Predicting the Success of Board Games

Dan Feldman

danfeldman90@gmail.com

This work is focused on gaining insight into what makes a board game successful, using the dataset created by the website Boardgamegeek.com and its users. We examined whether or not specific features of a board game correlate to its success, and attempted to build a model that can make predictions for the success of new games. The features we used for modeling were weight, year published, minimum number of players, maximum number of players, minimum playtime, and maximum playtime, while user number and standard deviation in voting were also used for feature analysis. To model the data, we built logistic regression and random forest classifiers, using methods to over- and under-sample the data in order to fix a discovered imbalance in the dataset. We found that over-sampling with the SVM-SMOTE algorithm worked best when combined with a random forest classifier, to accurately predict the success of a board game 69 percent of the time on average, based on the average recall (Type-II error). While the precision for all models was low for successful prediction, further examination shows that if ratings are assumed to lead to monetary success, our model would do a very good job of predicting overall monetary success for a board game store selling only games predicted to be successful.

1 Introduction

The modeling and predicting of the success of a product is one of the most fundamental applications of data science to the corporate world. At the heart of any business venture, product success is key; the more successful your product, the more money you'll make. For most products, success can be measured by how well the product works, generally achieving a particular goal. In some cases, the goal might be for practical use, and in other cases, the goal might be for the enjoyment of whomever purchases the product. In the gaming industry, success typically follows when a game is very enjoyable for a wide audience, and so ratings systems can usually be used to rank the successes and failures of any given game. But games come in many forms and genres, and can be defined by a large number of possible characteristics, so is it possible to predict whether or not a game will be successful?

For the sake of focus, we will examine this question for one niche of the gaming industry: board games. Despite this being such a straightforward question–"is this game going to be a success or not?"–there are some nuances that need to be addressed. For example, how does one define success? Does one look at ratings systems, or how many people own the game? If one chooses a ratings system, which one should be used? Moreover, there are many different ways board games can be defined, including by genre, maker, mechanics, or complexity. If we wish to build a model to predict whether a game will be successful, which of these do we consider, and should we consider all of them?

1.1 Potential Clients

These complex questions can usually be answered by your client. In the case of board game success, there are two major clients to consider. First would be the makers of board games. Most companies (or individuals, in some cases) will want to know how they should design their games for success. They might want a heavier focus on things they have control over, like the mechanics and complexity of a game, the genre, or the ability to reach their target audience. For companies that make board games, the metric for success could be how many people own the game, because their goal is to sell as many games as possible.

The second potential client would be the owners of game stores looking to sell board games directly to consumers. For these clients, they might be less interested in how mechanics or genre affect success

but instead be more interested in things like the makers and how previous success affects future success, if hype in the early stages translates to more sales, and if top sellers stay that way or fade out over time. Sellers might be more interested in a ratings system for success, since they might begin carrying a game after it has already been released, or be deciding whether to keep it on the shelves over time. Additionally, if their buyers enjoy the games they purchase at the store, they might be more likely to return for more games.

Because this is a project conducted without a specific client, we will choose to take a somewhat general approach, using the data available to guide which decisions to make in the analysis.

1.2 The Board Game Geek Dataset

The website boardgamegeek.com is a resource for people interested in learning about, rating, reviewing, and discussing board games. Users can browse through an immense catalog of board games, viewing their attributes (game length, complexity, genre, etc.), checking out their ratings and reviews, and even see price listings on eBay as well as BoardGameGeek's (BGG's) own marketplace. By creating a free account, users can create their own game entries; submit photos, reviews, and ratings for games they have played; create profiles with games they own, want to buy, and want to sell; and can post on the forums about any game or topic they want. As such, BGG has amassed a ton of data about board games, and this makes them a great source for a project such as this one.

There is, however, one major caveat to using this dataset. Not all gamers, especially casual gamers, will know about or be willing to use BGG to rate games. As such, there is a selection bias in this dataset, skewing the ratings we acquire to reflect the least casual gamers. This is made clear by a quote on BGG's about page[1]: "Old familiar games like Monopoly exist in the database, as well, although you'll find almost all users prefer games showing the advancements that have been made in game play and component quality since Monopoly was first published." So while this will likely reflect the desires and outcomes for more dedicated gamers, those who just occasionally might want to play a game on Christmas with the family won't be as well represented. As such, potential clients for the model we will create will find the most success if their target audience is mainly dedicated gamers.

[1] https://boardgamegeek.com/wiki/page/Welcome_to_BoardGameGeek

2 Data Acquisition & Cleaning

Obtaining the data for our analysis was a straightforward task, requiring queries to BGG's XML API[2]. The Python 3 script used for making queries to the API is located in the main GitHub repository for this project[3] and is named xml_bgg.py. The script queries board games in increments of 100 games using the entries' ID numbers from the first to the 250,000th entry. This resulted in a total of 2,500 queries, each taking approximately six seconds, for an approximate total acquisition time of 250 minutes (4.17 hours).

In each query, we obtained the following information about each board game: the name, ID number, rating (the Bayesian average of all ratings), weight (BGG's proxy for complexity on a scale from 1-5), number of total users that rated the game, minimum playtime, maximum playtime, year the game was published, minimum players, maximum players, standard deviation of all ratings, and the board game categories listed on the site. Any items in the database that did not have entries for all of those fields were skipped, leading to a total raw database containing 227,196 entries.

Despite this initial cleaning, there were still a few extra steps that needed to be taken to create a clean, workable dataset. The first difficulty to overcome was figuring out how to deal with missing data; for example, of the 227,196 entries, only 33,294 of them even had a value in the ratings field. For our project, the board game rating is the key to the analysis, as it is the value used to determine a game's success. There really wasn't a good way to try and fill in a rating to replace the missing values, so instead, we decided only to keep those 33,294 entries with a valid rating. Though a very large reduction of data, it still represented enough for a robust analysis.

Once the ratings were taken care of, there were still more steps required to clean the dataset. There were more null values in five remaining features: minimum playtime, maximum playtime, minimum players, maximum players, and year published. We removed those entries, and found that there were a number of outliers in each of those features that needed to be removed, as well as the weight feature. The weight was a number from one to five, so all entries with a weight of zero were removed. Both minimum and maximum playtime had entries with zero values, which are unphysical, so they were removed. The same was true for minimum and maximum players; there were also outliers on the high end of maximum players, so we set an upper limit of 20 players maximum. For years published,

[2] https://boardgamegeek.com/wiki/page/BGG_XML_API

[3] https://github.com/danfeldman90/Springboard-projects/tree/master/Capstone_Project1

we set 1900 to be the earliest year in order to remove outliers on the low end. Lastly, we set an upper limit of 1000 minutes (between 16 and 17 hours) for the maximum playtime, which seemed like a conservative number for a board game. After all of these limits were set, the total number of board games became 17,521. For more details of the data cleaning process, see the IPython notebook in the GitHub repository[3] named "Board_Game_EDA.ipynb."

3 Exploratory Data Analysis (EDA) and Inferential Statistics

Before diving into the modeling, it is always good practice to explore the dataset for any possible insights or pitfalls that may need to be fixed. Some steps we wanted to take were to determine if any remaining cleaning needed to occur before the data was ready for analysis, see if any correlations were present with the features and the ratings, and to make some visualizations to see if any additional insights could be gleaned.

3.1 Visualizations

The first thing we wanted to do was to make some plots of the data and see if any insights could be gleaned. Full details regarding this process, including all visualizations, can be found in the IPython notebook titled "Board_Game_EDA.ipynb." Here we will summarize the main findings.

The first thing we found was that the majority of the board game ratings are clustered between five and six out of ten. Additionally, it has a large tail on the high end of the distribution. See Figure 1 to see the distribution in more detail, plotted with a log scale on the y axis for clarity.

