can only be 0 or 1. Since we don’t really have a clear answer which subreddit is the best match for each game, I have to manually tune this equation, keep testing until I have a function that return most satisfying result.

2.2 Choose users from subreddit

This part of code can be viewed in the link below:

https://github.com/zokwu/Capstone2_vgr_reddit/blob/master/Get_comment.ipynb

After the searching subreddit parts, we were able to get a list of subreddit we’d like to collect information from. But there were too many users and too many comments on each subreddit each day, it’s impossible to collect or analyze it all. Therefore, we need to select the right audience for efficient analyzing suitable for the scale of this study. Since we are trying to build an interaction matrix between subreddit and user, we’d like to have users who comment on more than one game related subreddits. Here we select user who have commented on at least 3 game related subreddit for our analyze.

While we initially use reddit's official API praw for information retraction, it limit the amount of traffic we can get within a certain time, and won't return any result if we pass this limit. Due to the nature of this study required large amount of posts and comments to be retracted, I have to use pushshiftAPI psaw, which is very similar to use but don't have the traffic limit

Reddit API allow us to retract information based on subreddit or user, to complete the get comment process, we developed 2 sub-function to finish it step by step. First is user_info, which take one subreddit from our gaming subreddit list, and get all the users who has commented on this subreddit for a month (April 2020). To avoid too much content for super popular subreddit, we set a user limit to 500, so it will stop either fully collect users for a month, or reach 500 users limit before that. This function will return a list of user to get_comment function. Second step used function user_sub_comment, which goes over comments of one user for all subreddit within our gaming subreddit list under a limit of 100 comments. This should certainly be much more, but due to lack of computational power from my personal laptop, we limit it to 100 so we can finish it within a reasonable time. After retracting 100 comments from this user, we count how many subreddit this user had commented on. If there're more than three, we keep this user and return a list of game related comment to get_comment function.

This is the optimized step for this process, at first we use the build-in function from psaw api api.redditor_subreddit_activity to count the amount of subreddit for each user before collecting those comments. However, this build-in function is very slow as it's essentially collect all those comments for the user, run calculation and return a tallied result, after we decided selected user with this method, we have to collect those comments again, which means we are searching comments for each user twice. By clearing space after search for each user, it doesn't cost any additional space and it's much faster than the original idea we had for this process.

2.3 Summary

After the above three steps, we now have 3 useful dataframes we can use to build our recommendation engine.

1. Game_1000_85.json: This dataframe contains 232 games picked from section 2.1 for this study, with detail game information such as platform, genre, release and genre.

2. df_sub.csv: This dataframe contains matching subreddits for all 232 games from the above dataframe

3. df_comment: This dataframe contains total of 710090 comments on gaming subreddit from df_sub.csv, which only contains comment from users who has commented on three or more gaming subreddits.

3. Recommendation Engine:

Now we have collected all three dataframes, we finally have enough information to build gaming recommendation engine from reddit.

3.1 Recommendation Engine Study

Traditionally, building recommendation engine require on explicit data provided from customer like ratings to songs or movies they have viewed. With enough user-item interaction, we would be able to build a M x N matrix with M users on each row and N items on each column, and fill known explicit data in. Then we can factorize the interaction matrix with k features to two new latent matrices (M x k) (k x N) by minimizing the following Loss function[1]:

$$L = \sum_{u,i \in S} (r_{ui} - \mathbf{x}_u^\intercal \cdot \mathbf{y}_i)^2 + \lambda_x \sum_u \|\mathbf{x}_u\|^2 + \lambda_y \sum_i \|\mathbf{y}_i\|^2$$

We can use ALS (Alternating Least Squares) or SGD (Stochastic Gradient Descent) to minimize the Loss function to get optimized factorized matrix, then we can predict any unknown user-item interaction by multiplying user's feature vector to item's feature vector.

However, it has been harder to get explicit data nowadays. For many website, higher percentage of user would only interact with an item without leaving a valued rating, therefore it's hard to use the above method. Instead we can rely on user's past behavior without requiring the creation of explicit data[2]. We called this Weighted Regularized Matrix Factorization (WRMF). Compared to explicit matrix factorization, the Loss function become the following[3]:

$$L_{WRMF} = \sum_{u,i} c_{ui} \left(p_{ui} - \mathbf{x}_u^{\intercal} \cdot \mathbf{y}_i\right)^2 + \lambda_x \sum_{u} \left\|\mathbf{x}_u\right\|^2 + \lambda_y \sum_{u} \left\|\mathbf{y}_i\right\|^2$$

Two key difference is instead of optimizing rating r_{ui}, we are optimizing p_{ui}, which is preference. $p_{ui} = 1$ if there was interaction between user-item pair and $p_{ui} = 0$ if there's none. Other difference is c_{ui}, which is confidence matrix, and it roughly describe how confident that user in fact have preference over item.

