ORIE 4741 Final Report

Predicting Google Play Store App Ratings

mmd272, mp668, rt359

Contents

1 Introduction

In today's digital age, it is becoming increasingly difficult to create successful mobile applications. Currently there are over three million apps on the Google Play Store alone, creating a saturated and competitive market. We want to investigate if there are certain factors (i.e. category, memory size, price) that developers should focus on to achieve high ratings on the Google Play Store. We focus on ratings rather than number of downloads since often the number of downloads is strongly correlated with the number of marketing dollars spent. For our findings to be meaningful, our findings must be applicable to both a large, high-budget developer and a small, upstart developer.

The data set we are investigating consists of data from nearly 10,000 apps from the Google Play Store and includes features such as category, price, and user reviews. We plan to use the data to predict the rating of a given app.

1.1 Data Preparation

The googleplaystore.csv file has 10,841 rows and 13 columns. We drop two rows due to missing values, leaving us with 10,839 samples. Since we are using *Rating* as the target, we identified all rows with missing *Rating* values and use the imputation strategy outlined in Section 1.3

Out of the remaining 12 columns, we chose to use 10 features and performed the following cleaning and transformation measures:

Column	**Cleaning and Transformation**
Category	One-hot-encoding
Reviews	String to float
Size	We removed the "M" or "k" from every row and replace with the corresponding byte value as a float Then we impute missing rows (1.3)
Installs	We removed "," and "+" from each string and convert to float
Type	We removed a value of "0" and a value of "NaN" and one-hot-encode
Price	We removed the "$" from the *Price* values and converted to float
ContentRating	There is one missing value but it is removed during cleaning of *Type.* We then one-hot-encode
Genres	Many-hot-encoding
LastUpdated	String to float (Rata-Die system)

Table 1: Cleaning and Transformation of Features

1.2 Categorizing Data Types

- Real Valued Data: *Reviews, Size, Installs, Price, LastUpdated*
- Categorical Data: *Category, Type, ContentRating, Genres*
- Text Data: We originally planned to analyze the file googleplaystore_user_reviews.csv. However, after doing some initial analysis, we realized it would not improve any of our models. We matched the reviews csv to the original csv file by app name and discovered there were only 1,020 apps that had reviews. This was roughly only 10% of our data, meaning we would have to ignore the majority of entries in the googleplaystore.csv in order to incorporate text data into our model.

1.3 Missing Values

The *Rating* column, our target, contains 1,474 missing values, which means that 13.6 percent of observations have a missing target value. Because of the high number of missing Rating values, we decided not to simply drop these observations. We also decided against setting missing values equal to the mean of all values. This could introduce bias to our model since the existing *Rating* values are already right-skewed with a mean of 4.19 on a 5 point scale. We decided instead to fit a generalized low rank model to the model in order to impute the missing *Rating* values. We also use the generalized low rank model to impute values in the *Size* column which are equal to "Varies with device," since 1,696 observations have this value (15.6 percent of observations).

We chose to fit a rank 2 generalized low rank model and use Quadratic Loss for the real valued columns: *Reviews, Price, Installs, and LastUpdated* and Hinge Loss for the categorical columns: *Category, Type, ContentRating, and Genres.* The model uses some Quadratic Regularization with lambda=0.1 in addition to scaling and an offset. After fitting the model, we imputed the missing values and got a mean of 4.13 for all *Rating* values. It makes sense that the mean rating decreases with the addition of imputed values since a *Rating* value would be missing likely (1) if it has recently been updated and previously had a low rating before the update (2) is a newer app and has issues that are typical with an upstart app or (3) has no ratings because the app is neither exceptional nor issue-ridden; therefore, users don't feel the need to rate the app.

2 Exploratory Data Analysis

2.1 Data Features

As previously mentioned, each data point in our sample has 13 features. The first group of features are real valued features and include: *Reviews, Size, Installs, Price, LastUpdated.* The minimum and maximum number of reviews are 0 and 78 million. Most of the applications have less than one million reviews, with the mean being about 444 thousand. It is likely only the well-known applications have more than one million reviews. The application sizes range from 0 to 100 MB, and its distribution is right-skewed, with the mean being about 18.5 MB after imputation. The number of installs also follows a distribution that is right-skewed. The minimum and maximum number of installs are 0 and 1 billion, while the mean is about 15.5 million installs. Out of the total number of apps, only 800 are not free. Out of these apps, the minimum, maximum and mean price are \$0.99, \$400, and \$13.92 respectively. Finally, the last updated dates for the apps range from May 21, 2010 to August 8, 2018, with the median date being May 24, 2018.