Most of the visualizations of features plotted against the ratings were very messy, with good possibilities for correlations but nothing totally concrete. One exception is the user number, shown in Figure 2. There is a clear, positive correlation presented in the data, which seems to indicate an increase in rating as the number of voters increases. This suggests that users are more motivated to rate a game they've played if it is enjoyable, rather than those they do not enjoy.

Figure 3 shows all of the scatter plots between each feature at a glance. On the diagonal are all of the distributions for the features, which can be seen to be very strongly peaked, with the exception of

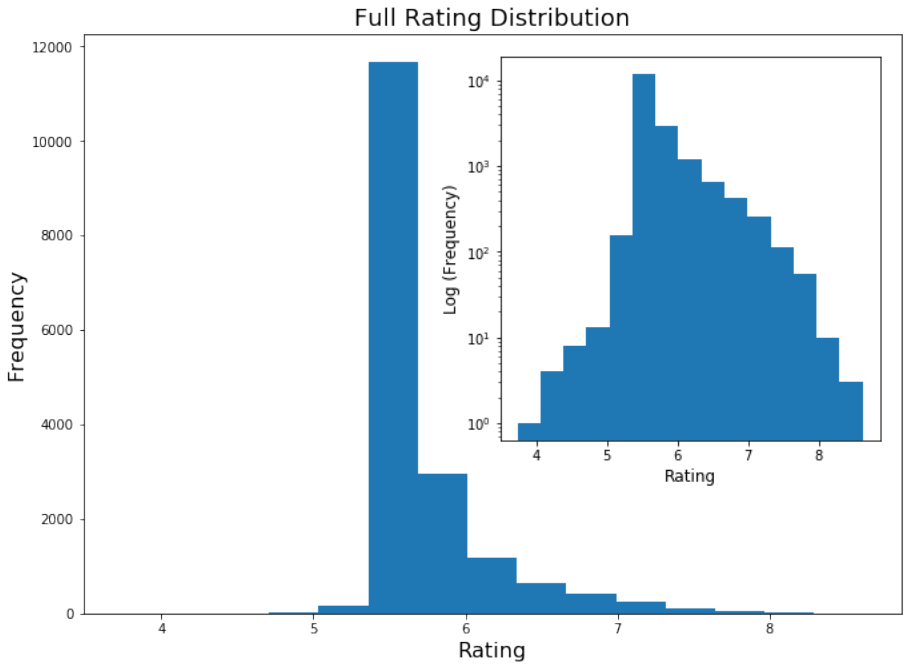

Figure 1: The rating distribution for the dataset. The majority of the entries are between 5 and 6 out of 10, and the distribution has a larger tail on the high end of the ratings. The inset is the same distribution where the y-axis is on a log scale for clarity at the ends of the distribution.

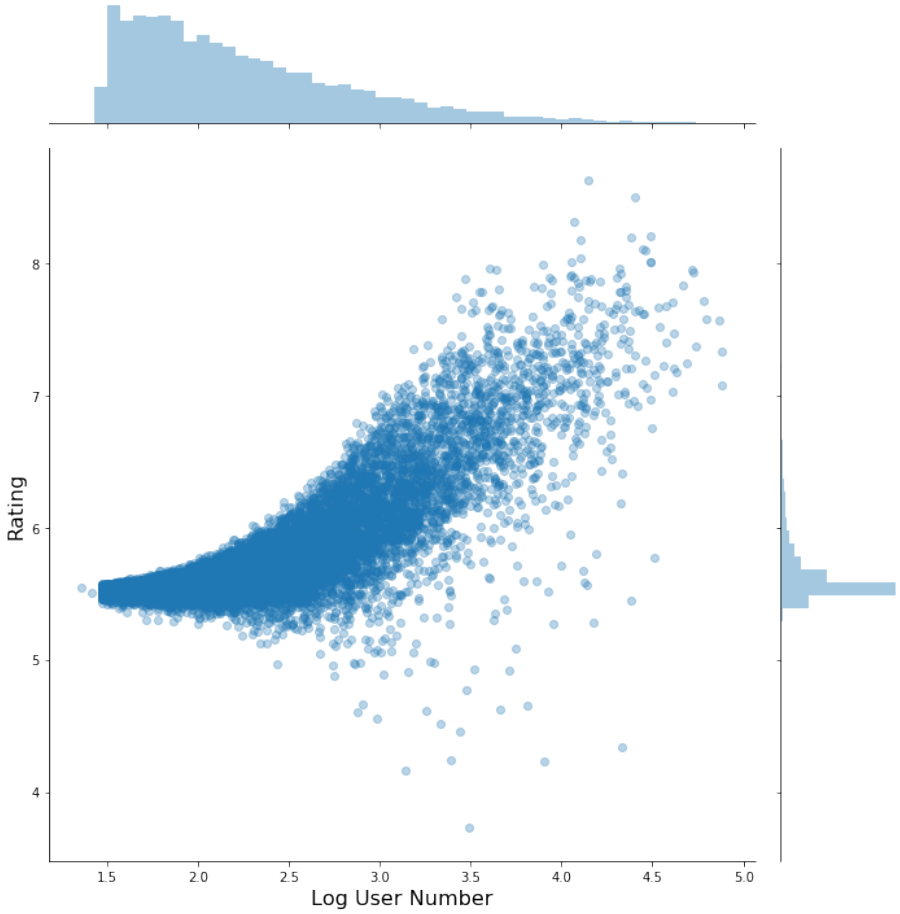

Figure 2: The number of users that contributed to a given board game rating, plotted against the rating itself. The x-axis is on a log scale for better clarity. Histograms on the top and right are the corresponding distributions to scale for the user number and rating, respectively.

the weight distribution. Also plotted are the linear regression lines between each of the plotted features.

3.2 Pearson Correlation Coefficients

In order to more statistically determine if correlations exist between the features in the dataset and the target variable (ratings), we used the pearson correlation coefficient[4]. The scipy.stats module in Python has a built in function, pearsonr(), which calculates the coefficients as well as the p-value associated with these coefficients. The results are shown in Table 1.