Now let's look back at our project to see if we can simply use WRMF to build our video game recommendation engine. We do have rich implicit data in the form of comments, but that's only the interaction between user and subreddit. Even though we have build interaction between games and subreddit, but if we don't provide additional information for those games from game dataframe, we were simply renaming subreddit to game name and build a subreddit recommendation engine, which is not bad actually but clearly not the goal here. What we need here is add side information to each game, which will start to distinguish this recommend engine to subreddit recommendation, as the side information for all the games won't be linked to each subreddit. Besides we can also include word analyze result for users and items as additional side information.

Originally we only have one matrix to factorization, M x N matrix for user-item interaction. Now we included confidence matrix, user side information matrix and item side information matrix[4]. Our goal now it to optimize matrix factorization for all four matrix, which is almost impossible to solve if using ALS method we mentioned above. Two alternative method can be used to solve matrix factorizing, Bayesian Personalized Ranking[5] (BPR) and Weighted Approximate-rank Pairwise[6] (WARP). The idea is centered around sampling positive and negative items and running pairwise comparisons. WARP is quite similar to BPR: you sample a positive and negative item for a user, predict for both, and take the difference. In BPR you make the SGD update with this difference as a weight. In WARP, you only run the SGD update if you predict wrong, for instance, you predict the negative item has a higher score than the positive item. If you predict correctly, then you keep drawing negative items until you either get the prediction wrong or reach some cutoff value[4]. Since this is exactly the problem we need to solve, we will compare both BPR and WARP in this project using LightFM package. LightFM is a Python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback, including efficient implementation of BPR and WARP ranking losses. It's easy to use, fast (via multithreaded model estimation), and produces high quality results.

3.2 Build Recommendation Engine with LightFM

This part of code can be viewed here:

https://github.com/zokwu/Capstone2_vgr_reddit/blob/master/Recommender%20with%20lightFM.ipynb

From the current data, we have 3720 user commented on 131 subreddits. Initially we tried to find subreddit for 232 games and got 133 subreddits, this is due to some game can't find or don't have matching subreddit, or the subreddit is too quite so didn't selected in our dataset. Another reason is lots of game match into the same subreddit, either due to same game of different edition, or a series of game shared a common subreddit.

We discussed whether to use common series subreddit or specific subreddit for that specific game, and from my experience, most of the time people discuss about the most current version of the game in the main subreddit instead of into the specific subreddit, therefore we decide to pick series subreddit from our search engine. We could be more specific about which game the user is actually discussing by searching submission key word or use time stamp to estimate, but it seems to be too complicated and may not be accurate, so for our first model training we just use relative easy method.

The final product is to recommend game to user, so subreddit is just a mean in the middle, we need to connect video game directly to user for our matrix in order to train the model. There're two possible way to build this connection without more complicated searching and comment analysis:

Pick the most popular game from the games a subreddit linked with, and return to the user. This way just simply assume user has only player the most popular game, and other game in the series is free to recommend to user. Advantage is we can recommend game from user's favorite series without kicking them out, but disadvantage is it's assumption could be wrong for a lot of cases

Match user's reaction to the subreddit to all games, this will eliminate recommendation to games from same series as it will assume user has played them all, but the advantage is recommended game will more likely to be fresh and less likely to be user's played games.

We currently have 3 set of train, test data split, m1 which is non-weighted implicit user-game interaction data which label interaction with 1 and non-interaction with 0. m2 use total number of comment as weight for each interaction. Difference between fm and m2 is the way data splited, m2 use customized function to make sure each user in test set have at least k item interacted to make precision@k meaningful, while fm dataset randomly pick test set. Both can be used to compare auc to evaluate model, but precision@k is not meaningful for fm dataset. However, the advantage of fm dataset is it avoid bias in test set for user interacted at more games.

So if we decide to use auc as our evaluation metrics, fm_train, fm_test should be more non-biased dataset to use. If we decide to use precision@k, then m1 or m2 are the only meaning option.