The next group of features are the categorical features including: *Category, Type, Content Rating.* There exists 33 unique categories of applications in our training set. As seen in Figure 1, Family, Game and Tools applications have the highest market share, with Family having a large lead. The type of the app specifies whether it is free or not. The majority of the apps are free (92.5%) and only a minority are paid for (7.5%). The content rating of the application is the intended audience. The ratings and percentage of apps that falls in each are: Everyone: 80%, Teen: 11%, Everyone 10+: 4%, Mature 17+: 4.5% and Adults only 18+ only has 2 observations while Unrated has only have 1 observation.

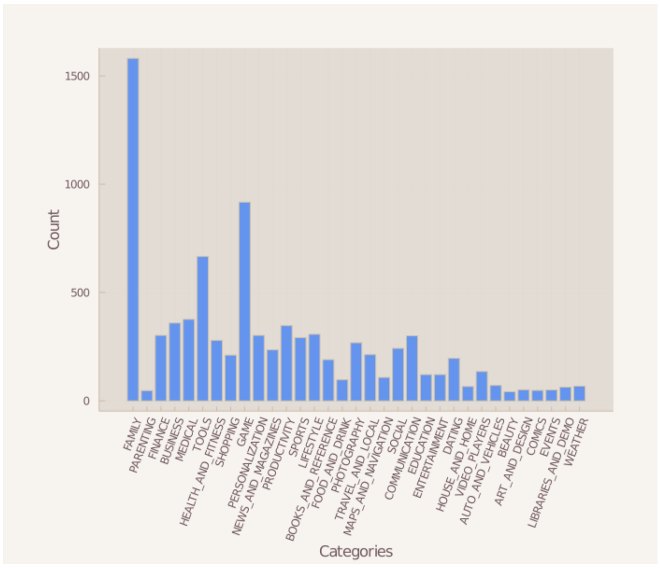

Figure 1: App Category Count

In addition to categorical data, there is also set data contained in the column *Genres.* There exists 53 unique genres of applications in our training set. The maximum number of genres in the set for an app is 2, with the majority, 92%, only having 1.

The final feature we analyzed is *Rating*, which is treated as the label for each observation. The mean rating for the apps without missing values is about 4.19. After imputation, the overall mean rating is about 4.13, and the minimum and maximum are 1.0 and 5.0 respectively. Figure 2 displays the density of ratings for the apps.

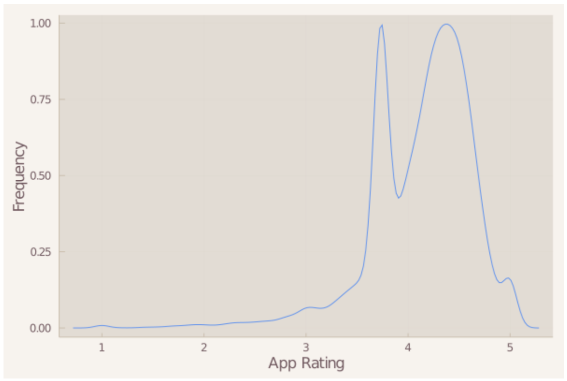

Figure 2: App Rating Frequency

The remaining columns include: *App, CurrentVer, AndroidVer.* These features should not provide anymore insight into the success of an application than the others already mentioned. *App* is simply the name of the app and cannot be incorporated into the model unless natural language processing is done. *CurrentVer* and *AndroidVer* are the version numbers of the applications. These features were not included because they are dependent on other factors such as the age of the app. It is likely that a newer app will have a lower version number than an older app. However, we did choose to include the *LastUpdated* column. This feature is the date of the last update made to the app. We made this decision because it shows whether or not there is an active developer community backing the app. If users are reporting issues and the app has not been updated in months, or even years, users may potentially stop using the app. This feature would not have the same bias towards older apps as version number because both old and new apps should be updated continuously.

2.2 Outliers

While we conducted our initial analysis on the data features, we searched for outliers and decided whether or not they should be removed. Our dependent variable *Rating* has values only in the range 1-5, all of which are valid on the Google Play Store, so there was no need to remove any samples based on the Rating.