[4]https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

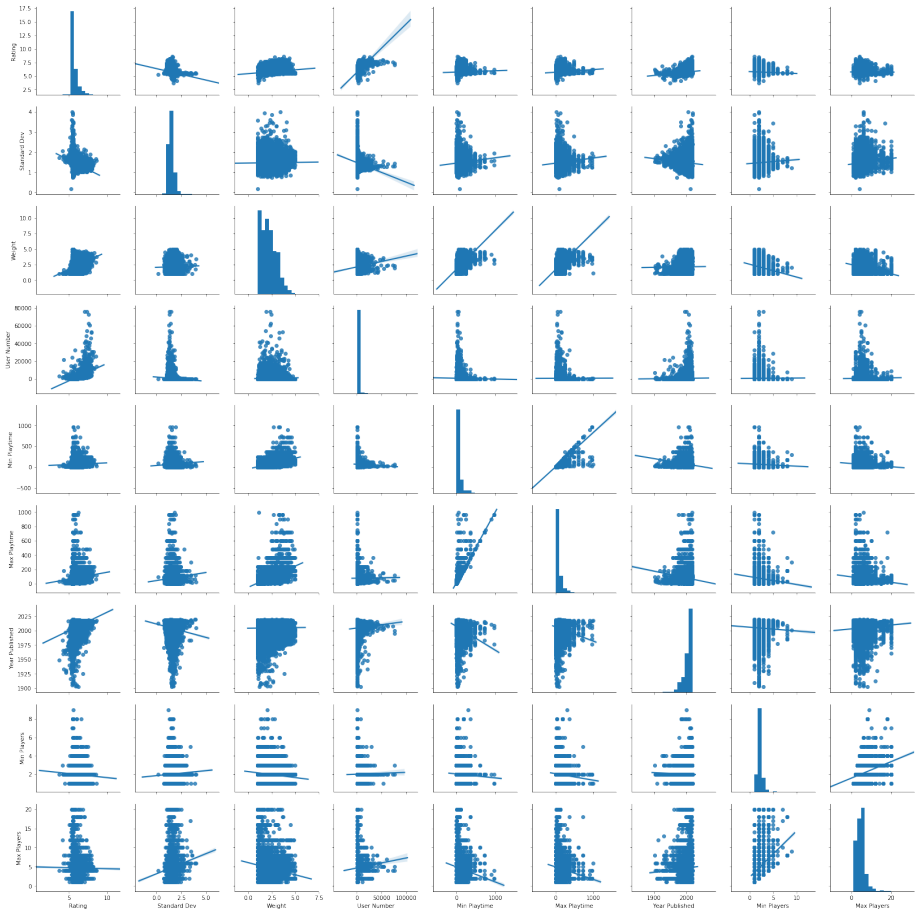

Figure 3: All of the scatterplots between features at a glance, and their linear regression lines. Along the diagonal are the distributions for each feature, which can be seen to be very peaked (with the exception of the weight distribution).

Feature	Pearson r	p-value
Standard Dev	-0.285	0.000
Weight	0.286	0.000
User Number	0.596	0.000
Min Playtime	0.045	3.56e-09
Max Playtime	0.098	1.79e-38
Year Published	0.199	6.70e-156
Min Players	-0.056	1.81e-13
Max Players	-0.009	0.256

Table 1: The Pearson Correlation Coefficients (r) for each feature vs. the target variable (Rating) as well as their p-values. Both were calculated using the scipy.stats function pearsonr.

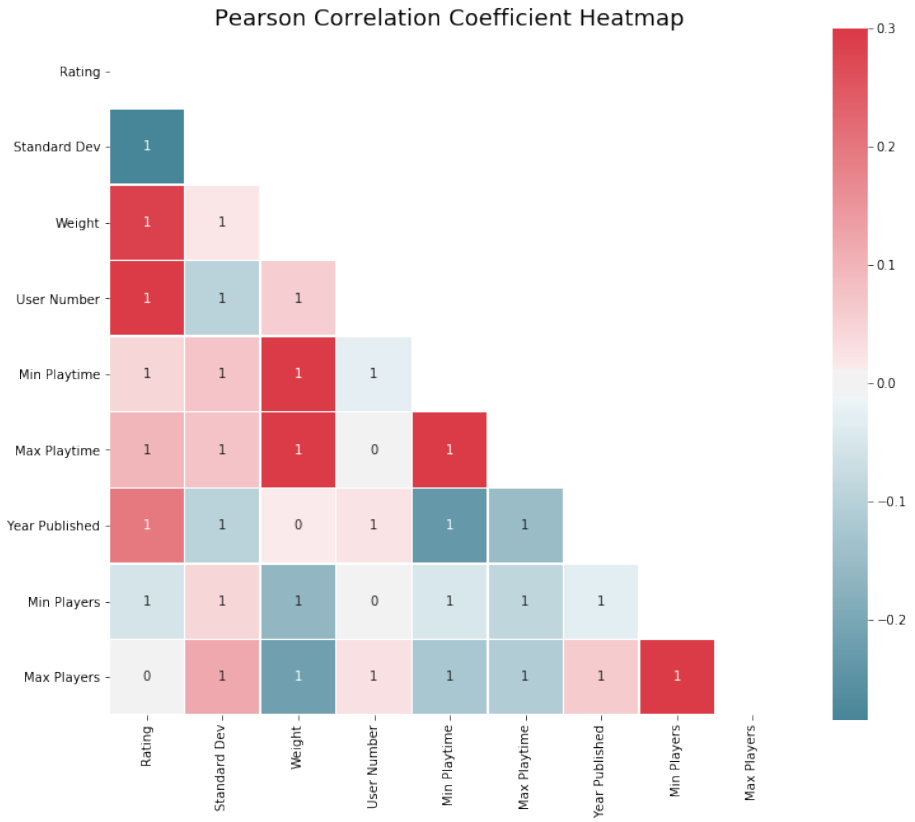

Figure 4: A heatmap of pearson correlation coefficients between all of the features. The more red the square, the more positive the correlation, while bluer corresponds to a more negative correlation. The overlaid ones and zeros represent whether or not the correlation coefficient is statistically significant or not, respectively.

The Pearson correlation coefficients show a few statistically significant correlations in the data. There are positive correlations for the weight, year published, and user number, with extremely small p-values. There is a negative correlation for the standard deviation, with a slight negative correlation for the minimum players. This suggests that generally the BGG crowd prefers newer, more complicated games that they can play with a smaller crowd. Additionally, as mentioned earlier, they are more likely to rate games they enjoy, but the standard deviation results suggest that more polarizing games generally get lower ratings than ones with a consensus.

Figure 4 shows a heat map of the correlation coefficients between all variables. Red corresponds to a positive correlation, while blue corresponds to a negative correlation. The overlaid ones and zeros correspond to whether or not the correlation coefficient was statistically significant or not, respectively. This showcases a few interesting correlations between the features themselves. For example, there is a positive correlation between the weight of a game and its playtime (minimum and maximum), which makes sense; as the complexity of a game increases, it would be more likely to take an increased time to finish playing. There are also similarly obvious positive correlations between minimum and maximum playtime and minimum and maximum players. An interestingly unexpected negative correlation was found between the minimum playtime and the year published, suggesting that in recent years, games have gotten generally shorter than in the past.

3.3 Regression to Classification Problem

Before we began the analysis, we originally considered modeling board game success by predicting the rating a game would receive on BGG, with higher ratings corresponding to higher success. That, however, is a very difficult problem, and it's arguable whether ratings are a perfect one-to-one predictor of success (i.e., a game with a rating of 8.2 is more successful than a game of rating 8.1). That being said, it does seem likely that ratings can be binned up for a general level of success, like that the top 10 percent of ratings are much more successful than the bottom half. As such, we decided to figure out a cutoff point, above which games can be considered successful, and below which games are considered unsuccessful.

To do this, we looked at the distribution of ratings in our cleaned dataset, and chose the top 20 percent of games to be considered successful (True class) and the others to be considered unsuccessful (False class). More specifically, the cutoff rating was decided to be

the 80th percentile, or a rating score of 5.88. By choosing a cutoff point in this fashion, we turned the problem from one of regression to one of binary classification, which makes the task more manageable than a true ratings predictor, and likely a more robust indicator of success.

4 Analysis and Results

In this section, we detail the models and analysis methods used in order to predict the success of board games in our dataset. We began with a simple logistic regression, detailed in Section 4.1, and later expanded to trees using the Random Forest classifier, detailed in Section 4.2. There will also be a discussion about how the dataset is imbalanced, and the under-sampling and over-sampling methods we used to correct this imbalance.

4.1 Logistic Regression

To begin, we attempted to model our dataset using the logistic regression model algorithm implemented in sklearn[5]. Logistic regression offers us a model well suited for binary classification problems, and comes with the added bonus of interpretability. Using this model will allow us to use the coefficients to determine the feature strengths for each attribute in our model.

Before attempting to do any modeling, it became clear that two of the possible features being considered for the model were not useful. Specifically these two features were the number of users that voted on a rating, and the standard deviations in the votes. Ultimately, the usefulness of this model will be in predicting power for new games as they come out; such games will not have any ratings at this point (otherwise, there'd be no need for this model at all), and so the user number and standard deviations will also not exist. As such, we removed those two features from our dataset before continuing with our analysis.

