JOURNAL OF
EMERGING INVESTIGATORS

Article

Can the attributes of an app predict its rating?

Nicolas Feng[1], Emily Ryu[2]
[1] Central Bucks HS South, Warrington, Pennsylvania
[2] Cornell University, Ithaca, New York

SUMMARY

When Google Play Store developers first publish their apps, it may seem that their success comes from the roll of a dice. Ways to improve the rating of their app may seem impossible to glean, with many relying on blind guessing to determine ways forward. In this paper, we hypothesize that the attributes of a Google Play Store apps, including maturity level, install count, and price, can estimate its rating, or stars out of five. We believed that comparing three models trained on these attributes, each with unique architectures, would lead to a higher final accuracy than one. By using Random Forest (RF), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) classifiers, we found that the attributes of an app can predict its rating, with review count, date of last update, and storage size being the most influential attributes. During modeling, we found that the RF classifier most successfully predicted the rating of an app, getting 79.3% of predictions correct. These results suggest a connection between the rating of an app and its attributes. The results from this paper can inform app developers and investors about improvement paths to increase the rating of their app, thereby increasing its success.

INTRODUCTION

Many developers turn to the Google Play Store when marketing their mobile apps. The store is where Android users interact and install these apps, which include newspapers, video games, and streaming services (1). Success on the Google Play Store, while subjective, is primarily determined by app rating, which many developers work towards maximizing. Benefits of increased app ratings include improving first impressions and gaining visibility to potential users, increasing usage (2). Better-rated apps also have higher chances of being featured on the Google Play Store's front page, further increasing its audience. For developers, controlling the rating of their app is nearly impossible, since it's entirely user-controlled and is difficult to predict. How they should improve their app to influence their rating is also hard to ascertain.

A 2014 paper predicted the success of Google Play Store apps using a combination of install count and a rating out of five using a generalized linear model (3). However, this model did not implement a more holistic range of app attributes, instead opting to choose two more readily interpretable attributes. This model used a self-proposed equation to combine these metrics into a binary successful/not successful result, instead of predicting a metric already defined in their scrapped data. This methodology, while useful for training purposes and simplifying their success equation, may not be entirely accurate in its predictions of actual market success, as the paper arbitrarily categorized successful apps as within the 95th percentile of ratings and installs. This percentile was chosen to address clusters of high installs from developers like Google, Facebook, and Snapchat, but somewhat blurs the line between semi-successful apps with hundreds of thousands of installs and mega-popular apps with installs in the hundreds of millions. Furthermore, the calculation of success within this paper would determine an app successful if it had over 50000 installs, a requirement that, while relevant in 2014, is outdated in more recent times, as app downloads have increased by 285% from 2015 to 2023 (4). The paper also utilized a linear regression for determining average app ratings but had limited predictive power.

This paper aims to provide a clearer way for developers to predict the rating of their app, as well as highlight critical areas to prioritize to maximize rating. These goals can be achieved through classification models, which are models that quintile each app into discrete rating labels, ranging from 1 to 5 given their attributes. After these classification models are trained, they can be used to predict the rating of new, unseen apps, or can return the relative weights of each attribute in determining rating. Since these models would be used by developers as a general predictor of the rating of their app, a continuous rating result that a regression model would return is unnecessary. Furthermore, although a neural net may prove more accurate than classification, we wished to retain the interpretability of our input classes (5). Instead, we used the Random Forest (RF), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) classifiers (6-8). These models are easily implemented and relatively intuitive, requiring less knowledge than other models, along with returning accurate predictions (9). For this paper, the KNN classifier predicted the rating of a data point with its "k"-nearest data points within the feature space, "k" being a hyperparameter; the SVM classifier used a hyperplane to separate different ratings, maximizing the distance between support vectors; and our last model, the RF classifier, created and combined decision trees using subsets of our data to make a generalized prediction of rating. These three individual models were used in the hopes that comparing them would lead to higher final accuracy, as each model has a unique architecture and methodology. We found that review count, days since the last update of the app, and storage size were the most influential attributes. Using this information, app developers can emphasize these key factors, using their estimated rating to locate areas of improvement for their apps (10).