Next, we considered our real-valued features including *Reviews, Size, Installs, Price and LastUpdated.* The maximum value for review count was 78 million, and belonged to the app Facebook. We chose to include apps with large review counts because they gave insight into the ratings for popular apps. The smallest value for review count was 0, but this was the most common value so it was not considered an outlier. The same rationale was used for the *Installs* column. The maximum number of installs was 1 billion, which belonged to the app Google Street View. The lowest number of installs was 0, and therefore would provide great insight in the features that could be preventing people from downloading it. For the feature *Size*, there were no true outliers, rather the data was just right-skewed. The lowest price for an app was $0 and was the most common. The outlier in this case was the maximum price at $400. We checked the app store to make sure this number was correct. Being that it was, we chose to keep this sample to help the app generalize well for a large range of prices for future use. The app that has not been updated in the longest time was last updated in May of 2010. The app had a rating of 4.2. We chose to keep this value so our models could gain insight into what other features can help generate a high rating for an app if it is not the *LastUpdated* feature.

Finally, we considered the categorical features including *Category, Type, ContentRating and Genres.* We calculated the counts for each *Category* of apps and found that the smallest category had 36 apps. While this number was low relative to how many samples we had, it means there is a decent chance that more apps will be added to this *Category* is the future. While the majority of apps had *Type* free, there were enough paid apps that we did not need to consider them as outliers. The next feature we observed was *ContentRating.* There were a few outliers here. Only two apps were placed in the Unrated category and only one was labeled as Adults only 18+. We ultimately decided to keep these samples because it is very possible that if they currently exist in some capacity in the Google Play Store, that more apps will be added to these categories in the future. In addition, if an app is missing this label, it can easily be placed under the Unrated category. The final feature was *Genres*. There were some genres that only had one app. We used the same logic that new apps added to the Google Play Store could easily be given these genres in the future.

2.3 Variable Correlation

As seen in Figure 3, the largest positive correlation exists between the number of *Reviews* and the number of *Installs*. This means that these variables tend to move in the same direction. If an application has a large number of reviews, it tends to have more downloads. On the other hand, if an application has a large number of downloads, customers tend to leave a review. This signifies that app owners should encourage reviews in order to increase the number of downloads, and potentially the popularity of the app. Another important observation is that the *LastUpdated* variable has the largest positive correlation with *Rating*. If the app has been updated recently, the rating of the app tends to increase. This will also help app owners increase their app's success in the Google Play Store. The *Size* of the app and the number of *Installs* for an app also display a positive correlation with *LastUpdated*. The remaining correlations in Figure 3 are all less than 0.1, suggesting that the features do not follow large trends.

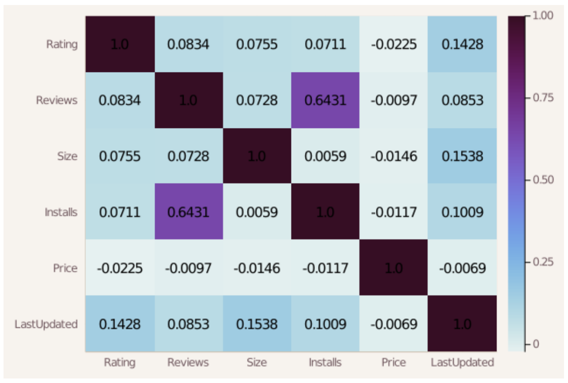

Figure 3: Real-Valued Feature Correlation

3 Methodology

3.1 Introduction to Models and Analysis

In order to find the best fitting and most accurate predicting model, we tried several linear models using Least Squares Regression and regularized models that optimized for Quadratic Loss, L1 Loss and Quantile Loss. We performed this analysis on different combinations of features until we found the features that consistently produced low training and test errors. We also fitted non-linear models such as Decision Trees and Random Forest which ended up yielding the smallest errors, suggesting that our data has non-linear trends.

3.2 Use of Mean Squared Error

Mean Squared Error (MSE) measures how close a regression line is to a set of data points by squaring the distances from the true output to the regression line. Squared distance is able to disregard a negative difference in the prediction and true output and give more weight to outliers. In our model, we decided that our error metric should be the MSE because of the small range seen in the dependent variable, *Ratings*. Our data-set only included *Ratings* in a range from 1 to 5, with a large majority around 4. Therefore, we wanted to ensure that small differences were considered significant by our models.