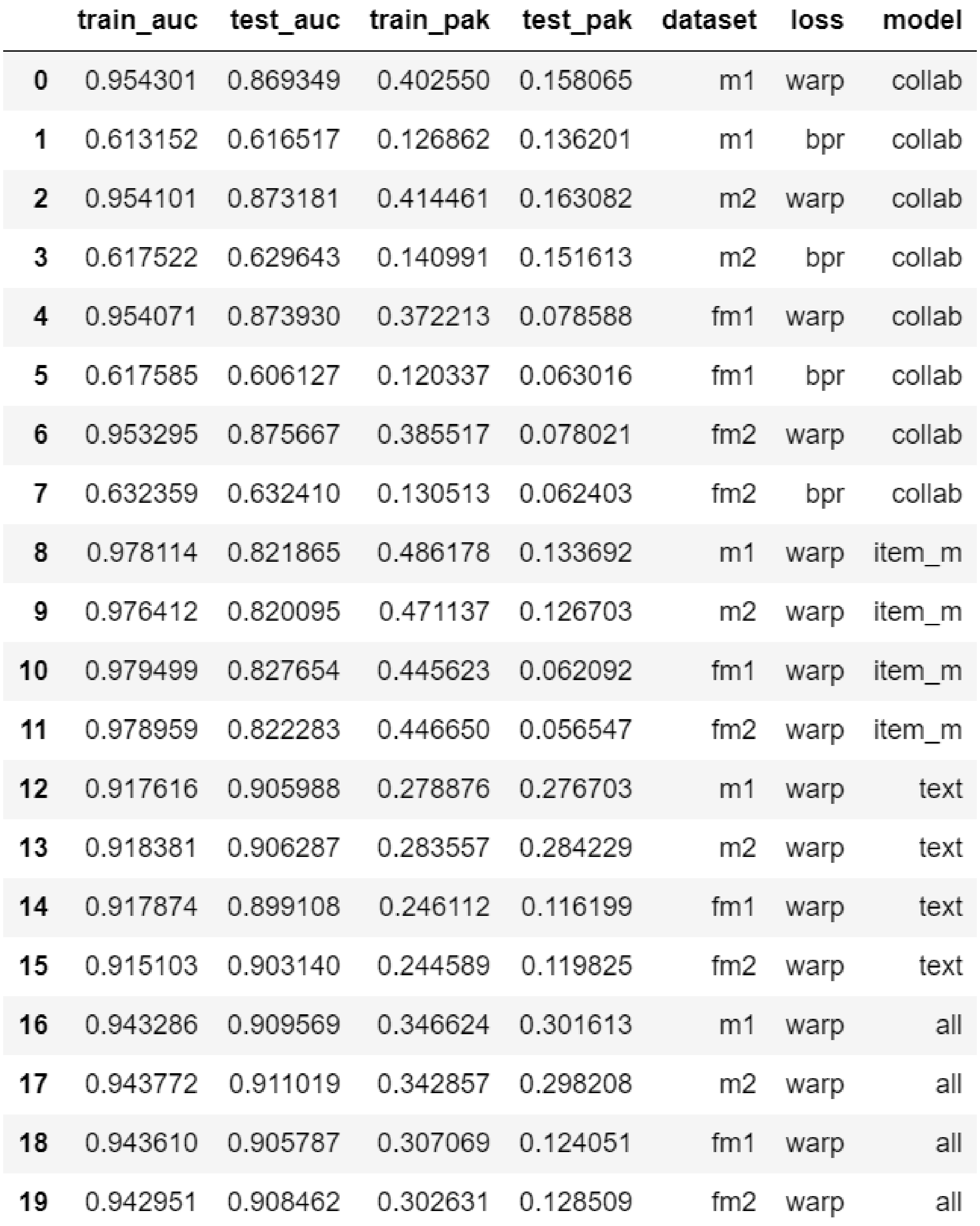
So before introduce user_text data, game_text data and game meta data, we would like to use the above three train_test split to decide a few things:

1 Which evaluation method to use?

2 Which dataset to use?

3 Which loss function (warp or bpr) and learning schedule (adagrad or adadelta) to use?

Since we can't decide any of the above, we decide to store all result into a dataset and evaluate result in the end. Since we are using a relative small dataset, cover all options and train model is not that costly. Once we chose and optimize the best model, we can just use this one for scale-up version of this recommender.



