Classifying User-Item Pair Recommendation Using Steam Reviews and Metadata

Oliver Korchnoy
University of California, San Diego
okorchnoy@ucsd.edu

Clyde Baron Rapiñan
University of California, San Diego
crapinan@ucsd.edu

Sasami Scott
University of California, San Diego
sjscott@ucsd.edu

1 DATASET

The data in our project comes from the online games seller Steam. This data originally came from two distinct sets; one is a set of Steam user reviews and the other is the metadata of a sample of games published and sold on Steam. There are 24,380 unique Australian Steam users who have a combined total of 59,305 reviews present within the dataset. These reviews had several attached attributes with the most important being “recommend”, a Boolean which is True if the review recommends the game, and "item_id", the game id. We also have the metadata for 32, 132 games on Steam. We heavily used the “tags”, “genres”, “specs”, “sentiment”, “publisher”, “developer”, and “id” fields. The “tags”, “genres”, and “specs” attributes all operate on Steam as different categories to describe game-type, often for searching purposes, and are stored as a list; “tags” has the highest number of unique values at 325. The “developer” and “publisher” attributes for the game are also extremely varied with over 500 distinct values each. "sentiment" refers to the ranking Steam gives games based on how positive or negative their ratings are overall and thus is very related to our user reviews.

Table 1: Relevant Combined Data Features

Features	Unique Values
users	24,380
games	3,195
genres	19
tags	325
specs	39
sentiment	17
publisher	848
developer	1128

The feature “id” in metadata and "item_id" in the reviews both identify a unique game, and are shared between them by 3,195 entries **[Table 1]**. The dataset we used going forward used games which had a review and were included in the metadata. We merged the two datasets to one where each entry is a review with metadata from the reviewed game. This gave us a final total of 53,988 entries.

The number of reviews and items was very important to us when choosing a task. There was a ratio of around 1 user per 2 reviews, and the number of reviews per user skewed heavily to the right **[Figure 1]**. The pattern was similar with games as only a little over 60% of games had more than 1 review. Another issue is that not every entry has every feature, which led to many being cut because of the missingness. Some had obvious ways to fill in this data, but in cases like the dropped feature "discount_price", which less than 1% of the games had information for, there is no way to use it later on. Lastly, it’s important to note that there was a recommendation bias amongst the reviews; nearly 88% of all reviews were recommending the relevant game rather than critiquing it. These factors ultimately had a profound effect on our solution and task.

.

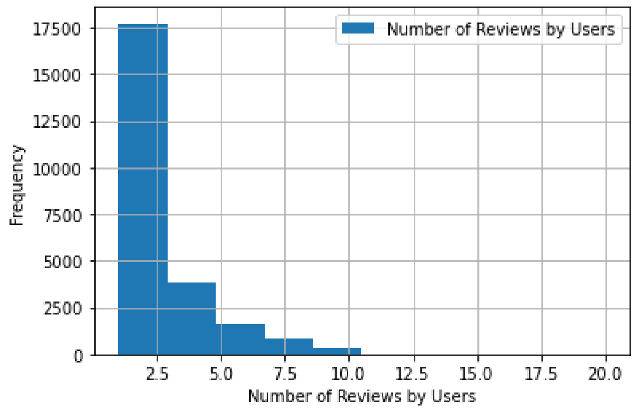

Figure 1: A histogram displaying the number of reviews written per user.

2 PREDICTIVE TASK

Our predictive task was to build a classifier that identified whether a given user would recommend a specific game; in this problem, a user not reviewing the game is the same as not recommending it in the review. The relatively low amount of data makes recommendation not only difficult to develop, but difficult to check for accuracy. So, by going the classification route we could use more of metadata of each game which is more plentiful. Many of the features from the metadata take a string form so these had to be expanded to one-hot encoded variables for use in the model. For those stored in a list per game like "tags", they were also expanded so that the one-hot encoding was for the unique values inside the lists. Any variables that were too sparse within the data were not considered further. Our most novel features were both different types of Jaccard similarity. We included two types because of the limitations surrounding the size of the dataset.

To ready this data for being fed into a model and testing, it was split into a training and test set. 5/6 of the data was used for training and 1/6 was used for testing, this way the model will still have over 50,000 entries for training. Normally these splits would be completely random, but the data’s extreme bias towards reviews that recommend games could lead the model to output more false positives. So, each section's data is random, but in such a way that there is the same number of true negative reviews for each section.

To design our baseline we decided to dive into the data once more to understand how different metadata attributes effected each games' recommendation rate. The data was manipulated such that each entry contained a game, its metadata, the amount of reviews it appears in from the user review data, and the recommendation rate from said reviews. From there we got the absolute value of the correlations with the non-Jaccard features. **[Figure2]** shows how within the twenty correlation values, sentiment values appears multiple times. This, along how related it naturally is to the reviews, made us chose it as our baseline feature to be used in a logistic regression model. Our baseline needed to be simple to show the improvement future tuning could bring, and logistical regression fit the bill because of its minimal amount of parameters. With this, the goal became to beat this baseline model's accuracy of 69.35%.