3.3 Prevent Overfitting

In order to prevent our models from over-fitting we shuffled and split our data 80:20 into train and test sets. If the models are able to generalize well, we expect the training loss to be only slightly higher than the test error. However, there were occasions when some of our models had test errors slightly lower than the training error. One explanation for this to occur would be that the test set sample is too small. However, given the large size of our data-set, we concluded that this phenomenon is likely a result of having less noise in the test set. In other words, the training set may have had more "hard" cases to learn while the test set had mostly "easy" cases to learn. In most cases, after re-shuffling and re-splitting the data, in a manner similar to cross validation, the result changed to what was expected.

3.4 Prevent Underfitting

In order to prevent underfitting we made sure to include a large number of features in our analysis as well as a large number of data samples. In addition, we avoided choosing models in which the train and test errors were both high.

4 Least Squares Regression

Least Squares Regression is a linear estimation of the best fit for a set of data points. It works by minimizing the sum of the offsets or residual points from the plotted curve. We used Least Squares Regression to predict the behavior of our dependent variable *Ratings* based on our other data columns, the independent variables.

	Train MSE	Test MSE
Real-Valued	0.24520	0.25278
+ One-Hot	0.23680	0.24655
+ Many-Hot	0.23451	0.24440
Bonus	0.23611	0.24560

Table 2: Least Square Training and Testing Error

4.1 Real-Valued Features

After cleaning and transforming our data into the correct data types, we created train and test data sets by splitting our data 80-20. We also shuffled our data before the split to limit the chance of having a different data distribution in the data-sets. We ran our first regression on *Ratings* using only the real-valued features: *Reviews, Installs, Size, LastUpdated, and Price* and an offset. The resulting train and test mean square error can be seen in Table 2. The training and test errors were very similar, signifying that this model was able to generalize well. The resulting plot for the test set can be seen in Figure 4. The range of the predicted ratings for the test set is rather small from about 3.6 to 4.8. The majority of the predictions fall between 3.7 and 4.6 which was expected since the mean rating for the entire data-set was 4.13.

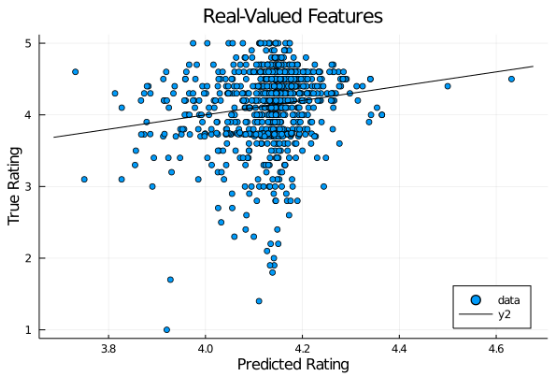

Figure 4: Least Squares Regression - Real-Valued

4.2 Categorical Features

We then added one hot encoding for all category type columns *Category, Type and ContentRating.* The percent reduction from this change is $\approx 3.4\%$ for training error and $\approx 2.5\%$ for testing error. Finally, we added many hot encoding for our column *Genres* to create a vector for every entry. This reduced our training and test errors very slightly, as shown in Table 2. The resulting plot for the test set can be seen in Figure 5. The range of the predicted ratings for the test set using all of the data features expands slightly from just the real-valued features. The lower bound decreases to 3.5 to include slightly lower ratings and the upper bound increases to 4.8 to include slightly higher ratings. The mean rating prediction for this model is 4.13, which is the true mean of the data-set.

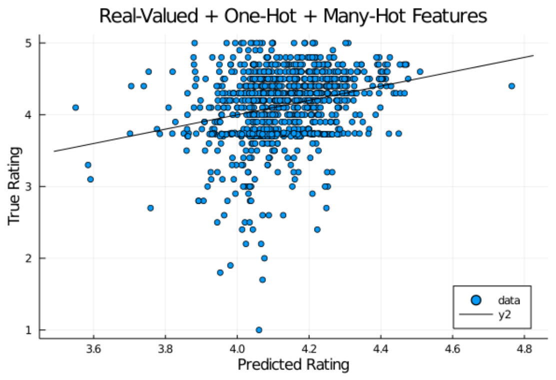

Figure 5: Least Squares Regression - All Features

4.3 Bonus Model

In attempt to further improve our least squares regression models, we decide to train and test using different combinations of features. We start this analysis by building regression models for each feature in our data-set individually. We also built regression models to test the possibility of polynomial relationships with the real-value features. We tried both squaring and cubing the real-value features. However, this did not yield stronger results. We are able to achieve the lowest training and test mean squared error using the following features individually: *LastUpdated, Genres and Categories.* As we saw in the correlation plot in Figure 3, the *LastUpdated* feature has the strongest correlation with *Rating*, so it is no surprise that it is able to predict *Rating* particularly well. Therefore, we create a model with these three features and an offset. Our resulting training and test MSE are 0.23611 and 0.24560 respectively as seen in Table 2. While these results are very similar to our errors for the model trained using all the data features, this model does not produce lower scores. The resulting plot can be seen in Figure 6. The mean rating prediction for the test set remains at 4.13 but the range of predictions slightly changes in comparison to the model using all data features. The lower bound remains at 3.5 and the upper bound decreases to 4.5.

