Metadata Analysis for User and Video Game Purchases

Michael Kentaro Cho, mkc012@ucsd.edu, (A10645866)

Brian Orensztein, borenszt@ucsd.edu, (A13017297)

Abstract:

The video game industry has been continuously growing as technology integrated into our everyday lives. Video game platforms such as Steam need to promote their games and be able to accurately recommend relevant games to their users. This can be done through careful analysis on User-Item data pairs along with various other fields on interest. We aim to explore both basic and complex methods to determine if a user will purchase a game or not. Correctly predicting this could lead to various benefits such as creating a recommender system that could predict the user rating or preference in relation to a game.

By using naive approaches, we can achieve relatively simple baseline models that perform quite well. However, we can see significant improvements with our complex models by extracting various features and applying different approaches to the data.

Keywords: Baseline, Support Vector Machine, Jaccard, Text Mining

Dataset Analysis:

In this report, we will be working with both the Steam User Reviews dataset and the Steam User Items Dataset acquired from the Professor's repository of Recommender System datasets. We will be using both of these datasets to collect enough information from both users and items as well as user reviews of their purchased items in order to make interesting categorical predictions that use prediction modeling techniques that we've learned in class. First we will examine each dataset and what fields of each entry might be useful and why.

'User Reviews' Dataset Overview:

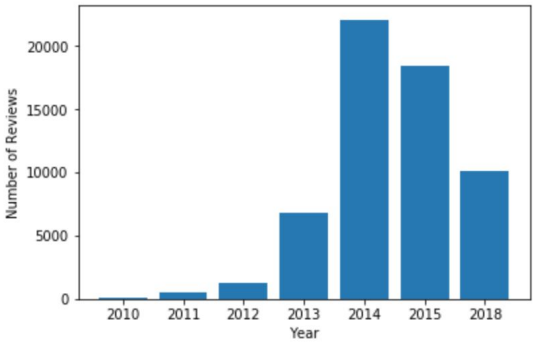
Within this dataset, the main fields of interest per entry are:

- *user_id (string)*
- *Reviews (object)*
  - *posted (string / date)*
  - *item_id (string)*
  - *helpful (string / ratio)*
  - *recommend (bool)*
  - *Review (string)*

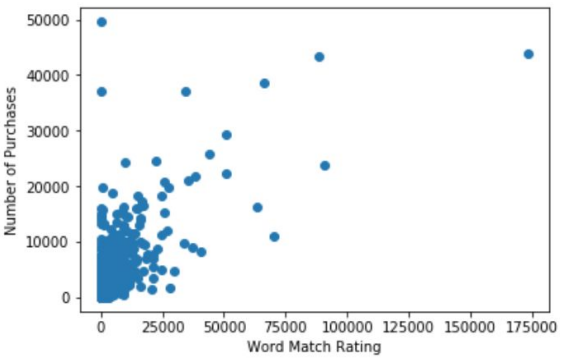
We can omit other fields like user_URL which isn't likely to play a role in our prediction model. As we did with Assignment 1, we can assume that users have purchased any game that they have written a review for. Thus we have the capability to produce user-item pairs that represent users that have purchased specific items. As such, we also have the capability to produce non-purchased pairs through a combination of random assignment, and checks to make sure these generated pairs do not represent an existing purchase. Looking at the fields of the data, we can also glean information about the user-item pair like whether or not the user recommended the item, how helpful this review is, and we can even perform some amount of text mining on the review text itself.

General Statistics and Plots for User Reviews:

- There are **59,305 reviews** in this dataset.
- There are **87,626 unique users**.
- Below is a bar chart showing the number of reviews written per year. Here we can see that our data ranges from the years 2010-2018. The year with the most number of reviews was 2014, showing a dramatic peak that tapers off into 2018.



- Below, we see a scatter plot showing the relationship between the number of words a games review shares with the top most popular game's reviews, and the number of purchases that the game has. Observe that games whose reviews share many words with popular game reviews, generally have higher rates of purchase. However, we do see some outliers.



