Rating Prediction for Google Play Store Apps

Abduokhapova Ayaulym, Yugai Sofya, Utemuratova Zhaniya

In this project, we analyze a dataset from the Google Play Store. First, we explore the characteristics of the dataset and then focus on the app success prediction problem. Our goal is to leverage the rich metadata available for each app to predict its success level, measured by the number of installs. To frame this as a classification task, we convert the *Installs* feature into four discrete categories: *Not Successful*, *Average*, *Above Average*, and *Successful*. We aim to predict these categories using features such as app category, number of reviews, size, price, type (free or paid), and content rating. We evaluate multiple machine learning models to identify which approaches best capture the relationships between features and install-based success.

Our work demonstrates that app metadata alone — without user reviews — can be effectively used to predict success. This has important implications for developers, marketers, and app store platforms seeking to assess or improve the potential reach of an app.

Dataset characteristics

The dataset used in this project is derived from the Google Play Store and contains metadata of mobile applications. The primary goal is to predict app ratings based on various features. These informations are captured in this file: googleplaystore.csv.

- **googleplaystore.cvs:** app, category, rating, reviews, size, installs, type, price, content rating, genres, lastupdated, current ver, android ver

The dataset required extensive preprocessing to ensure consistency and usability in our models:

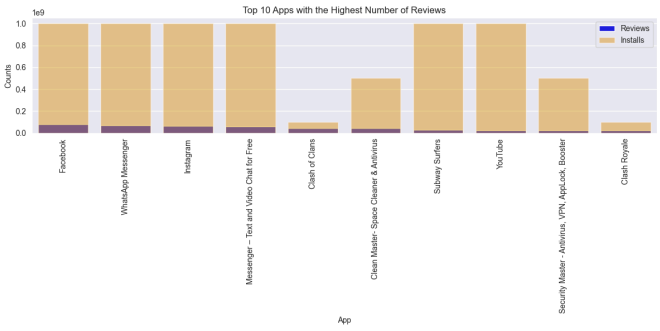
- **Missing Values**: Entries with missing values in critical fields such as Rating, Type, or Content Rating were removed.
- **Data Cleaning**: Fields such as Size, Price, and Installs included symbols like “M”, “+”, “$”, or strings such as “Varies with device”. These were cleaned and converted into numerical formats.
- **Feature Encoding**: Categorical features like Category, Genres, and Content Rating were label-encoded to numerical values.
- **Feature Scaling**: Since features like Reviews, Price, and Size had vastly different ranges, we applied either StandardScaler or MinMaxScaler for normalization.

After cleaning, the final dataset consisted of several thousand high-quality entries suitable for supervised learning. Notably, the **Installs** feature—used to determine app success—was discretized into four ordinal categories for classification, enabling us to model app success as a multiclass classification problem.

The original dataset as shown in Figure 1 spans across different categories and users.

Table 1: *Top 10 Apps in Google Play Store Apps*

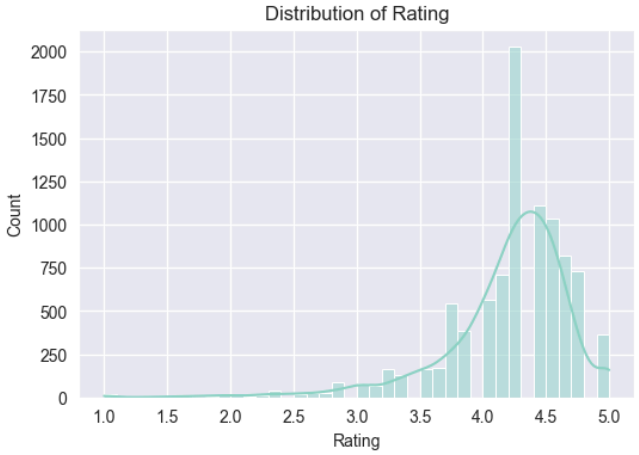
Apps
Subway Surfers
Facebook
WhatsApp Messenger
Instagram
Messenger - Text and Video Chat for free
Clash of Clans
Clean Master - Space cleaner & Antivirus
YouTube
Security Master - Antivirus, VPN, AppLock, Booster
Clash Royale