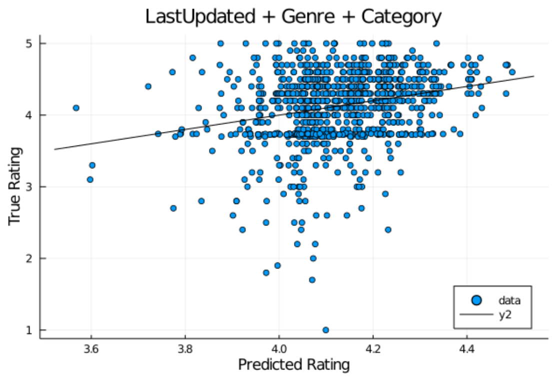

Figure 6: Least Squares Regression - Bonus

4.4 Evaluation

The data for Ratings ranges from 1 to 5 points with the mean value occurring at 4.13. Our least squares regression models predict ratings within a small range that includes this mean value. While the error is rather low for our data-set, we need to consider other factors before deciding whether or not it would perform well on new inputs. For instance, if the model was given a data-set that has a much lower true *Rating* mean, would the model make strong predictions? Given the large size of samples we trained with, it is very likely that our data-set is a strong representation of the entire population of Google Play Store Apps. Therefore, these least squares regression models should perform well on new inputs.

5 Proximal Gradient Method

In addition to least squares regression, we considered proximal gradient methods to optimize our model using a variety of loss functions and regularizers. In particular, we explored 3 loss functions—L1 Loss, Quadratic Loss, and Huber Loss—and various regularizers with each.

5.1 L1 Loss

We started by investigating L1 Loss with Quadratic Regularization. This yielded a training error of 16.41 and a test error 16.52, which is significantly higher than the testing and training error we observed with least squares regression. This model also yielded *Rating* values ranging from 0 to 9, however we need them to be in the range of 1 to 5.

We next experimented with some individual features to investigate this high error. We found that 3 features in particular—*Reviews, Installs, and Size*—all real-valued data, contained the highest error. *Reviews* in particular had a training error of 17.60, a test error of 18.06, and a spread of *Rating* values from 1.98 to 41.39. We tried removing these features from the model and got a training error of 0.26 and a training error of 0.23. The spread of the predicted ratings was small, however, with the minimum *Rating* being 4.19 and the max being 5.84. See Figure 7.

We next investigated using a different regularizer, namely L1 Regularization, and added back all the features. This yielded a training error of 0.26 and a testing error of 0.23, however we still had a small spread of predicted *Rating* values with a minimum *Rating* of 4.18 and a maximum of 4.20. See Figure 8.