	train_auc	test_auc	train_pak	test_pak	dataset	loss	model
0	0.954301	0.869349	0.402550	0.158065	m1	warp	collab
1	0.613152	0.616517	0.126862	0.136201	m1	bpr	collab
2	0.954101	0.873181	0.414461	0.163082	m2	warp	collab
3	0.617522	0.629643	0.140991	0.151613	m2	bpr	collab
4	0.954071	0.873930	0.372213	0.078588	fm1	warp	collab
5	0.617585	0.606127	0.120337	0.063016	fm1	bpr	collab
6	0.953295	0.875667	0.385517	0.078021	fm2	warp	collab
7	0.632359	0.632410	0.130513	0.062403	fm2	bpr	collab
8	0.978114	0.821865	0.486178	0.133692	m1	warp	item_m
9	0.976412	0.820095	0.471137	0.126703	m2	warp	item_m
10	0.979499	0.827654	0.445623	0.062092	fm1	warp	item_m
11	0.978959	0.822283	0.446650	0.056547	fm2	warp	item_m
12	0.917616	0.905988	0.278876	0.276703	m1	warp	text
13	0.918381	0.906287	0.283557	0.284229	m2	warp	text
14	0.917874	0.899108	0.246112	0.116199	fm1	warp	text
15	0.915103	0.903140	0.244589	0.119825	fm2	warp	text
16	0.943286	0.909569	0.346624	0.301613	m1	warp	all
17	0.943772	0.911019	0.342857	0.298208	m2	warp	all
18	0.943610	0.905787	0.307069	0.124051	fm1	warp	all
19	0.942951	0.908462	0.302631	0.128509	fm2	warp	all

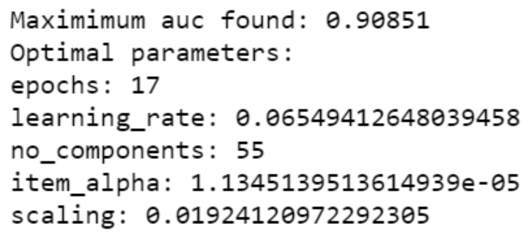
Above table shows all result from model training. From collaborative comparison, bpr is clear inferior to warp as loss function, we use only warp for future test to save computation power. Next we start to add side information into this model. First we take rawg game dataset and retract game meta data as item meta data for recommender model.

For game dataset, we decide to use three features as side information, which are platforms, genres and tags. Use function convert_item_feature, we were able to convert those information to a flat item feature dataset.

Row 8 to 11 shows result and comparison of collaborative only and include item meta data result. It seems with additional side information only result for train data shows improvement but for test data all shows lower score. This is due to score is based on subreddit interaction, not really user-game interaction. This is also the biggest challenge of this project, as it's hard to evaluate without asking each user which game they really liked to play. Nonetheless, we moved on to the last part and get more side information for both users and games using comments. With the help of sklearn TfidVectorizer, we extract text meta data for both user and game as side information. Next we use those information and interaction table only, without game meta data, to train models with different dataset.

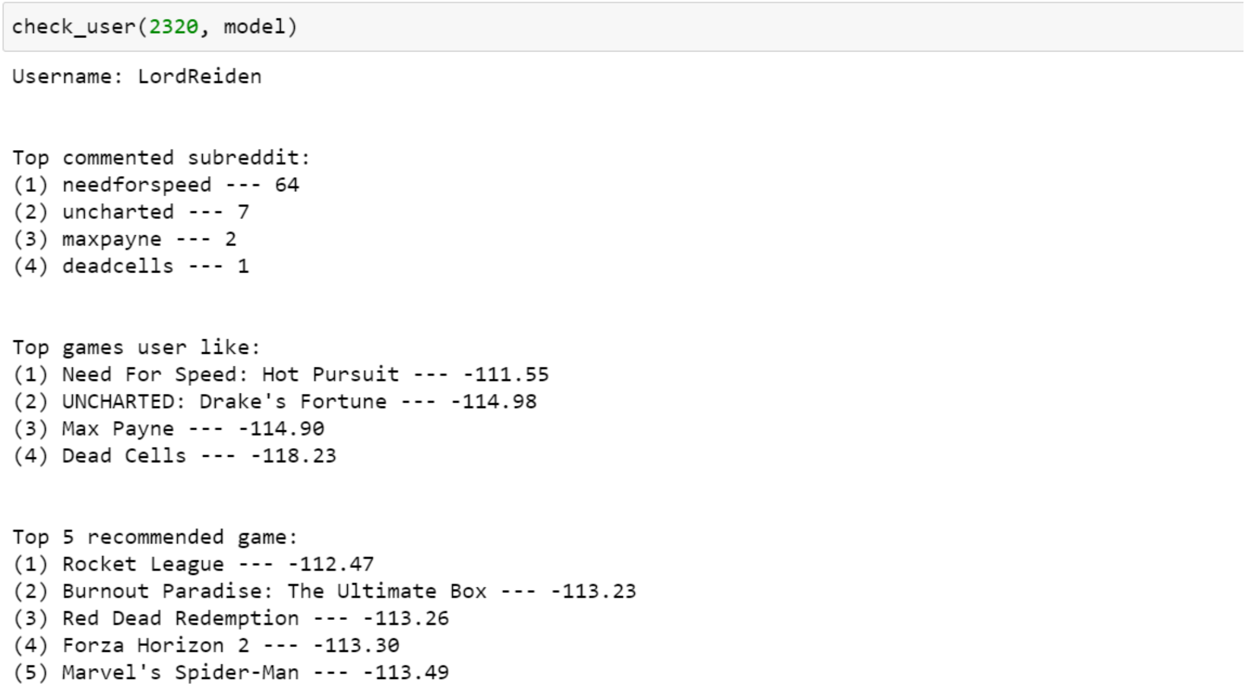
From row 12 to 15 we finally see some improvement, this is to be expected, as subreddit comment is very import meta data for subreddit recommender system. Final step is to combine game meta data with subreddit comment data as combined side information.

