Group member: Tom Higgins, Syed Qavi, Xintong Li

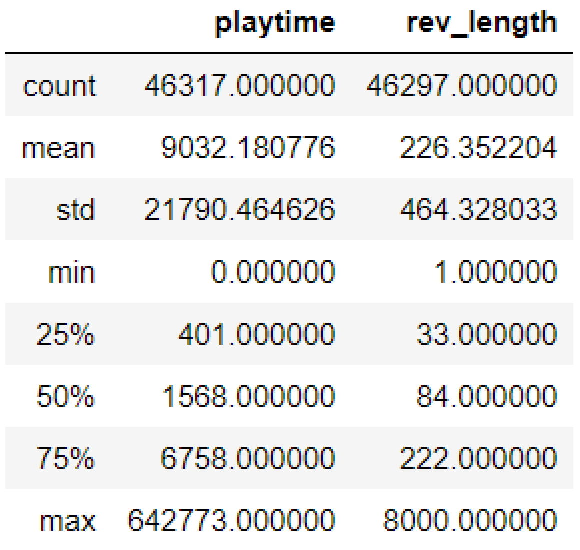
Presentation: https://youtu.be/QxW9CALbD48

Project Description

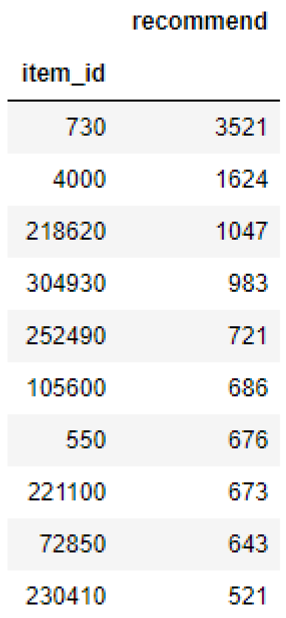
Our goal for this project is to optimize recommendation algorithms for use on the Steam dataset. In order to do this, we've used the Surprise package's implementation of several different algorithms and optimized their parameters with a grid search and calculate accuracy measures. We also did some work on a non-personalised content-based recommender using review text. The end goal is to figure out which methods we can use to best recommend games to interested users.

Data Description

Only two numerical variables from the dataset. Statistical summary on the left and top ten liked games on the right which shows the skewness of the dataset.



	playtime	rev_length
count	46317.000000	46297.000000
mean	9032.180776	226.352204
std	21790.464626	464.328033
min	0.000000	1.000000
25%	401.000000	33.000000
50%	1568.000000	84.000000
75%	6758.000000	222.000000
max	642773.000000	8000.000000

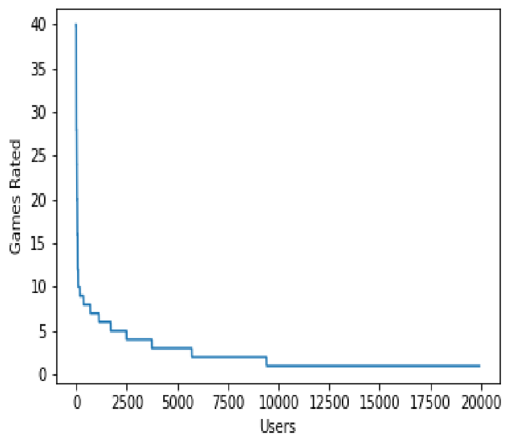


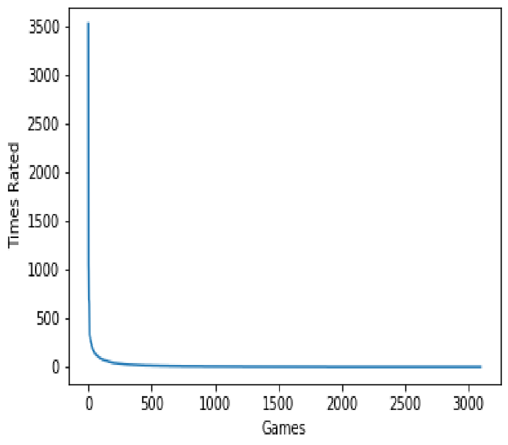
	recommend
item_id	
730	3521
4000	1624
218620	1047
304930	983
252490	721
105600	686
550	676
221100	673
72850	643
230410	521