‘User Items’ Dataset Overview:

Within this dataset, the main fields of interest per entry are:

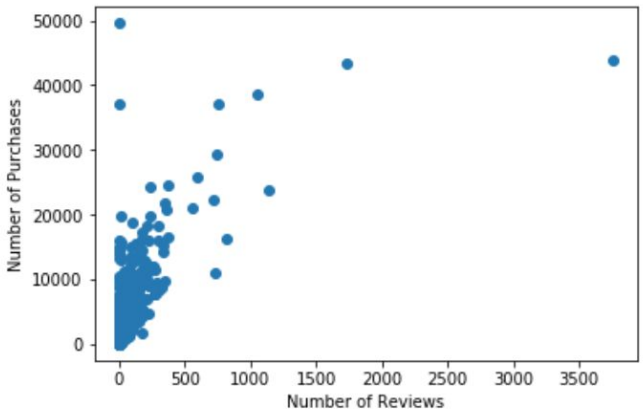
- user_id (string)
- items_count (int)
- items (object)
  - item_id (string)
  - item_name (string)
  - playtime_forever (int)

We can omit other fields like user_URL and playtime_2weeks which aren’t likely to play a role in our prediction model. Where the User Reviews dataset is useful for finding unique features that we can use to construct more complex predictors, the User Items Dataset is useful for constructing more simple predictors that we can use as baselines. By simple, we mean models that might simply count and develop ideas of basic probability. For example, we will set a baseline predictor that uses the fact that the average number of items a user owns is ~59 items, and then predict that the user will buy the item if they already own higher than the average (the user is a “Heavy Buyer”).

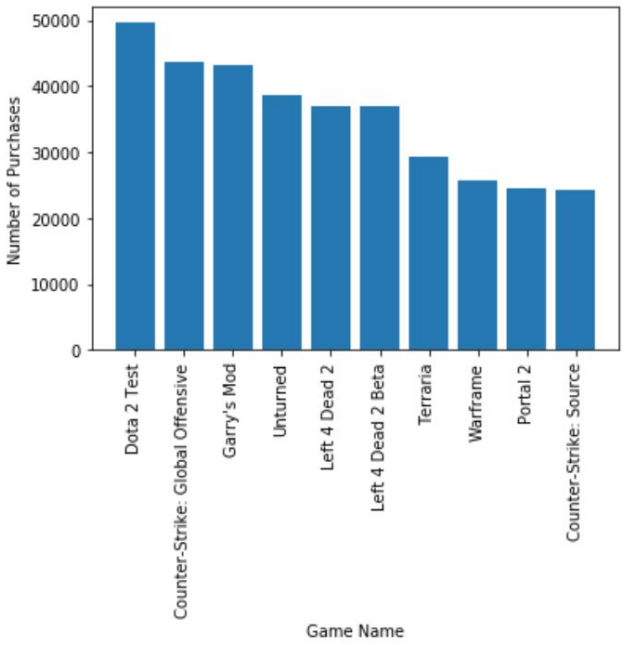
This set is also helpful for validation and testing since it can tell us whether or not user-item pairs exist or not and therefore whether or not our model predicted correctly or not.

General Statistics and Plots for User Items:

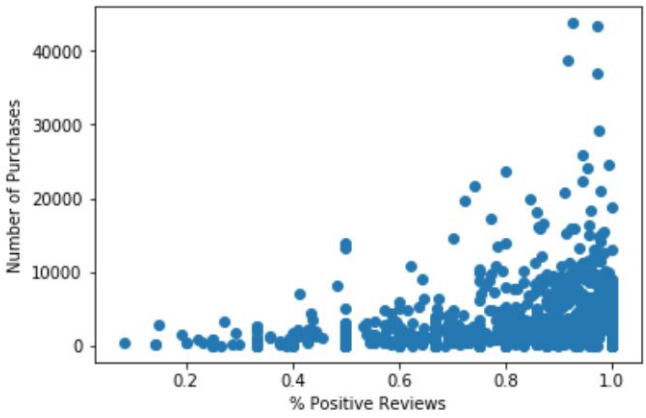
- There are 88,310 entries in this dataset
- The total number of unique items is 10,978.
- The average number of items owned by a user is 58.809 items.
- The standard deviation in the number of items owned by a user is 123.9
- The standard deviation in the popularity of games based on purchases is 1813.3



- Above is a scatter plot showing the relationship between how many reviews a game has, and how many times it has been purchased. We can see that there is very loose correlation between number of reviews and purchases but that it also true that some games that have very few reviews have some of the highest purchase rates.



- Above we see the top 10 most popular games by number of purchases. Note the huge difference between *Dota 2 Test* and *Counter Strike: Source* despite both being in the top 10 most owned games.



- Finally, above we can see that the number of purchases that a game gets and the ratio of the number of Recommendations it gets to Reviews (essentially its ratio of positive reviews) are correlated, suggesting this might be a good feature to use.