https://doi.org/10.59720/23-033

RESULTS

We aimed to provide developers pathways to improve their Google Play Store apps. This was done through predicting the rating of Google Play Store apps using their attributes and classificatiton models. The data set used in this paper was taken from Kaggle.com and scraped from the Google Play Store in 2018 (11). The data set contains the information of 10,841 apps, consisting of a variety of attributes (**Table 1**).

We first plotted the distribution of the rating attribute within our data set to obtain some preliminary insight (**Figure 1**). We found that the attribute followed a left-skewed distribution, with a mean of 4.192, a median of 4.3, a mode of 4.4, and a standard deviation of 0.515. The distribution suggested a typical real-world scenario, further supported by the fact that the dataset scraped all apps from the Google Play Store, rather than a subset. Before using models to find the attributes that affect rating the most, we also used a pairplot, which visualizes the relationships between each attribute. We omitted attributes that would require one-hot encoding (a process that converts non-ordinal categorical variables to a 2-D binary array), namely Category and Content Rating, as they contain large 2-D binary arrays unsuitable for plotting. The kernel density estimation graphs show the spread and correlation coefficients of each relationship (**Figure 2**). Most graphs between various attributes and rating grouped around a rating of ~4.2, pointing to the mean rating of our dataset. These graphs also extended along the vertical line at ratings ≈ 4.2, explaining some of the low correlation between each attribute and rating – since there are multiple attribute values with respect to a rating of ~4.2, they have low predictive power. This can be seen in each comparison's correlation coefficient, where Reviews, Last Updated, and Name Length had the largest coefficients of only 0.2098, -0.1428 (a negative coefficient points towards an inverse relationship), and 0.1383.

However, this pairplot, while useful for finding initial trends between two attributes, failed to capture the interactions between all attributes for each app. This was instead done by training three classifiers—testing our three classifiers in a validation set 20% the size of our training set, we found that our KNN model had an accuracy of 0.767, the SVM had an accuracy of 0.768, and the RF had an accuracy of 0.793. To further explore the accuracy of our most accurate and most bias-resistant model, the random forest classifier, we analyzed its classification metrics, showing the high precision, but low recall, of ratings 1 to 3, meaning the model had accurate predictions for these ratings, but failed to identify most positive instances **(Figure 3)**. To assess the overall effectiveness of the RF model's predictions for these ratings, we measured its F1 score—the harmonic mean of precision and recall—which was low for these ratings. However, the high/moderate precision and recall of ratings 4 and 5 resulted in a high/moderate F1 score. These results, in summary, indicate that our random forest classifier was effective in predicting many apps with a rating of 4, most apps with a rating of 5, and some apps with a rating between 1 and 3. These results may be partially due to the bias towards higher ratings, further confirmed by the distribution of the model's test data (**Figure 4**). This could be a factor of the contrasting F1 scores between ratings 1-3 and ratings 4-5, the latter having a higher number of data points and training data, resulting in a higher F1 score.

After analyzing the accuracy of our RF, we found the importance of each input class in determining rating, to increase our model's interpretability. We found the Gini-impurity-based feature importances within our RF classifier—the more important an attribute, the more it would decrease the impurity, or how often a random data point would be incorrectly classified. Each attribute's importance scores are then averaged and normalized, summing to 1 (or 100%), allowing us to interpret the relative impact of each attribute within the model. These showed that most individual attributes had little importance in determining the classification of an app (**Figure 5**). Individual values of Category and Content Rating had limited effects on the model's output, consisting of 33 and 6 unique values, respectively, and having weights of less than 0.015. The Reviews attribute had the highest effect on the random forest model's predictions of rating, having 0.06 greater weight than the second-most significant attribute, Category. The attributes Last Updated (days since last update), Size, App (length of app name), Installs, and Android Ver all had similar weights of 0.112 0.031. However, Price had a lower weight of 0.017, indicating the lowest effect on the model's predictions.