Once those two features were removed, we began using a simple logistic regression implementation. Sklearn's functionality made the training and testing of this model very straightforward, and can be seen in the IPython notebook titled "Logistic_Regression.ipynb." We used a train-test split of 70-30 percent respectively, cross-validating over 5 folds to determine the best regularization strength (C), between the values of 0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, and 100.0. We

[5]http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

decided to use L2 regularization for the computational savings.

In both cases, we discovered that the model was predicting with the accuracy reflected in the percentile cutoff point. In other words, we made the success cutoff at the 80th percentile, and the accuracy was 80 percent. Similarly, if we changed the cutoff to 75th or 85th percentiles, the accuracy adjusted accordingly. The reason for this was very clear when examining the confusion matrix for the regression model. In each case, the model predicted almost exclusively that the board games in the test set would be unsuccessful.

Why would the model be predicting so strongly one classification over the other? The reason is because the dataset is imbalanced. By picking the 80th percentile as the cutoff rating for success, only 20 percent of all the board games belong to the minority class, and 80 percent belong to the other. The model is attempting to minimize the errors in classification, not caring where the errors occur. As such, predicting the unsuccessful (False) class yields very accurate results overall, but is ultimately a useless model.

This imbalance in the data illustrates why accuracy is a poor metric for quantifying the success of our model. Instead, the recall (Type II error) makes for a much better metric for our situation. The recall is essentially a measurement of how well our model can successfully identify the successful board games in the test set. To extrapolate more generally, given a successful board game, it is the percentage our model will accurately identify it as such. Similarly, the recall for the false class serves as the opposite indicator; given an unsuccessful board game, the false class recall is the percentage that our model will accurately identify it as such.

If the classes were more balanced, the precision for the positive class would also be a particularly useful. That metric asks: of all of the games our model outputs as successful, what percentage of them are actually successful? The main issue here with the precision is that given the large amount of actually unsuccessful board games in our test set, even if we correctly predict a large majority of them to be unsuccessful, the false positives will still be a high number compared to the true positives we identify. So the precision will be biased to be lower than it would be for a balanced test set. However, since the recall only compares the true class with itself and the false class with itself, this imbalance will not become skewed the the comparison between the two.

In order to correct this imbalance, we decided to try two approaches. The first approach involves over-sampling the minority class, using the Synthetic Minority Over-sampling Technique, or

SMOTE[6]. SMOTE allows us to create synthetic board games that can be considered successes by taking two successful entries and randomly perturbing the features using a distance measure between them. This can be done a number of times until a given ratio between majority and minority class is achieved.

The second method is to under-sample the majority class. To do this, we randomly assembled a sample of unsuccessful board games from the training set, combined with the successful games in our training set, and used only these to train the logistic regression model. We then repeated this randomization one thousand times, each time tracking the model itself and the classifications it made on the test set. Once all of this was completed, we used a weighted sum to decide on the final classifications, as an ensemble classifier.

More details on these two methods and the results are given in the following two subsections, as well as the IPython notebook titled "Logistic_Regression.ipynb."

4.1.1 Over-sampling using SMOTE

The first method we used to combat the imbalanced nature of our dataset was the Synthetic Minority Over-sampling Technique, or SMOTE. The goal was to balance the numbers for each class by creating new, slightly perturbed entries based on the real data from the successful board games. To do this, we used the open source Python module imblearn[7].

The imblearn (short for imbalanced-learn) implementation of SMOTE contains four algorithms to choose from: regular, borderline SMOTE 1 & 2, and SVM-SMOTE. The imblearn documentation has some useful information regarding the differences in each SMOTE variant, as well as its comparison with other techniques[8]. For our purposes, we trained a logistic regression model using all four of these variants to test which gives us the best results.

An important thing to consider when using SMOTE is when to generate the synthetic samples. Doing so before splitting the data into a training and test set risks testing the model on fake data rather than real entries, and can negatively affect the generalizability for future data[9]. As such, we split our data into a test set before generating our synthetic samples, and using only the training set for the SMOTE algorithm. For the train-test split we used a 70-30

[6] https://jair.org/index.php/jair/article/view/10302

[7] http://contrib.scikit-learn.org/imbalanced-learn/stable/generated/imblearn.over_sampling.SMOTE.html

[8] http://contrib.scikit-learn.org/imbalanced-learn/stable/auto_examples/over-sampling/plot_comparison_over_sampling.html

[9] https://beckernick.github.io/oversampling-modeling/

SMOTE Variant	Accuracy	False Recall	True Recall	False Precision	True Precision
Regular	0.65	0.65	0.66	0.88	0.33
Borderline 1	0.65	0.64	0.67	0.88	0.33
Borderline 2	0.65	0.64	0.68	0.88	0.33
SVM	0.67	0.69	0.61	0.87	0.34

Table 2: The performance metrics evaluating each logistic regression classifier that used the different SMOTE variants. All metrics are rounded to two decimal places after full calculation. The recall values are the most important here for our purposes; the accuracy and precision are recorded for completeness.

SMOTE Variant	Weight	Min Time	Max Time	Year Pub	Min Players	Max Players
Regular	1.0485	-0.0056	0.0012	0.0270	0.0928	0.0475
Borderline 1	1.0220	-0.0043	0.0002	0.0299	0.1124	0.0426
Borderline 2	0.9486	-0.0050	0.0012	0.0280	0.1107	0.0402
SVM	1.2591	-0.0102	0.0047	0.0343	0.1190	0.0327

Table 3: The coefficients determined for each logistic regression classifier using the different SMOTE variants for synthetic sample generation.

percent split respectively, and created synthetic true class samples until the ratio for the training set was unity.

Table 2 summarizes the performance metrics for the logistic regression model using each variant of the SMOTE algorithm. The performances were fairly similar across the four variants, with minor differences in the recall for the false and true classes. The highest recall value for the true class was for the Borderline SMOTE-2 variant, with a recall of 0.68, while the highest average recall was a 0.67 for the SVM-SMOTE algorithm.

Table 3 shows the coefficients determined for each feature by the logistic regression models. Each model shows agreement in the overall trends for each feature, with minor differences in actual value. The strongest feature by far is shown to be the weight, or complexity of the board games, which gain a higher probability of success as the complexity increases. The same is shown to be the case on a lesser scale with the minimum and maximum player features, as well as the year published.

4.1.2 Under-sampling using an Ensemble Classifier

The next method we used to try and improve our model was to under-sample the majority class, since that avoids introducing any fake samples into the dataset. The potential downside of under-sampling is that for any given classifier, you are essentially getting rid of data that might be important. To deal with that, we decided

Accuracy	False Recall	True Recall	False Precision	True Precision
0.67	0.69	0.59	0.87	0.32

Table 4: The performance metrics evaluating the ensemble logistic regression classifier that trained with under-sampling of the majority class. All metrics are rounded to two decimal places after full calculation. The recall values are the most important here for our purposes; the accuracy and precision are recorded for completeness.

to create an ensemble classifier, built out of one thousand individual logistic regression classifiers.

To start, we had to separate out the true and false class entries. From there, we set aside twenty percent of each of those to build the test set to be used for the classifiers. Then, for each classifier, we randomly grabbed 26 percent of the remaining false class entries to use for the training set. When combined with the true class entries remaining, created a training set of 6,546 entries, 3,320 false class entries and 3,226 true class entries. The logistic regression classifier would be trained and then make its prediction for the test set, which would get recorded. The classifier would also get recorded and saved in a serialized pickle file so it could be retrieved later.

For each of the one thousand individual classifiers, we weighed its predictions by its overall accuracy. This is done by turning its true predictions (ones) into the accuracy score itself, and the false predictions (zeros) into one minus the accuracy score. Once all the predictions are made, we tally the votes by adding them all up; if the sum is greater than or equal to 500, then it's considered a success, otherwise a failure. Metrics were then calculated based on this ensemble weighted vote.