Training, Validation and Testing sets

Using data from both of the above mentioned datasets, we can process and construct a working overall set that we can use for our predictive task and its models. Our predictive task will be related to users and whether or not they

will purchase a particular game from the Steam platform. So we are mainly concerned with user-item pairs and whether our predictor will predict Bought or Not Bought. Thus we can represent our sets as tuples of User-Item strings pairings and predictions.

It is important to look at any problems we might have with the datasets before we can represent our features in simple and useful ways. The first thing we might notice is that in User Items Dataset, any user can have bought more than 1 game and many do. So we can expand this set and construct a larger set of user item pairs that are all unique and that all exist in the User Items Dataset. This results in a large set of Purchased User-Item Pairs of size 5,153,210. Our training, validation, and testing sets will derive from this large set.

However, the next thing we need to notice is that our sets consist of only user-item pairs that have been bought. So before we can start testing, we need to add to our validation set so that it includes non-bought user-item pairs to test for. These paris will be generated by randomly pulling from the total dataset and making sure that these pairs do not exist in the User Item set. After appending this non-bought set of pairs to the bought set of pairs, the validation set will be of length 3,435,472.

Predictive Task:

Above all else, Steam is a platform for selling games. As such, the most interesting prediction to make is whether or not a game will be bought or not. Specifically, we will be predicting whether or not a particular game will be bought by a particular user.

In examining the data sets detailed above, we can see we have some useful data to work with here. In particular, we can uniquely identify users and games, and we also have access to several variables that can play a part in determining the relationship between a game and a user, and therefore whether that user will buy the game or not. For example, we can see how many games a user owns, how many times that game was bought, how many reviews are written for it, how helpful any review for it is, and even look at the language patterns of reviews for games that the user owns.

The models that we can explore for this predictive task will be evaluated based on their accuracy in correctly predicting if a game has been bought by a user or not. The predictors will use features and properties gathered from the training set, they will then be checked and tuned through validation, and then tested for accuracy through the test set.

To be more precise, **a correct prediction is one where a user-item pair is presented to the predictor as input**. This pair represents a user and item for which the predictor must determine whether the item will be bought by the user. The predictor will output a 1 for “yes” and a 0 for “no.” The predictors **performance on the test set is our evaluation of its accuracy**.

Baseline

In order to keep track of how well our models perform, we need to establish a baseline prediction model for comparison.

1. The most basic baseline prediction model that we will be using won’t even make use of the User Reviews Dataset. This model will use the training set to determine the **average number of games that users own**, and then for each user-item pair, predict that the user will buy the item if they already own more than the average number of games that users own. Essentially this checks if the user is a “**Heavy Buyer**” and if they are, to predict that they will buy the item in question.
2. The next iteration of this model is to set a more informed threshold than just the average number of games that users own. Instead, while training, we can adjust the threshold of what we consider to be a “Heavy Buyer” until it provides the best accuracy after checking with the validation set.

This proposed baseline model, which is constructed through one iteration on a simple probabilistic style approach to prediction modeling, will serve as a point of reference as we start using more sophisticated techniques in more advanced predictors. For this baseline, performance will be evaluated with respect to how many correct predictions were made on the testing set.

Data Processing

Based on the baseline model proposed above and the other baselines that we will use, we will need to pre-process the dataset in order to construct the features that we need. The features we will use are:

- ❏ (Boolean) Is the user a “Heavy Buyer?” (Do they own more than the threshold?)
- ❏ (Boolean) Is the item a popular item? (Is the item more popular than the threshold?)

Since this is our baseline model, we expect this corresponding set of features to also be quite simple, as shown above. We can process the given data, as structured above in our Dataset Analysis, to generate these features for every user-item pair that we will need to predict for.

For the **“Heavy Buyer” feature**, the number of *games owned by each user* is already listed as a field associated to each user_id in the User Items Dataset dictionary. However in order to calculate the *average number of games* that users own, we will need to keep a total of all games owned by all users in our training set, and divide that number by the number of unique users in the set. The feature here will simply be whether or not the number of games owned by the user in the user-item pair is greater than or equal to the average number of games owned by users.

For the "**Popular Items**" **feature**, we will take a similar approach as we did in Assignment 1. We will count how many times each item in the user-item pair appears in the entire User Items Dataset, and store that number in a dictionary as a value associated with each unique item. If we then sort in descending order by that value, we can look at the top x items in the dictionary and know the most popular items in the dataset. Our feature will simply be whether or not the item in the user-item pair appears in the top x most popular items list or not.