DISCUSSION

We hypothesized that the rating of an app can be estimated with its attributes, which provides a clearer way for developers to predict the rating of their app, as well as highlights critical areas to prioritize during app development to maximize rating. We used three classification models, with our RF model being the most accurate with an accuracy of 0.793,

Feature	Description
App	The name of the app.
Category	The general theme of the app, including communication, entertainment, and fitness.
Rating	The rating of the app, in stars, from 1 to 5 – this is the feature our models try to predict.
Reviews	The number of reviews of the app.
Size	The space taken up by the app.
Installs	The number of total installs of the app.
Type	Whether the app was free or cost money.
Price	The initial cost of the app, where free apps have a price of 0 (not including in-app purchases or subscriptions).
Content Rating	A maturity rating that suggests an appropriate age to use the app.
Genres	The specific theme of the app, more granular than Category.
Last Updated	The date of the last update of the app.
Current Ver	The developer-specified version of the app.
Android Ver	The minimum Android version to use the app or a range of compatible versions.

Table 1: App features and their descriptions. Ver = Version.

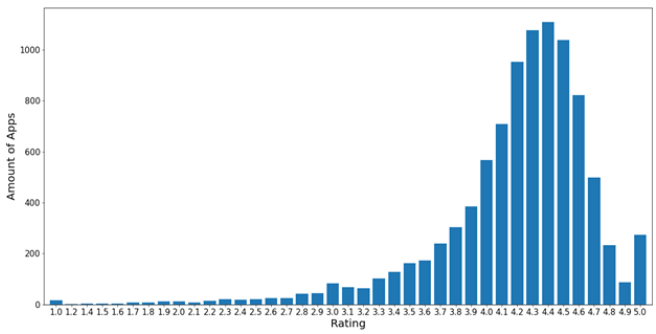

Figure 1: Rating frequency distribution. Occurrences of certain ratings with a total of 9366 apps. Our data was collected from a Kaggle data set on Google App Store apps. rating follows a left-skewed distribution with a mean of 4.192, a median of 4.3, a mode of 4.4, and a standard deviation of 0.515.

https://doi.org/10.59720/23-033

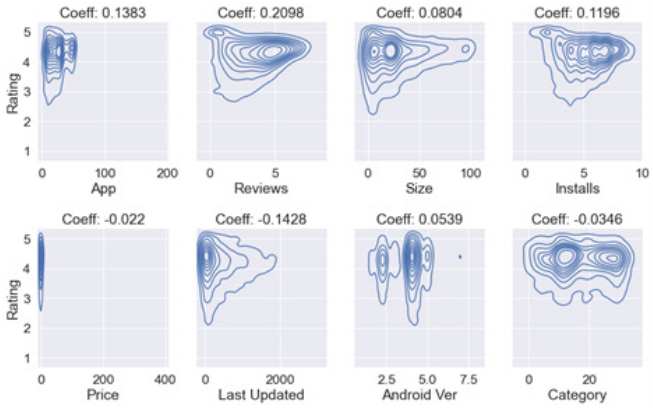

Figure 2: Pair plot of all attributes including respective correlation coefficients. Comparisons between each app attribute (omitting variables cleaned using one-hot encoding) shown as kernel density estimate plots (N=8). Individual relationships show weak linear correlations (which are calculated using the Pearson correlation coefficient) between different pairings, the magnitudes of which lay between ~0.02 and 0.21.

possibly due to its higher tolerance to biased datasets than SVM or KNN models. The model had high/moderate precision but low recall for ratings 1-3, and high precision and high recall for ratings 4-5. We also determined that Reviews, Last Update, and Size were the most important attributes. Through these results, we concluded that the rating of an app can be estimated through its attributes and that developers should focus on boosting Reviews, having more frequent updates, and reducing app size to maximize rating. Many have conducted similar experiments, and from the same data set we used, there have been multiple projects on graphing trends and similarities (12, 13). Rather than only analyzing the relationships between two attributes of an app, or comparing one attribute to rating, however, we aimed to provide a holistic view of the effects of all concurrent attributes on rating through training classification models.