The dataset was taken from Recommender Systems Data Set Repository by Julian McAuley at UCSD: https://cseweb.ucsd.edu/~jmcauley/datasets.html#google_local. The dataset consisted of selection of game reviews from Australian Steam users and user, item data. We merged the version one of the game reviews with the user and item data. We removed columns that we didn't think was relevant like the url, last edited date, steam id, and columns indicating whether the review was helpful or funny. The version we worked with contains user and item ids, along with the text of each review, the amount of time that user played that game for, and a field indicating whether that user liked that game. The code for this can be found in the Data_Exploration_Wrangling.ipynb. As with many similar datasets, this is a very sparse dataset. The recommendations column is a very uneven class; about 88% of the review grades are Like/True. The dataset consists of 46317 rows, number of unique users is 19913 and the number of unique items/games is 3097. The recommended column was used as a rating where we used a binary rating system by converting the true/false to 1/0. The average playtime was around 9032 hours and average review length was 226 words. Here is a sample of our cleaned data:

	user_id	item_id	review	recommend	item_name	playtime
0	76561197970982479	1250	Simple yet with great replayability. In my opi...	True	Killing Floor	10006
1	76561197970982479	22200	It's unique and worth a playthrough.	True	Zeno Clash	271
2	76561197970982479	43110	Great atmosphere. The gunplay can be a bit chu...	True	Metro 2033	834
3	js41637	251610	I know what you think when you see this title ...	True	Barbie™ Dreamhouse Party™	84
4	js41637	227300	For a simple (it's actually not all that simpl...	True	Euro Truck Simulator 2	551

A plot showing how many reviews individual users tend to be responsible (left):
A plot showing the frequency of reviews per game (right):





In general there are a very few games that get rated a lot, and most get rated almost never - the distribution is much more uneven than some of the other example datasets we've seen, there are just many more games that almost nobody seems to know or care about in this dataset.

Most users only review a few games, something like 90% of them review fewer than 5. But it's not as uneven as the reviews to games plot - in general we should be able to discern user profiles relatively easily, but some games will have relatively sparse review coverage.
All of this taken as a whole indicates there might be a significant cold start problem, and the bias in the dataset and lack of information on most games might be difficult to handle for the algorithms we have access to.

Approach

<u>Evaluation methodology</u>

For evaluation, we decided to use collaborative filtering techniques such as: KNNBaseline, SVD and SVDpp. We'll optimize parameters with a grid search and use cross-validation / held-out data in order to get a less biased view of our algorithm's performance. Once we have the best model from each algorithm, we will evaluate and compare our models, and then finalize the best model.

<u>Evaluation metrics:</u> Accuracy metrics such as RMSE, Recall, and Precision. Some algorithms are also evaluated with non accuracy metrics like Catalog Coverage and User Coverage.

<u>Algorithms</u>: The algorithms that are included are KNNBaseline for neighborhood algorithm, SVD, SVDpp, and CoClutersting for model-based algorithms. We also have an unpersonalized content-based recommender using the review text instead of scores; this isn't directly evaluated with accuracy metrics, but has Catalog Coverage.