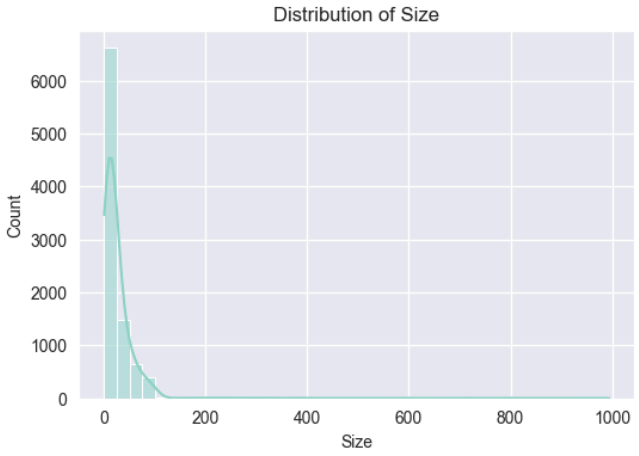
Here are the **top 10 applications** in the dataset based on the combined metrics of **number of reviews** and **installation counts**, representing the most popular and widely adopted apps on the Google Play Store. These apps typically belong to categories such as *Communication*, *Social*, and *Productivity*, and serve as strong positive examples in the "Successful" class of our target variable.

Understanding the distribution of key numerical features is crucial for effective preprocessing and model selection. In this section, we examine the distribution of several important attributes: **Rating, Size, Reviews,** and **Price.**

The **Rating** variable, which represents the average user rating for each app, is concentrated between 3.5 and 4.5. The distribution is left-skewed, indicating that most apps in the dataset receive relatively high user ratings. Very few apps have ratings below 3.0, which may reflect app store filtering or the general tendency of users to rate favorably.

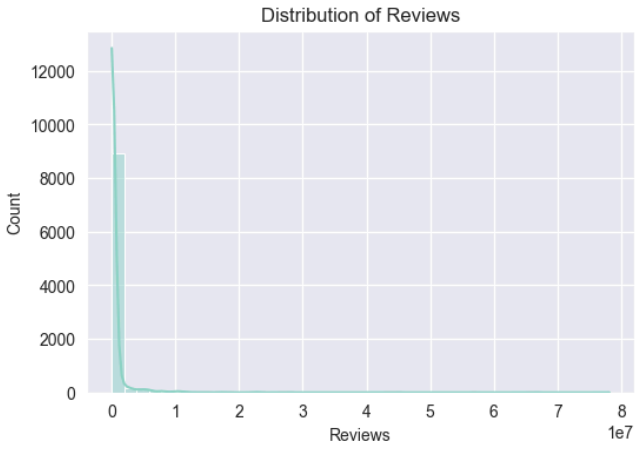


The **Size** feature, given in megabytes (MB), shows a **right-skewed distribution**. Most apps are relatively small in size (below 30MB), while a small number of resource-intensive apps (such as games or productivity tools) exceed 100MB. Some entries originally included "Varies with device", which were removed or treated as missing during cleaning.



The **Reviews** feature is **heavily skewed** with a long tail. While a large number of apps have fewer than 10,000 reviews, a small group of extremely popular apps has over a million reviews. Due to this extreme imbalance, we applied a **log** this extreme imbalance, we applied a **log**

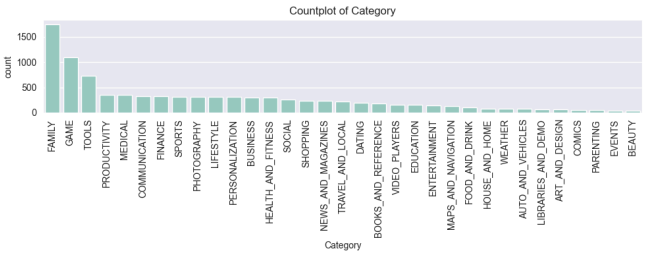
transformation to stabilize variance and reduce the impact of outliers.



The **Price** feature is dominated by free apps, resulting in a highly skewed distribution with a sharp peak at 0. A small subset of apps are paid, ranging in price from $0.99 to over $300. Since the range is large and sparse, we also applied a log transformation (after removing free apps) to normalize this feature.

To better understand the categorical composition of the dataset, we visualized the frequency of each app **Category** using a count plot (see Figure X). The results reveal that the dataset is **dominated by a few high-frequency categories**, such as *FAMILY*, *GAME*, and *TOOLS*, while many other categories like *BEAUTY*, *EVENTS* and *PARENTING* are significantly underrepresented(Fig 1.)
intermediate classes (e.g., misclassifying

Fig 1.



This imbalance is important to consider, as models might become biased toward these overrepresented categories during training. It also reflects real-world trends, where certain app types—especially entertainment and utility-based apps—tend to flood the market due to high user demand.