Through our research, we determined that the RF classifier had a higher accuracy than the KNN and SVM classifiers. Having an overall accuracy of 0.793, this model was generally successful in predicting higher ratings, though struggled with the recall of lower ratings. Although Random Forest mo-

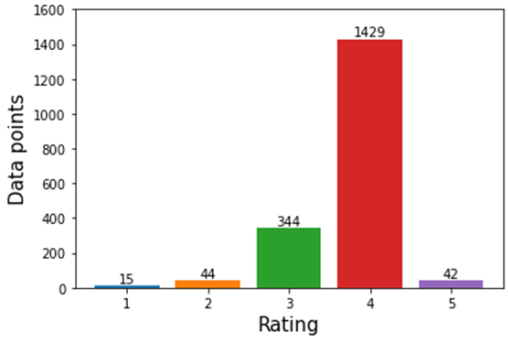

Figure 4: Random forest rating supports. Occurrences of each rating within the random forest's testing data (data points=1874), with each rating as a different color (N=5). Supports have similar distributions to training data (data points=7492) as well as total app distribution (data points=9366).

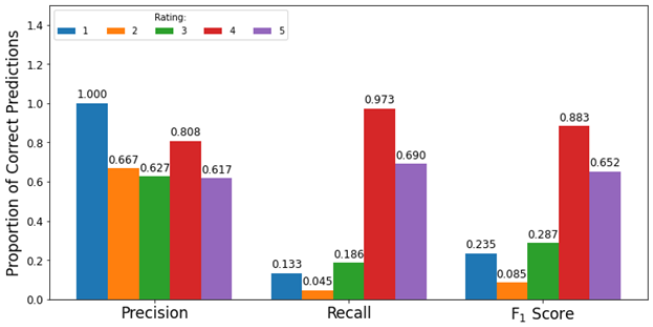

Figure 3: Random forest classification metrics. Random forest model's classification metrics (precision, recall, and F1 score) for each rating (N=5). Precision shows how often the model correctly rates an app (proportion of true positives to true and false positives), whereas recall shows how often the model correctly identifies rating out of all similar ratings (proportion of true positives to true positives and false negatives). F1 score is a harmonic mean of these two metrics.

dels have a high tolerance for biased datasets in comparison to Support Vector Machines and K-Nearest Neighbors, the small number of lower-rated apps may have been too small for adequate training. rating

The main limiting factor of the study was data—the dataset we used was heavily skewed towards apps with a 4+ rating, making our models lean more towards predicting those apps than equally representing all ratings A more suitable dataset would, ideally, include a more balanced pool of data points, as well as attributes specifying any bugs, load-in times, and popups for a more granular look into in-app variables. Furthermore, more precise data on points like installs would be beneficial. However, this dataset contains categories relevant to other studies—further analysis of the data could predict several variables, such as the optimal name for an app in a certain genre and a comparison of the size of an app and its install rate. Taken together, we conclude that the attributes of an app, mainly Price, Installs, and Genre, allow for the pre-