Proposed Model:

In order to decide which model we believe to be the best performing model, we will need to try and evaluate several modeling techniques and experiment with what features work best and where we can combine models into better ones. We will start with small improvements to the baseline model and gradually introduce more sophisticated techniques as we move forward. Let's start by examining the performances of our **baseline model**.

"Heavy User" Baseline Model (Average Threshold)

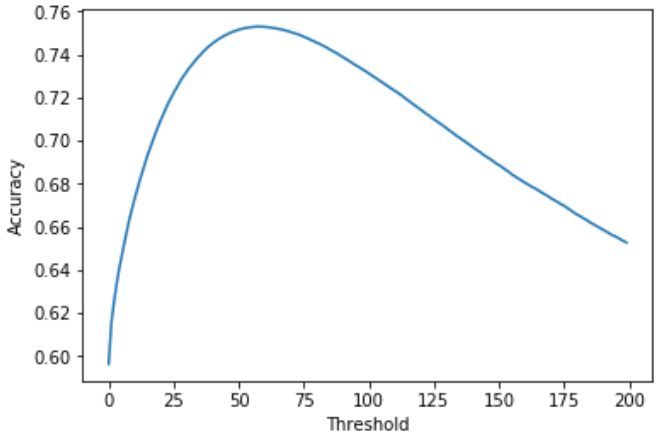
For this baseline model, we define the "Heavy Users" to be those who own more games than average, or greater than 77 games as calculated from the Training Set data.

- ★ **Accuracy on Testing Set: 74.6%**

"Heavy User" Baseline (Adjusted Threshold)

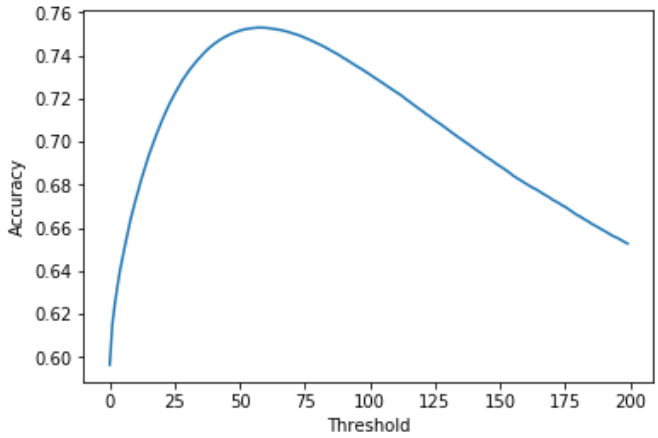
We will try improve on the baseline model by using our validation set to find a better threshold for "Heavy Users" than just the average number of games from the Training Set.

Below is a curve that shows how the accuracy on the **validation set** varies based on the threshold, from predicting 1 if a user owns more than 0 games and 0 otherwise to predicting 1 if a user owns more than 200 games and 0 otherwise.



We found the optimal threshold to be 58, which is to be expected since the average number of games owned by users in the much larger Validation Set is 58.

Below is a curve that shows how the accuracy on the **test set** varies based on the threshold, similar to the previous one.



We can see that the curves are almost identical. Since our validation and test set are very large, all the users are captured in both, making the average number of games owned, 58, the same for both. Thus It is important to note that this exposes an inflexibility in our baseline model in that the average will change as more users are added into the dataset, but as the dataset gets extremely large, the average will have less variation. While we could calculate this average on-the-go or adapt to this scaling issue, we've decided instead to move forward with other techniques.

Note that we only show the various accuracies for the test set to point out a weakness in the baseline due to extremely large datasets. Ideally, we would only predict on the test set once using the optimal threshold of 58 received from the validation set.

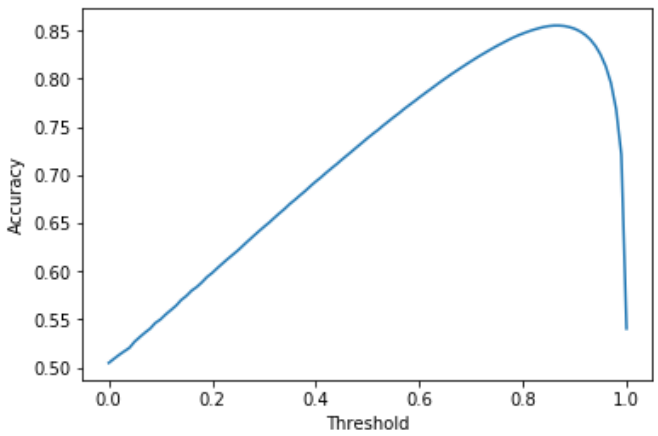
- ★ **Accuracy on Testing Set (Optimal Threshold): 75.3%**

After observing our baseline model performances, we can start to incorporate more techniques.