Results

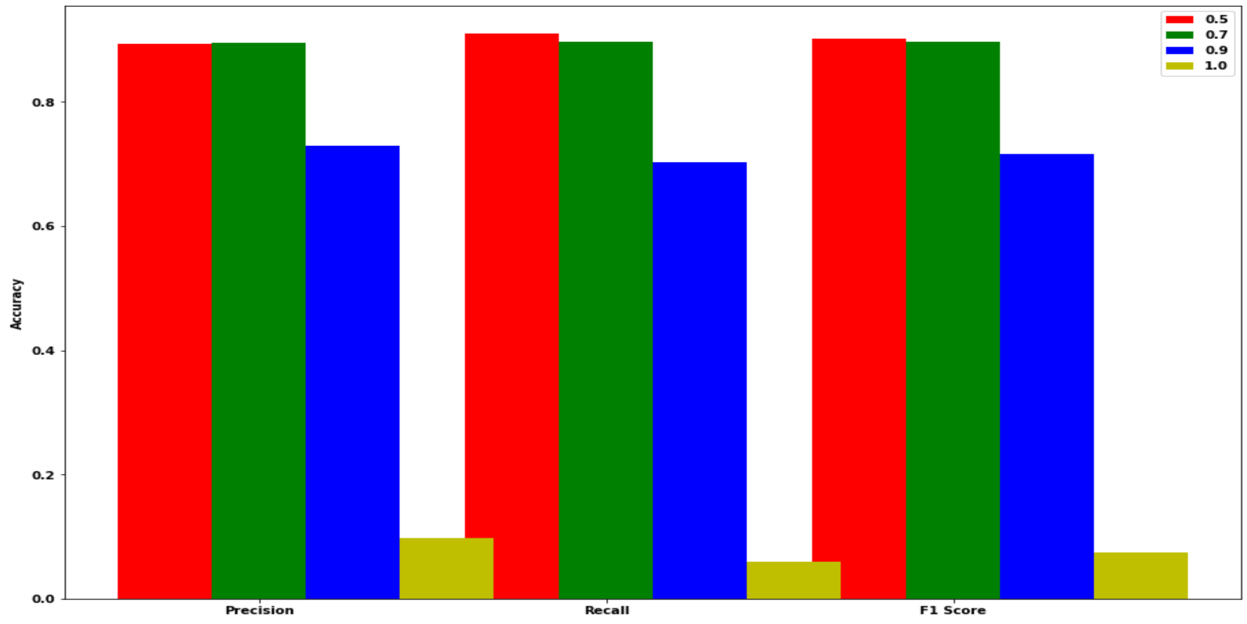
For neighborhood algorithms, after testing each one implemented in surprise and optimizing with a grid search, the best results on this kind of algorithm were a RMSE of .2986 on the held out validation data; this was achieved with a user-based version of the KNNBaseline algorithm, with settings of k = 30, using sgd with a learning rate of .01 to calculate the baseline.

When calculating precision and recall at k = 5 on the held out data with that KNNBaseline method, the results were very bad with thresholds closer to 1 (where 1 = True and 0 = False); they improved when relaxed closer to 0.5, but the results were never better than the naive approach of simply predicting that every user would like every game. The catalog coverage on the test set was roughly 80% of the potential games in that set, depending on threshold selection (much lower for the strictest one) - so a large portion of the catalog is never recommended, but most of it is. Our conclusion from this is that, at least with the data we have, it can't be said that neighborhood-based methods are a good way to recommend games to users.

For SVD, the best rmse result from 5-fold cross validation of the baseline model is 0.38. After using gridSearchCV for n_factors, lr_all, and reg_all, the best model parameters measured by rmse are n_factor = 10, lr_all = 0.07, and reg_all = 0.1 with the rmse score 0.2937. The best rmse result from 5-fold cross validation is 0.2885. The code for the SVD algorithm can be found in the SVD Jupyter notebook. (RMSE: 0.2885, 0.2924, 0.2890, 0.2970, 0.2970)

For SVDpp, First, the dataset was split 80/20 for train set and test set. Next, grid search was performed for optimization on the train data using rmse as the metric and the best rmse and parameters were

0.306833. A 5 fold cross validation was performed on the train data using the best parameters and the mean rmse was 0.306167. Next, using the best estimator from the grid search, fitted on the retrained train set and tested on the test set to get the rmse on the test set which was 0.221787 less than the rmse obtained from the grid search on training data. Ideally, the rmse of train and test set should be closer so SVDpp would perform better if the rmse of train set was lower. Next, baseline measures for the dataset were performed using ALS and SGD using default parameters. ALS had rmse of 0.3057 and SGD had rmse of 0.3163 so SVDpp performed better than SGD baseline but worse than ALS baseline. Other accuracy measures such as Precision, Recall and F1 Score were performed using the prediction from the test set for threshold at 0.5, 0.7, 0.9,1.0 at k=10 since the ratings are binary and the bar chart below shows that as the threshold decreases so does precision, recall and f1 score. As the threshold increases, the number of relevant items and relevant results returned by SVDpp decreases.