Table 4 shows the performance metrics for the ensemble classifier. The recall is no better than that of the one obtained using the SMOTE variants, with an average recall of 0.67. Additionally, because of the nature of the ensemble, it's not possible to obtain the coefficients for each feature. Figure 5 illustrates the breakdown of the voting distribution for the ensemble. It's clear from the distribution that many of the entries in the test set were close to unanimous, with only a few being truly on the border between the classes.

4.2 Random Forest

While the logistic regression classifier was helpful for obtaining interpretable results for feature importance analysis, it is not always the best to use for accuracy of prediction. To try to improve our results, we repeated the analysis performed, this time with Random

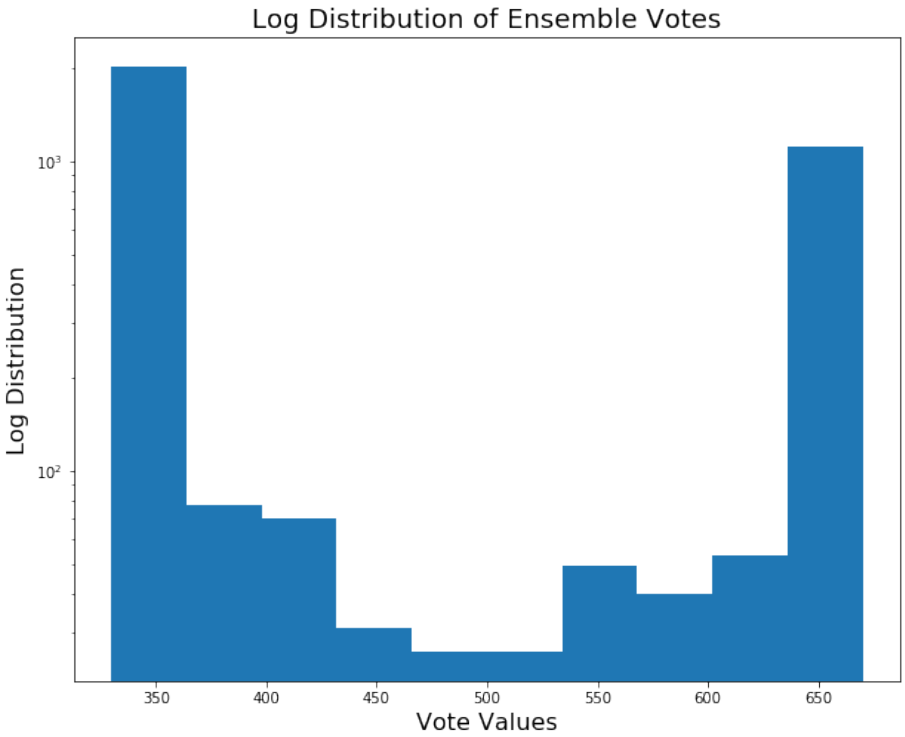

Figure 5: The log distribution of test set vote scores for the logistic regression ensemble classifier. A score of 500 separates the True class (greater than 500) from the False class (less than 500). The vast majority of cases were strongly in one class or the other, with fewer than 100 cases being on the border.

SMOTE Variant	Accuracy	False Recall	True Recall	False Precision	True Precision
Regular	0.69	0.69	0.66	0.89	0.36
Borderline 1	0.69	0.70	0.65	0.88	0.36
Borderline 2	0.68	0.69	0.66	0.88	0.36
SVM	0.69	0.71	0.64	0.88	0.36

Table 5: The performance metrics evaluating each random forest classifier that used the different SMOTE variants. All metrics are rounded to two decimal places after full calculation. The recall values are the most important here for our purposes; the accuracy and precision are recorded for completeness.

Forest classifiers[10].

To begin, we started with the baseline model; specifically, we used the random forest classifier without fixing the imbalance in our dataset. This was purely to repeat the analysis for direct comparison to the logistic regression classifier. To determine features, we used a grid search, with 150, 200, and 250 estimators; max features of 2, 3, and 4; and max depth of 1, 2, and 3.

As was found previously, doing so resulted in a classifier that only ever predicted the false class for the output. Further details can be found in the IPython notebook titled "Random_Forest.ipynb."

4.2.1 SMOTE Revisited

Given the fact that the dataset was found to be imbalanced, we continued the Random Forest analysis by revisiting the SMOTE algorithms used earlier for the logistic regression classifier. We used a grid search to determine the best hyperparameters; the parameter space was the same as for the baseline random forest classifier. As with logistic regression, we used all four SMOTE variants to try and build our classifier. Results are listed in Table 5. The SVM-SMOTE and Borderline-1 SMOTE variants performed the best, with an average recall of 0.69 and a precision of 0.36, which is lower due to the imbalance in the test set.

4.2.2 Ensemble Classifier Revisited

We also wanted to try the random forest classifier with the undersampling ensemble technique previously employed. As with the logistic regression classifiers, we trained one thousand individual random forest classifiers, using two max features, two hundred estimators, and a max depth of three. We used these hyperparameters because they were determined to be the best for the SVM-SMOTE

[10]https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

Accuracy	False Recall	True Recall	False Precision	True Precision
0.70	0.72	0.63	0.88	0.36

Table 6: The performance metrics evaluating the ensemble random forest classifier that trained with under-sampling of the majority class. All metrics are rounded to two decimal places after full calculation. The recall values are the most important here for our purposes; the accuracy and precision are recorded for completeness.

algorithm and wanted to save computing power by not performing one thousand grid searches.

Similar to before, we wanted to weigh the contribution of each individual classifier for the ensemble, so that stronger ones would have greater impact on the overall outcome. To do this, instead of using accuracy score like we did with logistic regression, we used the "OOB" (Out Of Bag) score[11]. After the votes were weighted and completed, they were tallied and determined in the same way as the logistic regression ensemble. Results are shown in Table 6. The distribution of votes are shown in Figure 6. Similar to the logistic regression ensemble classifier, the votes appear to be close to unanimous, with some exceptions along the border.

5 Discussion

5.1 Model Comparisons

For our analysis we essentially created four models with which to predict the successful and unsuccessful board games in the BGG dataset. The first two model types were the logistic regression models, with over-sampling via SMOTE and under-sampling using an ensemble classifier. The second two model types were similarly build, but using random forest models instead. The best way to compare these models are to look at the recall for each class, as well as the average recall.

Figure 7 compares these values for each model. The comparisons between the different methods are interesting in how similar they all are. In terms of average recall, the logistic regression models lagged behind the random forest models, with recalls of around 0.65. This is mainly because their false class recall was comparatively low; however, they in a few cases outperformed the random forest classifiers for the true class. The highest average recall was 0.70, achieved only by the under-sampling ensemble of random forest classifiers,