Popular Game Model

The next model we want to test uses popularity of the item in question. In other words, we can construct a list of the top x most popular items by counting how many times this item appears in the User Items **training set**, regardless of user. We can then predict that the user in question will buy the item **if the item belongs to the top x most popular items**.

Below is a curve that represents the accuracy of our Popular Game Model on the **validation set** as threshold changes. We get the optimal threshold to be 0.86, meaning we consider the top 86% most purchased games to be 'popular'.



- ★ **Accuracy on Test Set w/ Threshold top 50% most popular games:**
  - **73.8%**
- ★ **Accuracy on Test Set w/ Adjusted Threshold top 86% most popular games:**
  - **85.5%**

We can see that if we take the top 50% most popular games without tuning the parameter with the Validation Set, this model actually performs almost as well as the "Heavy User" models. However, if we change the threshold of what we consider to be popular to the top 86% games, then we gain substantial improvements to our accuracy on the test set. If we recall that the **standard deviation of game popularity is much higher than the standard deviation of the number of games owned by each user**, we might have expected this outcome. Based on the same principles derived from Principal Component Analysis, we can see that using game popularity as a feature, rather than games owned by users, **retains more variance** in our data and thus can provide more meaningful prediction features.

Jaccard Similarity

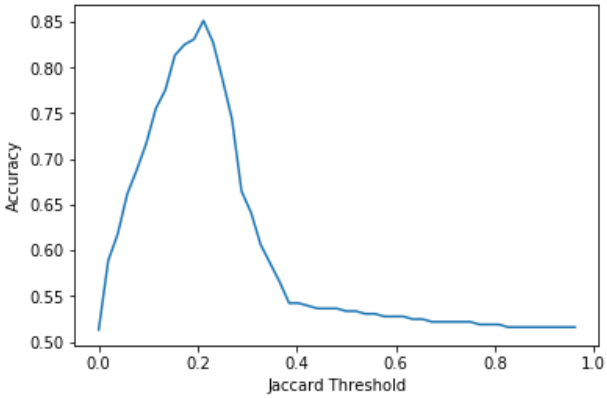
Since we have a very large data set, the Jaccard Similarity algorithm can be a favorable model. The algorithm is relatively simple, but is sensitive to small sample sizes and may give erroneous results. In our case, we have enough sample sizes to create a fairly optimal predictor.

The Jaccard Similarity algorithm predicts that a user will purchase the item if that user is *similar* to **any** of the other users that have purchased the item. The way we describe similarity is determined by the **Jaccard Index/Similarity Coefficient.** This is calculated by taking the size of the intersection of two sets and dividing by the size of the union of the two sets. The feature our two sets are defined by are the items user one purchased and the items user two purchased. We can then calculate their similarity using the method described above. Two sets that are exactly the same would be 100% similar and two sets that share no elements would be 0% similar. We then need to decide on a threshold that determines if two users are *similar enough* based on if their Jaccard Index exceeds that threshold.

- ★ **Accuracy on Training Set (threshold < 0.08):**
  - **99.4%**

Clearly, we can see the problem of overfitting on our training data due to the fact that we are calculating similarity based on users in the training set. Furthermore, by using a low threshold, we can predict the user purchased the item almost every time which will be close to 100% accurate since there are no negative samples in the training set.

We can then tune our threshold hyperparameter using our Validation Set which would now include users who did not purchase an item. Thus picking the right threshold will be crucial. Below is a curve representing the accuracy on the Validation Set as the threshold goes from 0 to 1. The optimal threshold ends up being 0.22, meaning the users must be roughly more than 22% similar to produce a positive prediction.



- ★ **Accuracy on Validation Set with optimal threshold of 0.22:**
  - **85.1%**
- ★ **Accuracy on Test Set with tuned threshold of 0.22:**
  - **84.5%**

Note that for new users, meaning they have purchased 0 items, we predict the user will purchase the item based on whether the item is popular or not using the previously discussed optimal threshold of 0.86.

Based on the performance of this model, we can conclude that Jaccard Similarity Index is suitable to be employed in the user-item similarity measurement. With the right threshold, the algorithm has relatively high efficiency while dealing with varying set sizes. However, we must note that for users with very small set sizes, the algorithm shows some weakness because some users are not heavy buyers, but end up deciding to purchase a popular game. This may cause an incorrect prediction due to a low index. Thus, other algorithms should be considered and tested such as Cosine Coefficients and Pearson Correlation Coefficients.