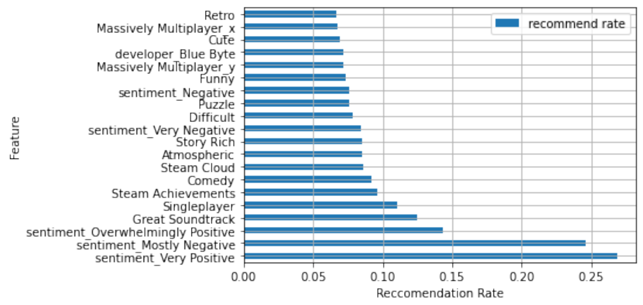

Figure 2: The correlation between various one-hot encoded features and recommendation rate per game. Duplicates signify overlap across the "tags" and "specs" features, which is expected.

3 LITERATURE

Our original datasets consisted of Version 1 of the Steam Review Dataset[4], along with version 2 of Steam Item metadata[6]. Previously Professor McAuley along with Professor Wan used the Steam Review Dataset in their paper, "Item Recommendation on Monotonic Behavior Chains"[4], a study where the group attempted to combine both 'Explicit' and 'Implicit' feedback in their own custom recommender. The professors used the user's purchase information, play time of games, text reviews, and whether the user recommended the game or not to effectively determine a 'purchase - play - review - recommend' chain for each user-item pair. The Steam item metadata was also used by Professor McAuley, but this time with Professor Kang for another paper, "Self-Attentive Sequential Recommendation"[6]. Where the professors propose a new sequential model, the SASRec to combine the simplicity of a Markov Chain Based Recommender[5] with the ability to determine 'relevant' semantics like an RNN[3]. The professor utilizes the metadata extracting a variety of features such as users' play hours, pricing information, media score, category, and the game developer. In our model, we also utilized many of these features using "tags", "genres", "specs", "sentiment", "publishers", and game developers as features in our predictor.

This was our semantics portion, with Jaccard recommendation[1] acting similarly to the Markov Chains in the professor's second paper by making predictions on "relatively few actions". The Jaccard was done with the dataset from the first paper utilizing more 'Implicit' features like judging user similarity by the games they owned in their inventory, and game similarity by the users who owned that particular game. In the professor's first paper, he concludes that using a combination of both 'explicit' and 'implicit' user behaviours and merging them together in an algorithm that models the full spectrum of a user's feedback is an efficient method of prediction. In our model, we do things differently and combine the user's feedback with semantics to produce an efficient model. We arrive at the conclusion of the professor's second paper that the best algorithm is one that utilizes both user-actions/similarities and specific precise features through metadata to better predict what a user will do.

4 MODEL

4.1 Model Choice

Our final approach to review recommendation prediction utilizes a binary Ridge Classification method [2] because of it's flexibility to penalize large weights in a linear classifier. We will denote classifier weights as $\Theta = \{\theta_1, ...\theta_{|\Theta|}\}$. Binary logistic regression takes on the form:

$$\mathbf{argmax}_{\theta} \Pi_i \delta(y_i = 1) p_\theta(y_i|X_i) + \delta(y_i = 0)(1 - p_\theta(y_i|X_i)) \quad (1)$$

where $\delta(argument)$ evaluates to 1 if the $argument$ is true and 0 otherwise and

$$p_\theta(y_i|X_i) = \sigma(X_i\theta) \equiv \frac{1}{1 + e^{-X_i\theta}} \quad (2)$$

This form of classification maximizes $p_\theta(y_i|X_i)$ and thus $X_i\theta$ when y_i is positive. This leads to potential over-weighting of some features with no checks involved. Ridge classification on the other-hand, optimizes for both error and weights.

$$\mathbf{min}||y_i - X_i\theta||_2^2 + \alpha||\theta||_2^2 \quad (3)$$

Where additionally, binary Ridge classification encodes y_i to $\{-1, 1\}$. With this form, we can pose Ridge classification as least squares problem.

(1) $\mathbf{min}||y_i - X_i\theta||_2^2 + \alpha||\theta||_2^2$
(2) $\mathbf{min}||y_i - X_i\theta||_2^2 + ||\alpha\theta||_2^2$
(3) $\mathbf{min}||y_i - X_i\theta + \alpha\theta||_2^2$
(4) $\mathbf{min}||y_i - \begin{bmatrix} X_i \\ -\alpha \end{bmatrix} \theta||_2^2$
(5) $\mathbf{min}||y_i - X_i'\theta||_2^2$

Thus it is very plausible to utilize this approach when fitting our model.