[11] http://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html

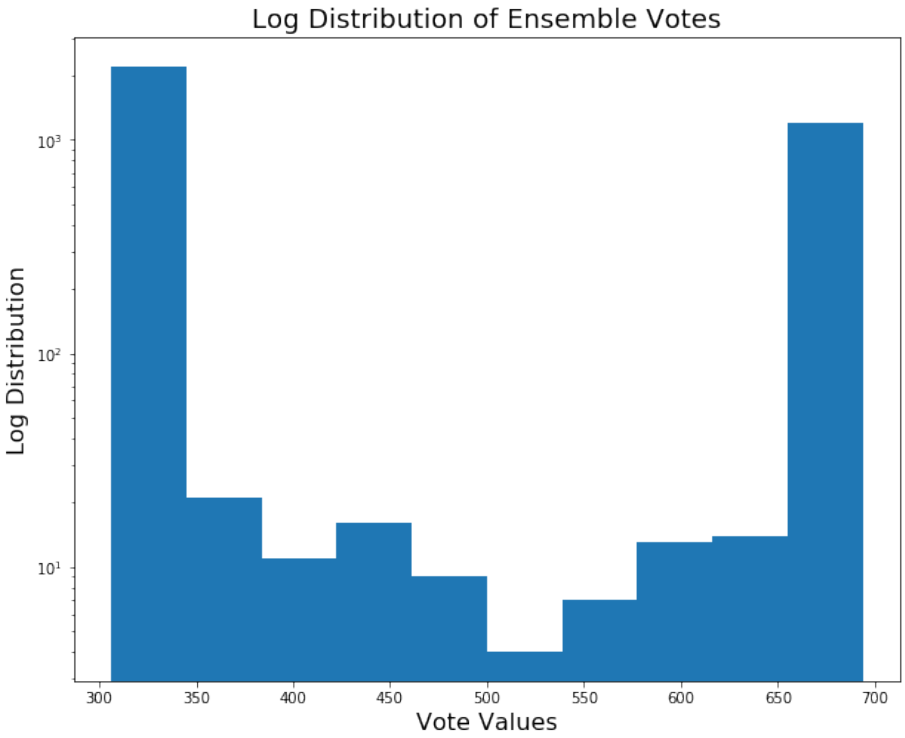

Figure 6: The log distribution of test set vote scores for the random forest ensemble classifier. A score of 500 separates the True class (greater than 500) from the False class (less than 500). Similar to the logistic regression ensemble, the vast majority of cases were strongly in one class or the other, with comparatively few cases being on the border.

Figure 7: A comparison of the average, false class, and true class recall values for each model. The averages are all very similar, with the highest being the under-sampling random forest ensemble method.

due to its superior false class recall (though accompanied by a lower true class recall). This was seen to be true for all of the modeling methods: as the false class recall went up, there was a corresponding lowering of the true class recall, and vice-versa.

One important thing for the logistic regression classifier compared to the random forest is that it is much more interpretable for feature analysis. The coefficients can translate to how important each feature is to the model, and how they affect the output (push the classification to the true or false class, shown by a negative or positive value). The coefficient results listed in Table 3 show some interesting trends. The minimum and maximum player coefficients suggest that players have generally been enjoying games that allow for more people to play at once, such as party games. The correlation with year published harkens back to the original quote from the BGG website, which claimed that most of its users prefer games that are more recent; the data does indeed support this conclusion. Lastly, the strong positive correlation with weight suggests that BGG users enjoy games with more complexity; this makes sense given the nature of BGG users, in that they are less likely to casually game and be more interested in intense games. That being said, the weight distribution compared to weights seen in the EDA IPython notebook does suggest that there is a bit of a sweet spot for a lot of gamers, somewhere in the middle range.

For each classifier, the precision was very low for the true class and very high for the false class. This has more to do with the comparison in numbers between the two classes; even though over- and under-sampling evens out the imbalance in the training data, the test set will still have an imbalance between the two classes. As such, because the false class greatly outnumbers the true class, if they have similar recalls (comparison within an individual class), the precision will skew. However, there is some potentially useful information to glean if we do a deeper dive into the precision, which we cover in the next section.

5.2 Deeper Examination of the Precision

The best metric for our models has been the recall because it most accurately reflects how well our model can successfully classify any individual board game without running into the test set imbalance issue. This notable compares to the precision, which is artificially inflated or deflated for the False and True classes, respectively, because of their relative numbers in the test set. However, a deeper dive into the True class precision can be another important way

to discuss the merits of the models we trained. To look into this, we delved into the results for two of the top performing models: the SVM-SMOTE random forest classifier, and the random forest under-sampling ensemble classifier.

The True class precision in this case can be useful for the following scenario: a board game seller takes a look at all of the games that were released in the past year, and has to decide which ones it will stock up on to sell in the store. If the test set was the list of all the past year's board games, then if the store sold only the games predicted to be the successful class, how many of those games would actually be a success?

At first glance, the 36 percent precision results appears really low, but this is only considering the games in the top twenty percent of ratings. If you assume that the game's overall BGG rating is indicative of success (that is, games with higher ratings make more money as the rating increases), then the top twenty percent of games aren't only games that are making money for the store; instead, those games are the *biggest* moneymakers for the store.

By looking at the ratings distribution for all the games *predicted* to be successful by our model, we can get an approximate view of how much money a store might make (or lose) if it trusted only our model's predictions. Figure 8 shows the ratings distribution for the ensemble model when broken up into four bins. These bins represent those in the bottom 25 percent, middle 25 percent, 50th-80th percentiles, and above 80th percentile ("true" successes). As such, the right most bin reflects the True class precision reported in the results. However, the distribution is heavily skewed to the top 50th percent, which means that over seventy percent of board games predicted to be a success would correlate to at least some monetary success for the client, with 36 percent of the games being very successful. Conversely, only seven or eight percent of games would be major failures, and twenty percent being slight failures. To examine this distribution in more fine bins, see Figure 9, which splits each bin into one tenth of the distribution.

We looked at this distribution for the SVM-SMOTE classifier as well, giving us an additional comparison. Figures 10 and 11 show the distributions with four and ten bins, respectively, using the same cutoffs as with the ensemble classifier distributions. Both models have similar skews to the successful side of the ratings distribution, but the SVM-SMOTE classifier does have a better success-to-failure rate, and would be more effective at making the client money in this scenario. For additional clarity in the comparisons, see Figures 12 and 13. For more details about these distributions

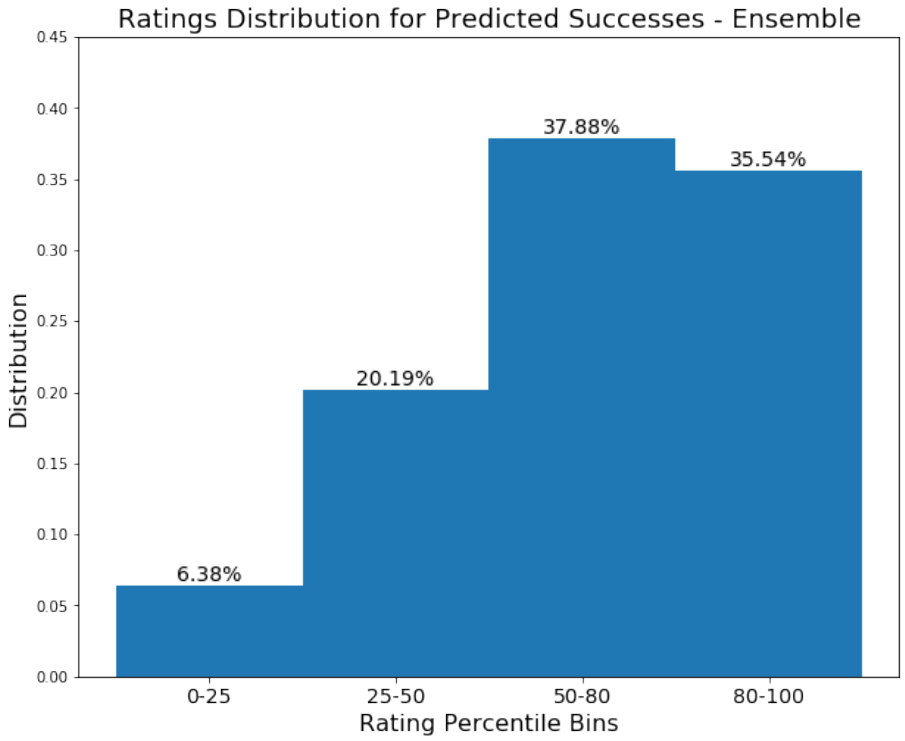

Figure 8: The ratings distribution for the games predicted to be a success by the ensemble classifier. The right-most bin represents the "true" successes, or the games in the 80th percentile and above. The bins from left to right, if you assume ratings correlate with monetary success, represent major losses, minor losses to breaking even, breaking even to minor gains, and major gains. The distributions skew to the right side of the distribution, with over seventy percent being in the bins correlating to minor or major monetary success.