Text Mining

Another technique that we haven't explored yet is text mining. That is, parsing any chunks of text in our data points that might serve as features for our predictor. We will first try a primitive text mining model and then try to incorporate a text mining feature into our Support Vector Machine's feature vector.

Our simple text mining model is to take the top 100 most popular words of the top 10 most popular games from the Training Set (these numbers were chosen arbitrarily as finding the optimums here would take a long time) and to predict that a game will be purchased by any user if that game's reviews contains a higher than average overlap with words found in the most popular games' reviews. With the average word overlap per game in the training data at 1472, we see a performance that does better than the baseline but is not optimal:

- ★ **Accuracy on Validation Set:**
  - **76.634%**
- ★ **Accuracy on Test Set:**
  - **76.623%**

Note that for games with no reviews, we will predict '0'

Support Vector Machine (SVM)

A SVM is a promising model for classifying on our dataset. However, it is important to notice that since our analysis of the data set did not yield and extremely strong patterns of correlation between explicit features and labels, our choice of feature vectors will matter greatly as we train our SVM. We will start with simple feature vectors and add to its complexity gradually, measuring the performance of each model accordingly.

For every datapoint in our training set, we will construct a feature vector to represent that datapoint (User-Item pair). Due to the massive size of our validation set, we will be subsampling sets of size 15,000 in order to shorten the time spent training for our SVMs. Each subsample will be randomly generated from both purchased and non-purchased User-Item pairs present in the Validation Set. Each SVM should be tuned through validation to the optimal Penalty Parameter between .001 and 1000.

Our first feature vector will essentially be a combination of our baseline model and popular game model:

- ❑ Feature Vector (list)
  - ❑ 1 (int)
  - ❑ PopularGame (bool)
  - ❑ HeavyBuyer (bool)

With this feature vector to train upon, and with the validation set's y-vector to compare against, we get the following performance readings:

- ★ **Accuracy on Validation Set:**
  - **75.9.%**
- ★ **Accuracy on Test Set:**
  - **75.1%**

The popularity of a game seems to dominate the SVM model's predictive tendencies. In fact, based on our tests, if we omit the HeavyBuyer feature from our feature vector, the SVM retains a similar accuracy as above. However, if we remove the Popular Game feature,, the SVM's accuracy drops to baseline accuracy, indicating the SVM's strong dependency on the Popular Game feature in determining if a game is owned by a user. This makes sense, as gamers tend to own games that are popular especially on the Steam platform since many games are played online with other players.

Support Vector Machine with different Feature Vectors

Next, we try to incorporate the word overlap feature into our SVM's feature vector. When our feature vector is:

- ❑ Feature Vector (list)
  - ❑ 1
  - ❑ WordOverlap

We see a performance of:

- ★ **Accuracy on Validation Set:**
  - **76.2 %**
- ★ **Accuracy on Testing Set:**
  - **77.1 %**

This is a decent performance but still not as strong as our simple game popularity model. This further reinforces our hypothesis that out of the features we have used, game popularity is the strongest teller of whether a game will be purchased or not.

If we add another feature that checks how many of a game's reviews are accompanied by a "recommends" flag, and use the ratio of "recommend"/totalReviews as a feature, we get:

- ❑ Feature Vector (list)
  - ❑ 1
  - ❑ %Positive Reviews
  - ❑ WordOverlap

With:

- ★ **Accuracy on Validation Set:**
  - **76.8 %**
- ★ **Accuracy on Testing Set:**
  - **76.5 %**

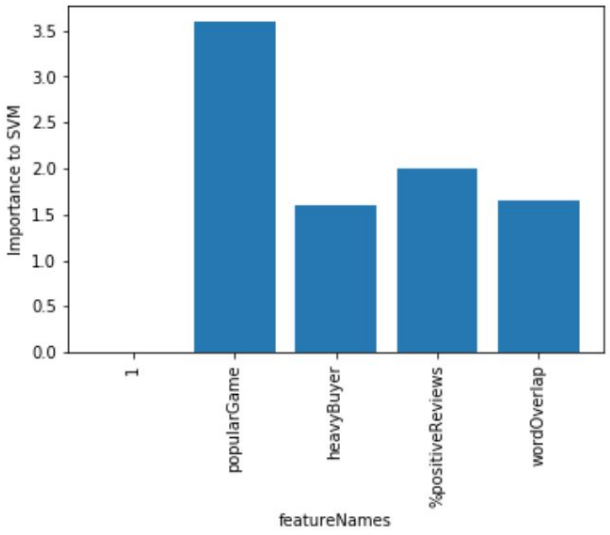
We don't observe too much variance here with a solid performance of around 76% when using any combination of WordOverlap and %Positive Reviews as our feature vector. However, once we add back in our original features of PopularGame and HeavyBuyer, we have:

- ❏ Feature Vector (list)
  - ❏ 1
  - ❏ PopularGame
  - ❏ HeavyBuyer
  - ❏ %Positive Reviews
  - ❏ WordOverlap

With:

- ★ **Accuracy on Validation Set:**
  - **81.2 %**
- ★ **Accuracy on Testing Set:**
  - **81.3 %**

This is the **highest performance we see with an SVM model** using our chosen features, training set size, and arbitrarily chosen parameters (for the sake of time). The takeaway here should be that, among our simple features chosen for this task, it seems to be clear that the **SVM's dependency on the popularGame** feature is the strongest. That being said, including other features does lead to varying levels of improvement albeit with diminishing returns. This is a good example of how sometimes SVM's with smaller feature vectors can perform quite well depending on the choice of features to include.



Above, we've constructed a bar graph using python's *SVM.coef_* function to determine which features are most useful to the SVM. In line with our findings in our Data Analysis, %PositiveReviews is actually a fairly strongly correlated feature to determine if a game will be purchased or not. And also in line with our findings, is that whether a game is owned by more people than average or not, is the most powerful factor in determining if a particular user owns the game or not. This is the simplest feature we have, but also the most influencing.

Optimizations on this model would be to use the Validation Set, as we did with previous models, to tune our parameters like our Penalty Parameter in the SVM, Validation Subsample Size, etc. However, due to the time intensive nature of running SVM's we could not run all of our optimization options.

Literature:

Steam is a massively popular platform for the distribution, selling, and playing of video games digitally. As such, it has been a large collector of data with respect to video games and the communities surrounding them. We've used the dataset of Steam User Reviews as well as the dataset of Steam User Items and constructed fairly simple models to help predict whether or not a user will buy a specific game on the Steam Platform. How else is Steam leveraged for predictive tasks?

Research performed under a Professor Doug Downey of Northwestern University outlines a Steam Game Recommender System. While the objective of the model discussed in this paper differs slightly from our own, the researchers used a similar dataset to our ours, consisting of 10,000 Steam user libraries. They also used some of the same techniques as we did in their own models. For example, they tried to use SVMs and Linear Regression which are two techniques that we've learned in class. Key differences include the fact that they split their data set 80/20 for training and testing, and evaluated their models via f-measure rather than raw accuracy, as we did. In the end they found Decision Trees to be most accurate for determining if users own a game and found all models to be approximately the same in accuracy for determining if users do not own a game. Interestingly, they noted that choosing games to evaluate their classifiers on was a challenge simply due to immense variance in the number of users for any given game. To combat this, they took into account the age of a game and correlated that to the number of users for that game. They then limited the training of their classifiers to older games with higher numbers of users. This is an interesting technique to workaround the high amount of variability that we also see in our Steam dataset.

In a Master's Thesis written by Michael Trneny of Masaryk University, Trneny provides an analysis of the factors that influence a games sales. Three factors are identified: Clicks in Search Engine results, Reviews, and Video Services. Trneny discusses how it used to be believed that game reviews from magazines were the best predictor for a game's success, however, with access to more sources of online data, users turn more towards the internet and the content of other users. Search clicks on Google for example show a 0.92 correlation between "clicks during the 10 month game launch cycle and game units sold during the first 4 months post-release". However, the sample for this study was small and the researchers acknowledge that too many other factors exist. Surprisingly in line with our own findings show that a high number of positive reviews doesn't indicate a games success as much as one might have thought, but rather it correlates better with a games minimum sales. As we saw in our scatter plot of Game Purchases vs. % Positive Reviews, we can see

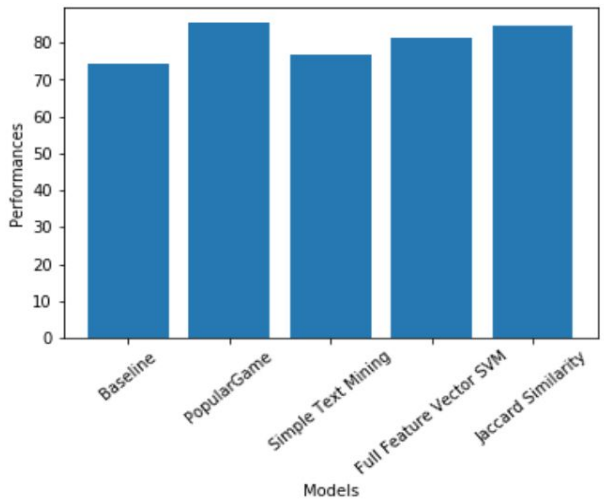
that generally games with good reviews sell better, but that there are many outliers and we can reason that many other factors like the number of game users, reviewers, the age of the game, etc. are left out, limiting our predictive power beyond a certain point.