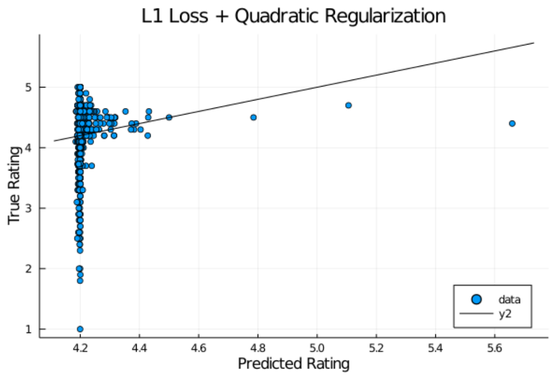

Figure 7: L1 Loss and Quadratic Regularization

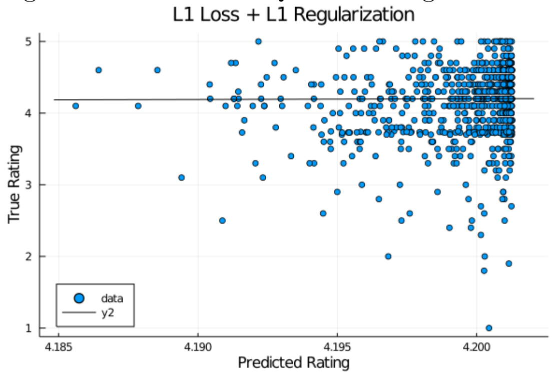

Figure 8: L1 Loss and L1 Regularization

5.2 Quadratic Loss

We next tried using Quadratic Loss to see if this would help increase the spread of our *Rating* values and reduce both the training and test error. Using no regularizer, we obtained high training and testing errors, similar to L1 loss with Quadratic Regularization, with a training error of 0.35 and a test error of 0.33. Since Quadratic Regularization would likely inflate the effect of certain outlier real-valued observations, as we saw with L1 Loss and Quadratic Regularization, we disregarded it in this case.

We saw that using L1 regularization also resulted in high training and test error; therefore, we decided to consider other loss functions.

5.3 Huber Loss

Since Huber Loss is generally less sensitive to outlier data, we tried it next. When we used Huber Loss with L1 regularization and no regularization, we got errors only slightly better than in the Quadratic Loss case. However, when we removed certain real valued columns—*Reviews, Installs, and Size*—we got low errors, as in the L1 loss case. Our training error was 0.26 and the test error is 0.23. Our spread of *Rating* values, however, was still small, with the minimum *Rating* being 4.14 and the max being 4.16. See Figure 9.

To see if we could increase the spread of the predicted *Rating* values, we investigated the individual features to see how they performed on their own. The features with the lowest errors were *Category, ContentRating, Genre,* and *LastUpdate*. One interesting observation was that using *Category* gave almost identical training and test error as above, 0.26 and 0.23 respectively. However, it predicts every *Rating* value to be 4.05. See Figure 10.

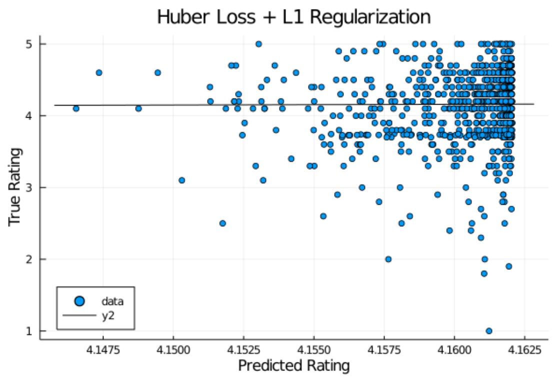

Figure 9: Huber Loss and L1 Regularization

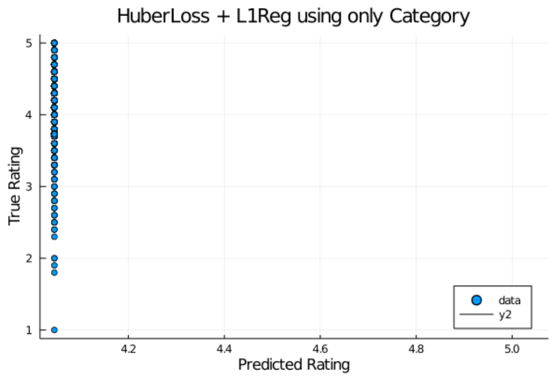

Figure 10: Huber Loss and L1 Regularization (Category Only)

5.4 Evaluation

	Train MSE	Test MSE
L1 Loss	0.26	0.23
Quadratic Loss	0.35	0.33
Huber Loss	0.26	0.23

Table 3: Best Proximal Gradient Training and Testing Error

Overall we saw that Least Squares Regression performed slightly better than all the Proximal Gradient models obtained using a variety of loss functions and regularizers. In addition, the spread of *Rating* values is very small and does not cover lower *Rating* values. Therefore, in practice we would not consider any of the Proximal Gradient models.

6 Non-Linear Models

As we have seen above, we built several regression models, all of which explored linear relationships with the features and app ratings. Therefore, next we decided to explore some non-linear models including decision trees. The goal of implementing decision trees is to create a model that predicts the *Rating* for a given app by learning simple decision rules inferred from the independent features.