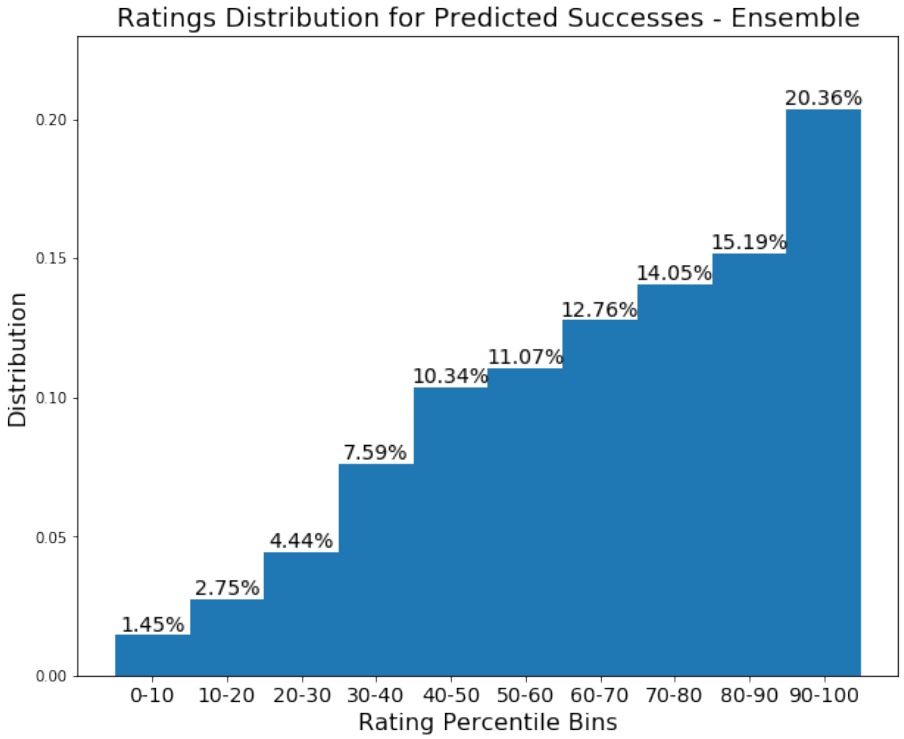

Figure 9: The ratings distribution for the games predicted to be a success by the ensemble classifier, broken up into ten bins rather than four. In this case, each bin represents a ten-percentile range. Again, it's clear that the distribution skews to the right side, which correlates to success.

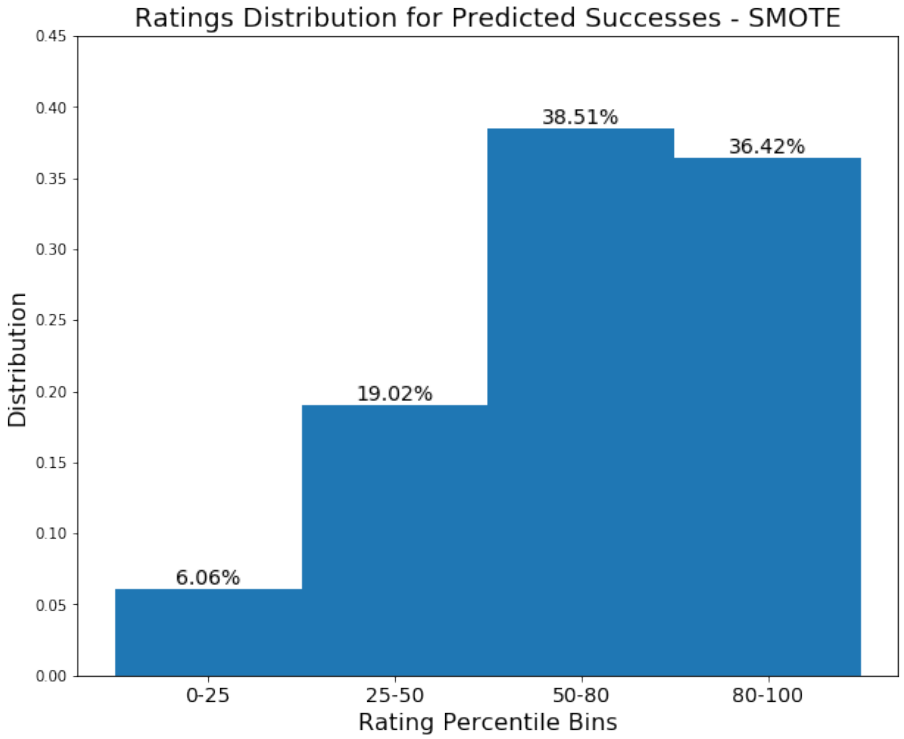

Figure 10: The ratings distribution for the games predicted to be a success by the SVM-SMOTE classifier. The right-most bin represents the "true" successes, or the games in the 80th percentile and above. The bins from left to right, if you assume ratings correlate with monetary success, represent major losses, minor losses to breaking even, breaking even to minor gains, and major gains. The distributions skew to the right side of the distribution, with over seventy percent being in the bins correlating to minor or major monetary success. The SVM-SMOTE classifier has a better distribution than the ensemble classifier, with a greater skew towards success.

and how they were calculated, see the IPython notebooks titled "Random_Forest.ipynb" and "Model_Statistics.ipynb."

6 Conclusions, Recommendations, and Future Work

When we set out on this project, the goals were to see if we could discover insights into board game success using the BGG dataset, as well as to try and build a model that could help hypothetical clients make money off of these insights. Part of this was the intent to see whether or not specific board games features were related to their successes and failures. As a reminder, this dataset does contain a

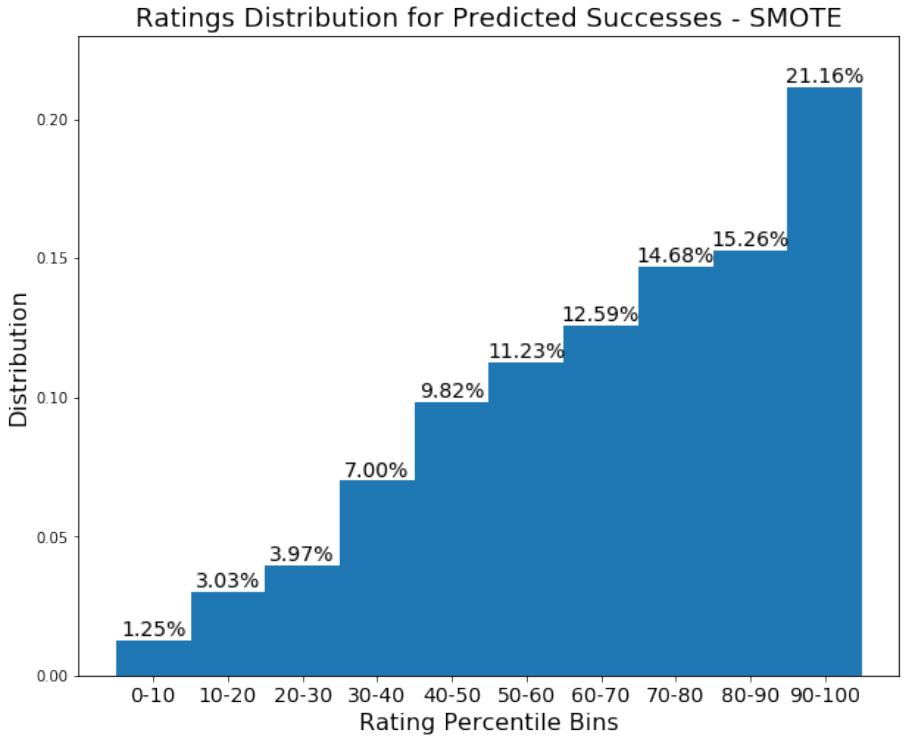

Figure 11: The ratings distribution for the games predicted to be a success by the SVM-SMOTE classifier, broken up into ten bins rather than four. In this case, each bin represents a ten-percentile range. Again, it's clear that the distribution skews to the right side, which correlates to success.