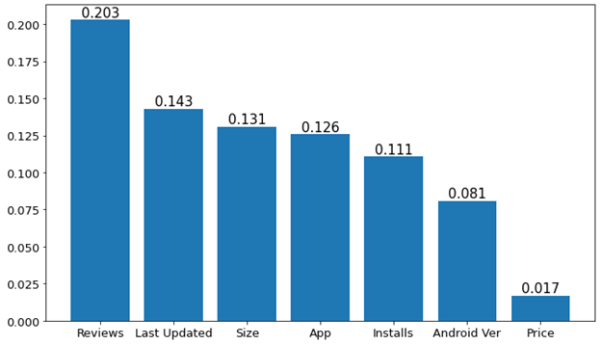

Figure 5: Random forest classifier's relative attribute weights. Relative importance of each attribute (N=7). with the categorical attributes Category and Content Rating omitted (attribute importance<0.015). The categorical attributes Category and Content Rating were omitted, as each category and content rating (33 and 6, respectively) had an attribute importance < 0.015. These feature importances were found using Gini impurity—the more important an attribute, the more it would decrease the impurity, or how often a random data point would be incorrectly classified. They are then averaged and normalized, summing to 1 (or 100%), allowing for relativistic interpretation.

https://doi.org/10.59720/23-033

diction of the rating of an app. The results of this paper can impact the decisions of app developers looking to optimize and predict the rating of their app—from what attributes to prioritize their development, to predicting the trajectory of their app, our paper can provide insights into the creation of a popular Google Play Store app.

MATERIALS AND METHODS

Data Exploration and Cleaning

The data set used in this paper was taken from Kaggle.com and was scraped from the Google Play Store in 2018 and contains the info for 10,841 apps within 14 attributes (**Table 1**) (11). After cleaning, every attribute except for Type, Genre, and Current Ver were used. Type, while able to be converted into numerical values, is less accurate than Price. Although Genre is more specific than Category, we felt that Category would be more applicable to developers, as apps compete in broader, app-wide categories rather than niche sub-genres. We also did not include Current Ver since it is determined by the app developer and is entirely subjective.

We began cleaning the data by removing all apps with NaN ratings, as well as removing improperly formatted data points. For Size, 17% of all values were listed as "Varies with device" —instead of removing a significant part of our data set, we opted to convert them to the average size of the other apps to prevent additional bias. For App name, we decided to convert it to "name length", as wordier names are less catchy and harder to type into search bars, reducing the visibility of the app. For Category and Content Rating, we used one-hot encoding to convert their names to numerical IDs, each a 2-D integer array that prevents natural ordering (1, 2, 3, …) which could result in falsely perceived correlations. For Last Updated, we converted the attribute to Days Since Final Update, with 0 equal to the latest update within the dataset, increasing by 1 for each day earlier. Finally, we converted Android Ver using a similar process to cleaning Size—14% of the values were listed as "Varies with device", so they were converted to the median version, as Android Ver contains discrete values rather than Size's continuous values.

Exploratory Data Analysis

All analyses were done in Python version 3.9.12 using the pandas, Matplotlib, Seaborn, and Scikit-learn packages (14-18).

To analyze the distribution of rating, we used the Matplotlib interface Pyplot to graph ratings against their occurrences in a bar graph (**Figure 1**). Then, to find the relation between ratings and other app attributes, we used a pairplot from the Seaborn library. Attributes cleaned using one-hot encoding, namely Category and Content Rating, were not included (**Figure 2**).

Model Fitting

After concatenating all our usable attributes, we used a MinMaxScalar from Scikit-learn to standardize the data into a range of 0 to 1. Afterward, we split the data into an 80-20 training and testing set for our models. Because our classifiers are predicting rating, we decided to fit the attribute into 5 buckets through flooring. We chose to floor rating due to two reasons—firstly, app developers hoping to meet rating goals would benefit from a more pessimistic outlook. Secondly, we believe that a perfect 5 rating carries more credibility than a 4.5 or 4.6, so we wanted to preserve that distinction.