	Train MSE	Test MSE
Decision Tree (Depth=7)	0.19891	0.20887
Reduced Tree (Depth=7)	0.20107	0.20531
Random Forest (Depth=7)	0.18826	0.19631

Table 4: Non-Linear Training and Testing Error

6.1 Decision Trees

The first non-linear model we built was a Decision Tree using the *DecisionTreeRegressor* from Scikit-Learn. For our first model we kept the default hyper-parameters. This meant that the criterion to measure the quality of a split was left as mean squared error. Also, there was no max depth provided so the nodes were expanded until all leaves were pure. We ran this model on all of our features combined to get a training MSE of $3.41697e^{-6}$ and a test MSE of 0.34855. Our model was massively over-fitting. In order to improve the model, we took two approaches. First, we experimented with the hyper-parameter max-depth. We tested values 1:10. The goal of analyzing the results with different training and test errors was to find the lowest training and test error to ensure our model was not over-fitting and would perform well on data it had never seen before. At a max depth of 7, this was achieved. The training MSE was 0.19891 and test MSE was 0.20887. Figure 11 displays the predicted ratings versus the true ratings for each app. The range of predictions for the test set using this model is from about 2.8 to 4.7, much larger than was seen in the linear models.

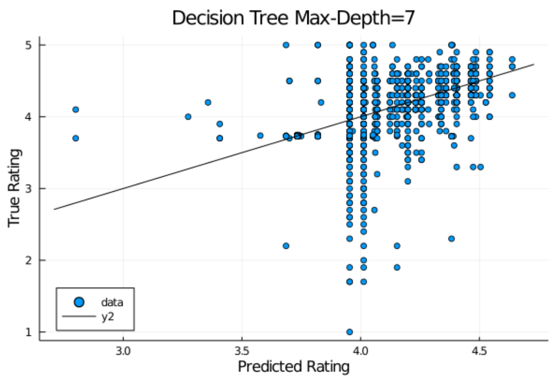

Figure 11: Decision Tree Model (Max Depth of 7)

In attempt to further reduce our test error, and give the Decision Tree a better chance of finding features that are discriminative, we performed feature selection. Given that we were already using a Scikit-Learn model, we chose to use a built in function rather than manually observing the error with different combinations of features. We chose to use the feature selection package *SelectFromModel*. This transformer selects features based on importance weights from the Decision Tree model with max depth of 7. The first step is to determine what to set as the threshold value to use for feature selection. We print out the array that assigns each feature an importance value from the decision tree. Out of the 100 features we have (most from one hot and many hot encoding), there are only 33 non-zero weights for this model. The mean feature importance value is about 0.01. Therefore, we experiment with thresholds close to 0.01. After observing the training and test error for several thresholds including: 0.005, 0.01, 0.05, 0.1, 0.15, we see the best performance for 0.01. This threshold leaves us with ten features and produces a training MSE of 0.20107 and a test error of 0.20531. While the training error did increase slightly, the resulting test error decreased slightly as well, meaning the model was able to generalize better. The range of predictions remained the same from the previous tree.

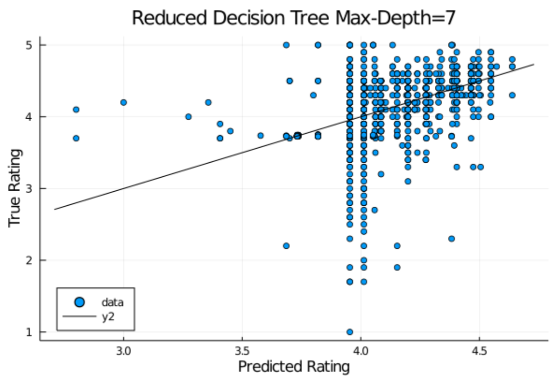

Figure 12: Reduced Decision Tree Model (Max Depth of 7)

While this transformation only produced a slight decrease in test error, it is important to note other improvements such as time and space complexity. Fitting the model without feature selection took 0.03532 seconds and with feature selection took 0.01377 seconds, which could potentially save a lot of time if this project were to be scaled to include a larger dataset. In addition the size of the training data decreased from 0.87577 *MB* to 0.06937 *MB*.

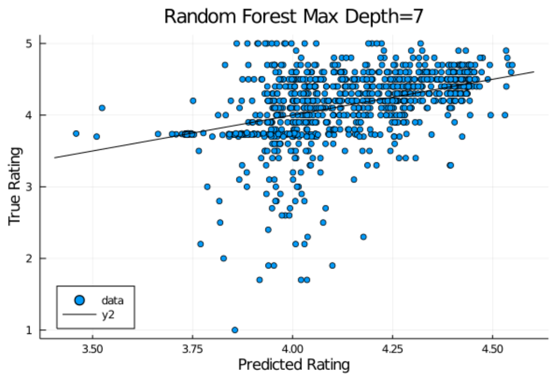

Figure 13: Random Forest Model (Max Depth of 7)