With all side information included, we can see although blindly add game meta data as side information decrease auc score for all models, add game meta data in addition to text data actually improve the score a little. This is what we hope to see as we believe game meta information should help make this recommender more personal, also distinguish it from subreddit recommender system. So even the 'all' model shows worse result, we would still use it to make recommendation to games without corresponding subreddit more accurate. Now we’ll use the best model and move on to hyperparameter tuning. Since we will use fm2 dataset, we can only use auc as our standard for hyperparameter tuning, and we use random forest_minimize from skopt to optimize hyperparameter including epochs, learning rate, number of components, item_alpha and user_scaling. Result can be seen in the picture below. From above result we can see optimization didn't improve result on testing set, after 50 calls of optimization auc score is even lower than out-of-box result. Therefore we just used original parameter as our model.



```
Maximimum auc found: 0.90851
Optimal parameters:
epochs: 17
learning_rate: 0.06549412648039458
no_components: 55
item_alpha: 1.1345139513614939e-05
scaling: 0.01924120972292305
```

Now we have our final model, let’s manually check some recommendation for some users and judge the taste of the machine (and the user) ourselves.



```
check_user(2320, model)
```

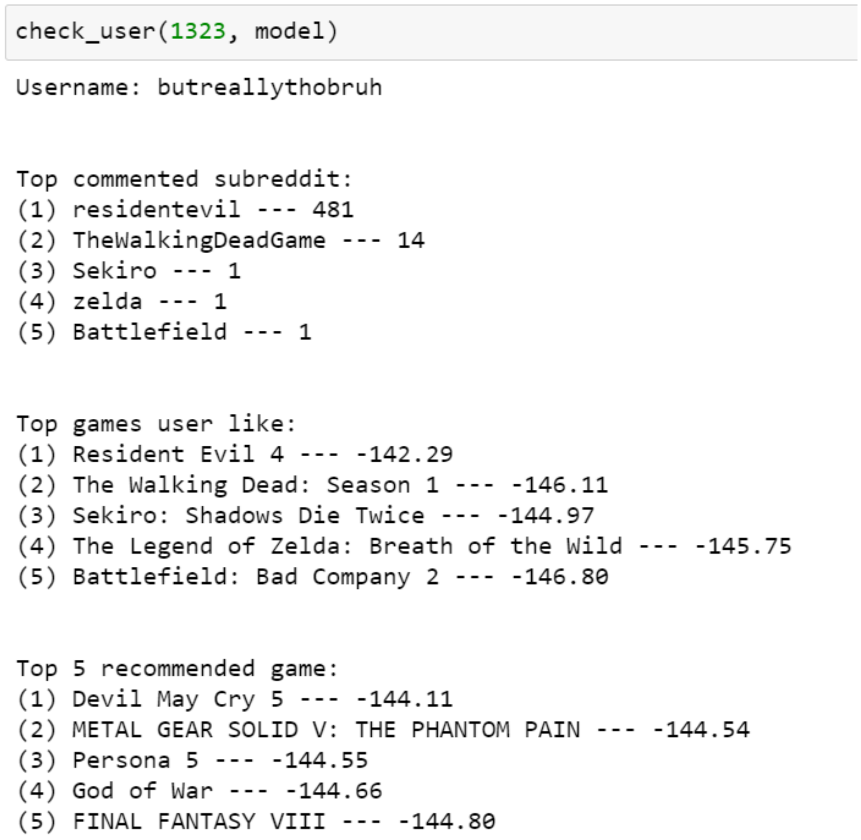
```
Username: LordReiden


Top commented subreddit:
(1) needforspeed --- 64
(2) uncharted --- 7
(3) maxpayne --- 2
(4) deadcells --- 1


Top games user like:
(1) Need For Speed: Hot Pursuit --- -111.55
(2) UNCHARTED: Drake's Fortune --- -114.98
(3) Max Payne --- -114.90
(4) Dead Cells --- -118.23


Top 5 recommended game:
(1) Rocket League --- -112.47
(2) Burnout Paradise: The Ultimate Box --- -113.23
(3) Red Dead Redemption --- -113.26
(4) Forza Horizon 2 --- -113.30
(5) Marvel's Spider-Man --- -113.49
```