Next, User Coverage was performed on the prediction from the test set of SVDpp algorithm and top n recommendation where user coverage is basically percentage of users have at least one good recommendation based on threshold. The threshold used were 0.5, 0.7, 0.9, 1.0 with results 98.2%, 91.9%, 71.95%, 9.09% so we can see that the user coverage decreases as threshold increases like what was seen in precision and recall.

For CoClustering, we used grid-search to find the best parameters of n_cltr_i, n_cltr_u, and n_epochs. The best rmse and parameters were 0.336 {'n_cltr_i': 1, 'n_cltr_u': 1, 'n_epochs': 20}. A 5-fold cross validation was performed and the best rmse was 0.326. (RMSE: 0.3281, 0.3258, 0.3271, 0.3343, 0.3320)

For non-personalized content-based recommendation, the approach was to construct a similarity matrix from the processed text content of every review given to each game (with cosine similarity), and then recommend the k most similar games to that. It isn't possible to evaluate this accuracy-wise the same way user recommendations can be checked; however, a qualitative survey of several recommendation sets shows that this is plausibly a workable similarity measure, although any methods using it need to worry about novelty, since games in the same franchise show up as highly similar by this measure. Additionally, we implemented catalog coverage for this approach, and found that this approach has decent catalog coverage - when giving 5 recommendations for each item, about 78% of the games in the dataset are recommended at least once.

One of the challenges for this dataset was working with binary ratings. An example of the limitation would be when cluster analysis using kmeans was performed and most of the games were all clustered in one cluster. If the ratings were continuous, then cluster analysis would have performed better using the means of the ratings. Another challenge was when trying to perform content-based recommendation and the only content we had was the item name which are unique so not useful and reviews which just gave a positive comment if the game was recommended and negative review if the game was not recommended. Also, some of the reviews were not in English.

	CoClustering	KNNBaseline	SVD	SVDpp
RMSE	0.336	0.2986	0.2937	0.306167

Summary

From the algorithms KNNBaseline, SVD, SVDpp, and CoClustering, SVD had the lowest RMSE of 0.2937. However, our approaches are not particularly better than the baseline cases, so more sophisticated approaches might be needed to make especially good recommendations for this problem. We leaned how to work with dataset with binary ratings and how some algorithms like SVD perform better than other algorithms like CoClustering for this type of dataset and how some other alogorithms like KNNBaseline and SVDpp are really close behind in terms of rmse so other accuracy metrics such as Precision and Recall can give better insight into the dataset.

There's a lot more we'd like to do with this dataset if we had more time. Implementing other non-accuracy metrics would be a priority for evaluating how useful users would find these recommendations. Additionally, we'd like to look more into how the content-based method of using review text to calculate similarities works on the dataset - we have a proof of concept that it's workable, but since we didn't finish any personalized implementations of it, it's not clear how well it performs against the other methods. Finally, none of these methods improve upon the baseline by as much as we'd hope - it would be interesting to see whether an ensemble recommender could make bigger gains.

Appendix: Notebook Contents

Since we submitted several notebooks containing our work, this is a quick guide to the contents of the notebook.

SVD and CoClustering.ipynb contains code and results related to our work optimizing and evaluating SVD and CoClustering for use on this dataset.

Content, Item Game Recs.ipynb contains code and results related to our work implementing content-based recommendations using the review text in our dataset.

KNNProject-ToSubmit contains code and results related to our work on optimizing and evaluating neighborhood-based methods for use on this dataset.

SVDpp_analysis.ipynb contains code to optimizing and evaluating SVDpp algorithm and cluster analysis.

Data_Exploration_Wrangling.ipynb contains the code to data exploration, data analysis and data wrangling.