Finally, in a paper written by Mengting Wan and Julian McAuley of UCSD, titled Item Recommendation on Monotonic Behavior Chains, we can see what types of state-of-the-art technologies can be taken advantage of to improve the predictive power of recommendation systems across many fields. Many of the problems that we faced in trying to develop a good model for our Steam dataset stemmed from the fact that we have trouble using more features than just the fundamental features like number of purchases, relationship to averages, and basic presence of words in text. However, this paper proposes a model that uses the full spectrum of a users feedback and observes that many of these instances of feedback display monotonic dependencies on each other; that the presence of one kind of feedback implies the presence of another weaker feedback. The use of these patterns allows for a model that takes User-Item interaction, and essentially constructs explicit responses to Items out of implicit data based on a behavior chain that represents a user's decision making process. This is cutting edge prediction modeling that is clearly much more sophisticated than our modeling attempts and more intelligently takes advantage of data to construct more useful features than are immediately noticeable and present.

Results and Conclusion:

After analyzing our dataset and identifying features that might be useful for our predictive task, we implemented several different prediction techniques and evaluated their accuracy with respect to how many correct predictions they made on our testing set.

Based on our findings, the two most useful features for our predictive task are the Jaccard Similarity between users, and the popularity of a game. Jaccard Similarity predicts that a user will buy a particular game if that game has been bought by other users that are similar to them. The popularity of a game is determined by whether or not that game is owned by more users than on average.



Above we can see the different performances of each model we tested on our Testing Set. With a relatively high baseline performance, we were able to construct models with as much as an **11% improvement on the baseline mode**l.

With these results, we can conclude that among our models, the most effective were our **Popular Game Model** and our **Jaccard Similarity model** with **85.5% and 84.5% accuracy** respectively. However, it is also important to review the weaknesses of these two models.

As we mentioned in our analysis, the Popular Game model suffers from the root problem with our dataset: the variance in the number of users for any given game is massive. Even among the top 10 games alone, the **difference between the top most popular game and then 10th most popular game is over 20,000 users**. This means that unless our dataset is extremely large and contains enough data points to calculate a reasonable average for number of users per game, our Popular Game Model is prone to inaccuracies. On the other hand it is arguably the most intuitive model as it predicts a games success on its perceived popularity.

Jaccard Similarity does not face the same weaknesses as the Popular Game Model and provides a similar performance with an ~11% improvement to the baseline model. However, as calculating the Jaccard Index is crucial to proper classification of similarity, it is necessary to have a large and complete Validation Set.

As we think about what features were more useful than others, especially in the context of our SVM models, we start to recognize why our Data Analysis yielded the results it did. In particular, we saw that the **percent of positive reviews per game was a relatively strong feature** to predict on, but that the number of words a review shares with the words in the top most popular games' reviews did not influence our predictions very much. This actually makes sense intuitively given that the text content of **reviews is often very dependent on the writing tendencies of the user**. In other words, what some users might write in order to express positivity, might not be the same as other users. We should also note, however, that our SVM's were performing on subsample sizes of about 15,000 and with unoptimized Penalty Parameters which could have improved their performances, particularly with our Full Feature Vector SVM model.

In conclusion, we've learned that constructing effective predictive models for predictive tasks, even simple ones, often requires the use of features that may not be immediately present. The dataset set presented to us appeared to be quite minimal, which limited our decisions in what features to pull from each data point. However, while we have observed that simple features and models can be surprisingly powerful in predictions (Jaccard Similarity, Popular Game Model), we also observed that more substantial improvements to performance can become quite difficult without more sophisticated methods of feature construction and set manipulation, as is exemplified in our review of related literature.

References:

1. https://www.kgarg.com/files/projects/steam-game-recommender.pdf
2. https://is.muni.cz/th/k2c5b/diploma_thesis_trneny.pdf
3. http://cseweb.ucsd.edu/~jmcauley/pdfs/recsys18b.pdf