6.2 Random Forest

In a final attempt to improve the stability and accuracy of the Decision Tree model, we decided to implement bootstrap aggregating (bagging) via the *RandomForestRegressor* model from Scikit-Learn. This model fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. Again, we perform an analysis to find the best max-depth for the model and find that after a max depth of 7 is reached, the model begins to overfit. At a max depth of 7, the train MSE is 0.18826 and a test MSE of 0.19631. This is the best test error rate we have seen thus far. The model can be see in Figure 13. Again, we perform feature selection but are unable to achieve a lower test error rate. The range slightly decreases to 3.45 to 4.55 when modeling with Random Forest.

6.3 Evaluation

The non-linear models were able to produce a larger range of ratings for the test set than the linear models were able to. This resulted in producing the lowest test and training errors for our data-set. These results suggest that some of the data features do not have a linear shape. Therefore, the Decision Tree and Random Forest are able to capture the non-linearity in the data by splitting the data-set into smaller and smaller subsets.

7 Weapons of Math Destruction

7.1 Weapons of Math Destruction

When deciding whether a model and its outputs are a Weapon of Math Destruction we must consider the following questions:

1. **Are the outcomes of the model hard to measure?** No, app ratings are a numerical value and therefore are quantifiable.

2. **Could the model's predictions harm anyone?** No, an app rating is harmless. However, there is the chance that predictions could slow innovation in the cases where apps receive bad ratings.

3. **Could the model help create a feedback loop?** The information given by the model could create a feedback loop. This is because it is likely that the apps that are already popular will be predicted to receive a higher rating and this will probably lead to more downloads and higher rating too. The opposite would be expected to happen to unpopular apps. This way, this model could make already popular apps more popular and unpopular apps less popular.

7.2 Fairness

We did not have information on the creator of the app and this could bias ratings. For example, an app created by say Elon Musk is likely to receive better ratings - due to the pedigree of the creator- than the same app created by one of us three. Having this data could definitely change our model.

Furthermore, we were apprehensive about using *Installs* as the dependent variable because it is highly correlated with the amount of money spent in marketing. We saw in the correlation plot that *Installs* was not highly correlated with *Rating*, which means that *Rating* is not as affected by marketing budget, so we saw this as a more fair measure.

8 Conclusion

	Train MSE	Test MSE
Least Squares Regression	0.234510	0.24440
Proximal Gradient Descent	0.26	0.23
Random Forest (Depth=7)	0.18826	0.19631

Table 5: Best Overall Training and Testing Error

Our Random Forest model with maximum depth 7 has the lowest testing and training error, and we would recommend using it to make predictions. This way, we are pretty confident that the results we got are correct.

We would be willing to use this model in a company setting for a variety of purposes, whether we are in the process of building a new app or are trying to improve the *Rating* of an existing app. If it is the latter case, we can try changing our existing features to see the projected *Rating*. For instance, we can double say the *Reviews or Installs* and see if it improves our *Rating* significantly. If so, then we can brainstorm ways to be able to get increase these features, whether that is via marketing or not. We could also experiment with listing our app under different *Genres or Categories* if it makes sense given the nature of the app.

On the other hand, if we are trying to develop a new app, we would not have certain features such as *Installs or Reviews.* However, the bonus model that is trained using least squares regression only includes *LastUpdated, Genre and Category.* These are all features that would be known for an app still in development, with the *LastUpdated* being the day they plan to launch the app on the Google Play Store. While we did not include scores for the Random Forest model for this combination of features, it is very likely that they would produce strong predictions given the success of the model on all of the features.

9 Future Improvements

The most significant way in which we could improve our model would be increasing the number of features. We were planning on conducting sentiment analysis on written user reviews, but unfortunately we did not have enough reviews to reach any meaningful conclusions. Gathering enough written reviews to be able to conduct sentiment analysis and reach meaningful conclusions would definitely improve our model.

In addition, we only explored one non-linear model, Decision Trees, and we could possibly improve our model by exploring other supervised learning models, for example Neural Networks, Multi-Layer Perceptrons, Naive Bayes Regression or Nearest Neighbors Regression.

Finally, we could compare this data to Apple store data and see if it could give us any more insights and include whether the apps have different versions for different devices (i.e. tablet, smart watch). More importantly, it could provide insight into whether an app would be more successful in terms of *Rating* on the App Store or the Google Play Store.