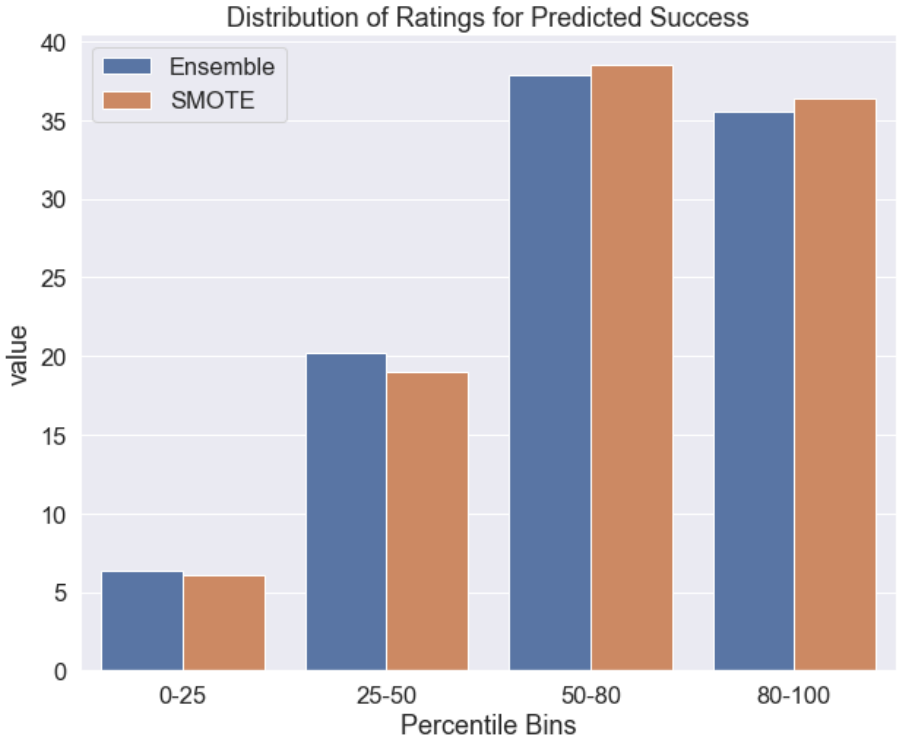

Figure 12: A direct comparison of the 4-binned ratings distributions for the predicted successes for the SVM-SMOTE and ensemble random forest classifiers. It is clear that the SVM-SMOTE results from the random forest classifier are slightly better than that of the ensemble random forest classifier.

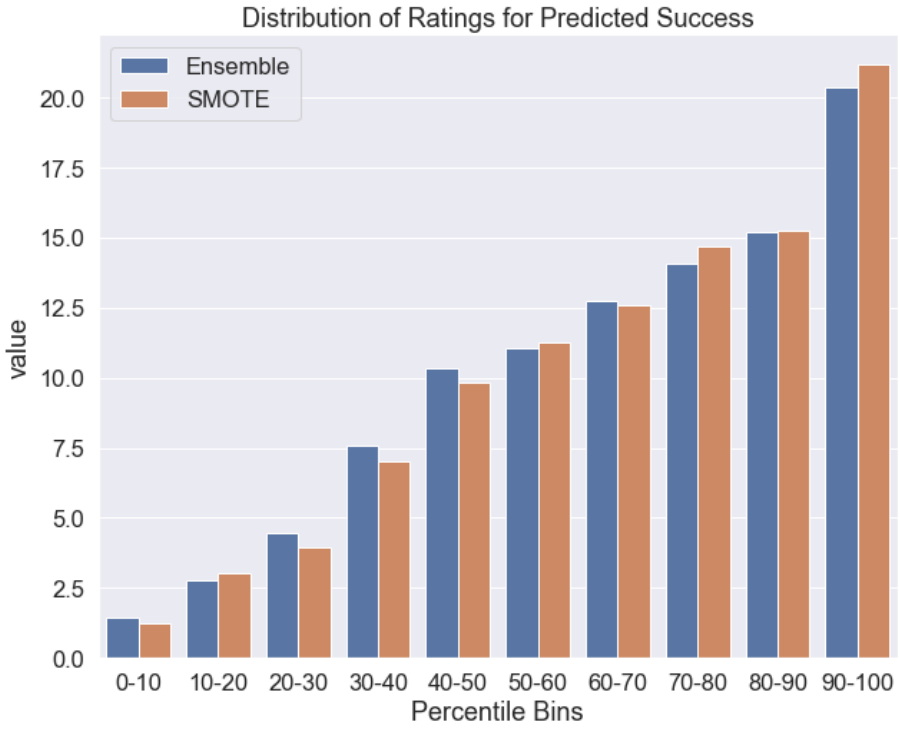

Figure 13: A direct comparison of the 10-binned ratings distributions for the predicted successes for the SVM-SMOTE and ensemble random forest classifiers. This comparison makes it even clearer that the SVM-SMOTE results from the random forest classifier are slightly better than that of the ensemble random forest classifier, especially at the high end of ratings.

likely selection bias, such that the results might not be as applicable for casual gamers than for ones who are much more into the medium as their entertainment.

For our modeling of board game success, we found the best algorithm to be the SVM-SMOTE over-sampling combined with a random forest classifier, with a close second being the under-sampling ensemble methodology, again using the random forest classifiers. However, most of the models we used performed similarly, with average recalls of roughly the same value. In both cases, we achieved a prediction accuracy of roughly 70%, and in examining the precision more closely we determined that the model would be very effective in making money for a board game seller, if one assumes that ratings are positively correlated to monetary success. Given the strong, positive correlation between the user number and ratings in the BGG database, it is not a far stretch to assume that the higher rated games are bought and enjoyed by more players.

An important place for future work to continue would be to try to gain deeper insight into these trends. For one, branching out to other websites with game ratings, such as Amazon, could help correct for any selection biases and make for useful comparisons. Perhaps different gaming communities are served by different types of games.

The results of our exploration and modeling suggest that gamers have been much keener on games recently created than those of the past. As such, they should encourage game makers to continue to create original content, pushing game mechanics to new places, rather than seeking to simply rebrand older games. Gamers using BGG seem to have an affinity for more complex games than simpler ones, though there does seem to be a sweet spot somewhere in the middle, with a number of heavily complex games being rejected by the community. The community also seems to enjoy games with a more intimate crowd, though the correlations aren't that strong, so party games are also likely to be popular.

In summary, we believe that this model can be very useful for a potential client trying to determine which board games will be successful, and therefore potentially useful for determining which games will make the company money if created or sold. On it's own, a board game seller could stand to improve their yield by taking this model into account before deciding which ones to put in their stores. And with some future work for improvements, this could eventually become more accurate and better for serving the board game community, on both the creation and receiving ends of gaming entertainment.

Future work could be done to confirm these suspicions, by taking our model for ratings success and extrapolating it to a model of monetary success. This would be especially useful for potential clients who would like to use this model and have an expectation for risk assessment.

Additionally, future work can focus on looking into additional board game features. We ignored a number of factors that could be important for success, such as the board game maker (and if they've had previous successful games before), whether or not a game was crowd-funded versus privately created, the genre of a particular game, etc. Adding in these features for analysis could improve the model's accuracy and potential monetary yield for a given client.