For user LordReiden, he commented on needforspeed, uncharted, maxpayne and deadcells. All four related game has been recognized correctly. Games recommended to the user include Rocket League, Burnout Paradise, Red Dear Redemption, Forza and Marvel’s Spiderman. Based on his taste to uncharted and maxpayne, I think RDR is most likely a game this user will enjoy. Since he like racing game need for speed, Rocket League and Forze, two of the most popular racing or car related games, sounds like good recommendation to me too.



```
check_user(1323, model)
```

```
Username: butreallythobruh


Top commented subreddit:
(1) residentevil --- 481
(2) TheWalkingDeadGame --- 14
(3) Sekiro --- 1
(4) zelda --- 1
(5) Battlefield --- 1


Top games user like:
(1) Resident Evil 4 --- -142.29
(2) The Walking Dead: Season 1 --- -146.11
(3) Sekiro: Shadows Die Twice --- -144.97
(4) The Legend of Zelda: Breath of the Wild --- -145.75
(5) Battlefield: Bad Company 2 --- -146.80


Top 5 recommended game:
(1) Devil May Cry 5 --- -144.11
(2) METAL GEAR SOLID V: THE PHANTOM PAIN --- -144.54
(3) Persona 5 --- -144.55
(4) God of War --- -144.66
(5) FINAL FANTASY VIII --- -144.80
```

User butreallythobruh Seems to prefer resident evil above everything else, while prefer horror action game and RPG games. Recommendation include Devil May Cry, which is an horror action game, would definitely get user's interest if he/she give it a shot. All other game seems to lean toward action RPG, as four of the five games user player are RPG games, though the direction might be not quite the same, they are all good RPG games worth a shot for the user in my opinion.

Conclusion

For this project, we are able to build a recommendation engine based on any user's reddit comment history to recommend video games to user's taste. We collect game information from Rawg database, retract reddit user, comments information using reddit API and pushshift. We connect video games to subreddit based on Levenshtein distance, and create a customize score system to pick the most suitable subreddit for each game. In the end, we use LightFM package WARP model to train recommendation engine, with all the side information retracted from game meta data and user's comments, and we get 94.3 and 90.8 as auc score for train set and test set, which is very solid result. In the future we have plan to increase the scale of this project and potentially build a website for user to input their reddit user name and get a list of recommendation video games.

Reference

1. https://www.ethanrosenthal.com/2016/01/09/explicit-matrix-factorization-sgd-als/

2. Yifan Hu, Collaborative Filtering for Implicit Feedback Datasets, link:
http://yifanhu.net/PUB/cf.pdf

3. https://www.ethanrosenthal.com/2016/10/19/implicit-mf-part-1/

4, https://www.ethanrosenthal.com/2016/11/07/implicit-mf-part-2/

5. Steffen Rendle, BPR: Bayesian Personalized Ranking from Implicit Feedback, link:
https://arxiv.org/ftp/arxiv/papers/1205/1205.2618.pdf

6. Jason Weston, WSABIE: Scaling Up To Large Vocabulary Image Annotation, link:
http://www.thespermwhale.com/jaseweston/papers/wsabie-ijcai.pdf