4.2 Feature Selection

With this model, we have yet to optimize both the construction of X_i from the data and the α hyper-parameter. We initially defined X_i as

$$X_i = [1, (tag_{oh}), (gen_{oh}), (spe_{oh}), (sen_{oh}), (pub_{oh}), (dev_{oh})] \quad (4)$$

Where key_{oh} denotes a one-hot encoding of "tags", "genre", game "specs", game "sentiment", "publisher", and "developer", respectively.

4.2.1 *Feature evaluation.*

As described in section 2, we can evaluate our classifier overall by the simple metric of accuracy: the number of predictions of the total predictions. We can also utilize this metric to perform an **ablation** experiment on the features to find the relative importance of each feature by comparing the change accuracy of the whole model with different combinations of features in X_i. Our evaluation of features is shown in **[Table 2]**.

Table 2: Ablation of Features X_i

features	accuracy(%)
all features	69.16
excluding tags	69.07
excluding genres	69.17
excluding specs	69.15
excluding sentiment	68.93
excluding publisher	69.05
excluding developer	69.082

To our surprise, **genres** and **specs** had little to no effect while **sentiment** remained unsurprisingly the most important due to it's high correlation. While these features create a decent model, based on our predictive task, we are still not relating users to each individual items. Therefore, we decided to include a Jaccard score as a feature.

4.3 Addition of Jaccard Features

Jaccard set similarity can be used to evaluate the similarity of two items[1]. In our case, we wish to find the similarity between a particular user and a game. Initially, we tried implementing a standard Jaccard similarity workflow comparing the each user that recommended that game with the current user. However, we found this computationally taxing, as to predict a single user, game pair, you would need to compare every user that reviewed that game. We believed some optimization could be made to ensure that we maintain a short prediction time for our model.

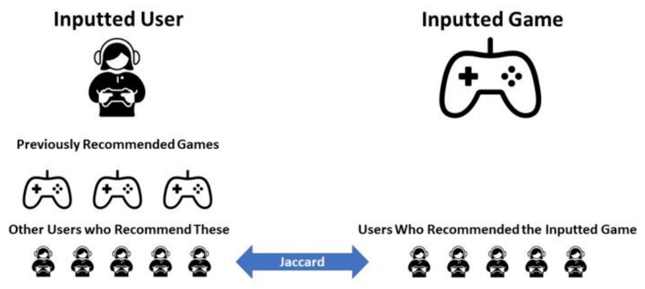
Inputted User
Inputted Game
Previously Recommended Games
Other Users who Recommend These
Jaccard
Users Who Recommended the Inputted Game

Figure 3: Jaccard Formulation

Our formulation of Jaccard **[Figure 3]** both maintains comparing similarities between games and users without the excess computation while taking into consideration the user's past behavior. By comparing the set of users that also recommended the same games as the user in question and the users that recommended the game in question, we can successfully compare sets of users and get a similarity score with a single Jaccard computation. Adding this score to our model will enable the model to find a threshold of similarity to classify user-game recommendability.

With our initial optimal set of features ("tags", "sentiment", "published", "developer"), we were able to increase our accuracy to 69.97%. Upon further investigation, we found that the Jaccard feature alone accounted for most of the accuracy (67.36%), which is reasonable since this score directly compares user-game pairs. However, we also acknowledged that adding the other features also helped accuracy since game popularity overall can have a positive correlation with recommendability.

With this in mind, we decided to add an additional Jaccard score as a feature with a complementary formulation **[Figure 4]**. Instead of comparing sets of users like in the previous Jaccard feature, we would be comparing the set of all the games the user in question recommended with the set of games other users also recommended that recommended the game in question.

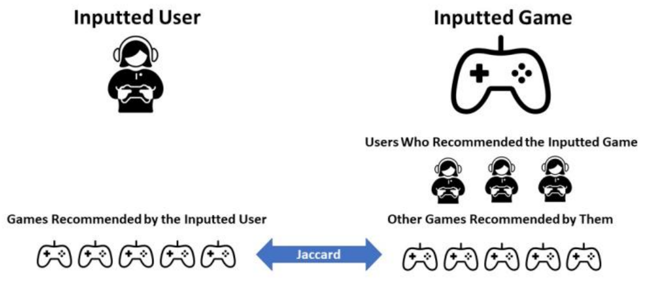
Inputted User
Inputted Game
Users Who Recommended the Inputted Game
Games Recommended by the Inputted User
Other Games Recommended by Them
Jaccard

Figure 4: Complementary Jaccard Formulation

With the addition of this feature, our accuracy rose to 70.16%, a more significant jump than all of the other features mentioned in 4.2. Again, upon inspecting the Jaccard features' significance, we were able to find that they only scored 67.87%.