Before training our models, we first performed grid searches to find the optimal model parameters. Starting with our KNN model, we found the optimal amount of "k" neighbors to be 50. The grid search for our SVM and RF models confirmed their default parameters, but due to the random forest model's resistance to overfitting, we set its "n_estimators" parameter to an arbitrarily high value of 1000. After fitting the training data, the models had the following accuracies: Random Forest—0.793, Support Vector Machine—0.768, and K-Nearest Neighbors—0.767. Instead of pursuing an analysis of each model, we decided that sufficient knowledge would be gained from an analysis of just our highest performing model, the random forest classifier, as it better handles overfitting and biased datasets than the other two classifiers.

Received: February 11, 2023
Accepted: September 13, 2023
Published: July 3, 2024

REFERENCES

1. "How Google Play Works." *Google Play*, 2022. play.google/howplayworks
2. Bhandari, Aparajita and Sara Bimo. "Why's Everyone on TikTok Now? The Algorithmized Self and the Future of Self-Making on Social Media." *Social Media + Society*, 22 Mar. 2022. https://doi.org/10.1177/20563051221086241
3. Tuckerman, C.J. "Predicting mobile application success." 2014. cs229.stanford.edu/proj2014/Cameron%20Tuckerman,%20Predicting%20Mobile%20Application%20Success.pdf
4. Iqbal, Mansoor. "App Download Data (2023)." *Business of Apps*, 2 May 2023. www.businessofapps.com/data/app-statistics
5. Blouin, Lou. "AI's mysterious 'black box' problem, explained." *University of Michigan*-Dearborn, 6 Mar. 2023. umdearborn.edu/news/ais-mysterious-black-box-problem-explained
6. Breiman, L. "Random Forests." *Machine Learning*, vol. 45, Oct. 2001, pp. 5-32. https://doi.org/10.1023/A:1010933404324
7. Cortes, C. and Vapnik, V. "Support-Vector Networks." *Machine Learning*, vol. 20(3), pp. 273–297. https://doi.org/10.1007/BF00994018
8. Peterson, Leif. "K-Nearest Neighbor." *Scholarpedia*, 2009. https://doi.org/10.4249/scholarpedia.1883
9. Boateng, Ernest, et al. "Basic Tenets of Classification Algorithms K-Nearest Neighbor, Support Vector Machine, Random Forest, and Neural Network: A Review." *Journal of Data Analysis and Information Processing*, vol. 8, pp. 341-357. https://doi.org/10.4236/jdaip.2020.84020
10. Pinheiro, Mariana, et al. "Predictors of the Number of Installs in Psychiatry Smartphone Apps: Systematic Search on App Stores and Content Analysis." *JMIR Publications*, vol. 6, no. 11, Nov. 2019. https://doi.org/10.2196/15064
11. Gupta, Lavanya. "Google Play Store Apps." 2018. *Kaggle*. www.kaggle.com/datasets/lava18/google-play-store-apps
12. "How to get "High" rating on Play Store." *Kaggle*, 29 Sep. 2018. www.kaggle.com/code/tanetboss/how-to-get-high-rating-on-play-store
13. Gupta, Lavanya. "All that you need to know about the Android market." *Kaggle*, 18 Sep. 2018. www.kaggle.

https://doi.org/10.59720/23-033

com/code/lava18/all-that-you-need-to-know-about-the-android-market
14. Rossum, Guido, et al. "Python Reference Manual." *Association for Computing Machinery*, 1995. dl.acm.org/doi/abs/10.5555/869369
15. (15) McKinney, Wes. "Data Structures for Statistical Computing in Python." *Proceedings of the 9th Python in Science Conference*, vol. 445, no. 1, 1 Nov. 2010. conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf
16. Hunter, John. "Matplotlib: A 2D Graphics Environment", *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, May/Jun. 2007. https://doi.org/10.1109/MCSE.2007.55
17. Waskom, Michael. "Seaborn: Statistical Data Visualization." *Journal of Open Source Software*, 6 Apr. 2021. https://doi.org/10.21105/joss.03021
18. Pedregosa, Fabian, et al. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, vol. 12, Nov. 2011, pp. 2825-2830. dl.acm.org/doi/10.5555/1953048.2078195