Adding the Jaccard features enabled us to gain accuracy because of their direct user-game compatibility. However, it should stand that Jaccard alone does not reflect nearly as much accuracy as out metadata features mentioned in 4.2. This may be due to the positive recommendation skew mentioned in section 2 and we hypothesize that with a more balanced dataset, Jaccard would ultimately out perform the metadata features.

4.4 Hyper-Parameter Optimization

Now we have established a successful feature set that relates user-game pairs, we can optimize for our penalizing parameter in the Ridge Classifier model, α. We can do this by iteratively training and testing our model and performing the search again with a smaller increment size and smaller range around the maximum performing value. This way, we can effectively approach the maximally efficient value to arbitrary precision without searching through every value in a large range.

Our accuracies are recorded with this process in **[Table 3]**. The first couple iterations of finding your optimized parameter revealed that a maximum accuracy is achieved around $\alpha = 5$. However, through a longer iterative process, we were able to narrow down several maximal hyper parameter values. Initially, we deduced that

Table 3: Ablation of Features X_i

alpha value	accuracy(%)
3	70.23
4	70.28
5	70.30
6	70.29
7	70.29
...	
5.299999999994	70.31562569
5.399999999994	70.31562569
...	
5.599999999994	70.3045121138
5.699999999994	70.3045121138
5.799999999994	70.31562569

this could be a product of over-fitting the data, however, because our workflow includes testing on a disjoint, randomized set, we concluded that this was due to either one of our methods having a non-linear component or a floating point error and thus arbitrarily chose $\alpha = 5.799999999994$.

5 RESULTS / CONCLUSIONS

In conclusion while our model did beat our baseline, it did not perform as well as we would have liked. Maxing out at around 70.4% by utilizing the linear least squares solver with the aforementioned features. Meanwhile our baseline performed with a 69.35% accuracy. Our best performance only had a very minimal increase of 1.35%. Ultimately we associate this with a variety of issues with our model. It appears that our simple baseline of a logistical regression on "sentiment" and it's corresponding high accuracy seems to imply that Steam users tend to rate games with already pre-existing positive sentiment more positively, and games with pre-existing negative sentiment, more negatively, implying a bias of the masses where users tend to review in a similar fashion to others who have beforehand.

Our current model associates a compatibility between user-game pairs via our Jaccard features with additional assistance from the selected game's metadata. This has the effect of primarily determining recommendability between a user and a game with additional considerations to the general features of the game overall. The Jaccard feature scores, while weighted over the whole training set, maintain personalization for each user-game pair. This has the effect of creating a similarity threshold that a user-game pair must score above to be deemed recommendable. This contrasts with the "tags", "genres", "publisher", "developer", "sentiment", and "specs", being encoded directly into the model's features, having the result of weighing those features over the whole training set of the data.

There are several areas in which we could improve our final model. For one, when utilizing metadata features such as "genres" and "specs", our algorithm provides no personalization for the individual user. Categories like "Action" or "Adventure" are not popular enough among all users to be useful without knowing how the user's preferences. Thus, this data can damage our model. Our Jaccard feature, as mentioned previously in 4.3, when introduced raised our accuracy from 69.97% to 70.16%. Interestingly, however, the Jaccard feature by itself was in fact the weakest when utilized. We believe this is due to an inherent weakness in our Jaccard where all games regardless of user play-time were weighted equally. Outside of additional optimizations to our proposed method, we could enhance our predictions with text analysis. Since we extract user-game pairs from the Steam Reviews Dataset, we could utilize n-gram one-hot encodings to determine recommendability per each review. This addition would add even more personalization to the user, as well as potentially detect a positive or negative opinions of the game directly from the text content of the review.

To summarize, while this solution can be categorized a success, there is much more to be done in this space. Finding ways to incorporate more review specific features like the text and personalizing results based on user history will improve results and lead to more applicable models.

REFERENCES

[1] Sam Fletcher and Md Islam. 2018. Comparing sets of patterns with the Jaccard index. *Australasian Journal of Information Systems* 22 (03 2018). https://doi.org/10.3127/ajis.v22i0.1538

[2] Martin Grüning and Siegfried Kropf. 2006. A Ridge Classification Method for High-dimensional Observations. In *From Data and Information Analysis to Knowledge Engineering*, Myra Spiliopoulou, Rudolf Kruse, Christian Borgelt, Andreas Nürnberger, and Wolfgang Gaul (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 684–691.

[3] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based Recommendations with Recurrent Neural Networks. (11 2015).

[4] Julian McAuley Mengting Wan. 2018. *Item Recommendation on Monotonic Behavior Chains.* RecSys.

[5] J. McAuley R. He, W. Kang. 2017. Translation-based recommendation. *RecSys* (2017).

[6] Julian McAuley Wang-Cheng Kang. 2018. *Self-Attentive Sequential Recommendation.* ICDM.