Methodology

At the early stages of our project, we explored linear and polynomial regression models as potential approaches to predict app success based on install counts. Regression models are widely used in predictive modeling tasks where the goal is to estimate a continuous target variable from a set of input features. Given that the original **Installs** field appears numerical at first glance (e.g., “1,000+”, “10,000+”, etc.), it seemed natural to consider a regression-based solution. The initial motivation to apply linear regression stemmed from its interpretability and simplicity.
We hypothesized that metadata such as **Rating**, **Reviews**, **Size**, and **Price** could linearly influence the install count. To allow for more flexible modeling of non-linear relationships, we also considered polynomial regression, which adds higher-degree terms of the input variables, thus capturing more complex interactions between features.
As a result, when we applied linear regression, the model produced poor fit (low R^2 score and high RMSE), and residual plots showed strong evidence of model misfit. Polynomial regression, while more flexible, led to **overfitting** due to the limited number of distinct install values, and further exacerbated the prediction instability for

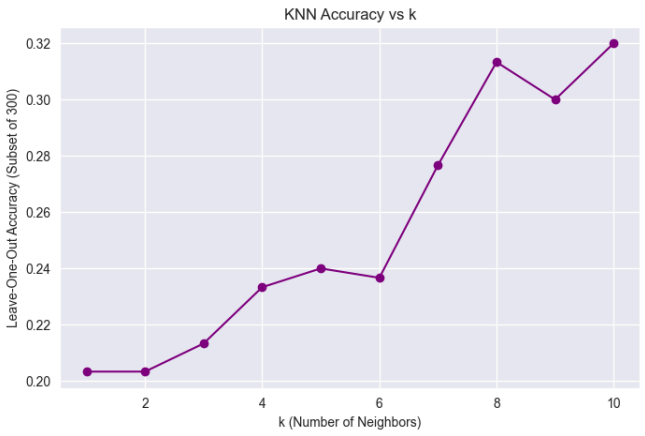
"100,000" as "1,000,000").

We trained and evaluated multiple machine learning models, including:

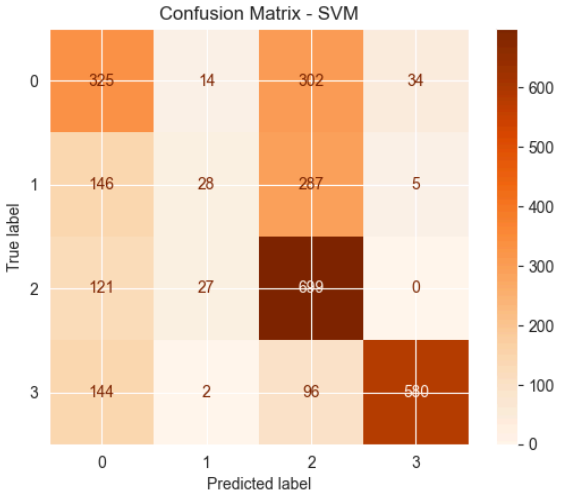
- K-Nearest Neighbors (KNN)
- Support Vector Machine (SVM)
- Decision Tree Classifier
- Random Forest Classifier

Each of these models has distinct strengths and weaknesses, making them valuable to evaluate within the context of our structured, metadata-driven dataset.

The **KNN classifier** is a non-parametric algorithm that classifies data points based on the majority class among their *k* nearest neighbors in feature space. We chose KNN initially for its simplicity and intuitive interpretation, especially useful during baseline model development.

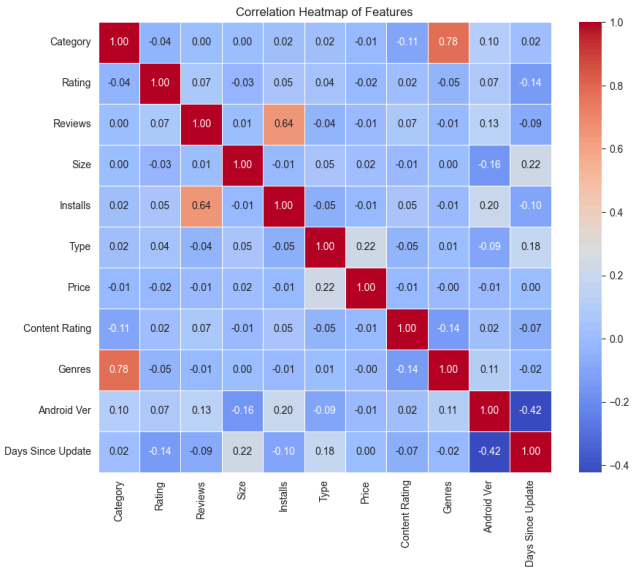


To improve the efficiency and performance of the Support Vector Machine (SVM) classifier, we applied **Principal Component Analysis (PCA)** for dimensionality reduction. PCA is a technique that transforms the original feature space into a set of orthogonal components, ranked by the amount of variance they capture from the data. By reducing the number of features, PCA not only speeds up training for computationally intensive models like SVM but also helps mitigate the effects of noise and multicollinearity in the input data.



To explore the relationships between numerical features in the dataset, we generated a **correlation heatmap** (see Figure X). A heatmap is a graphical representation of the correlation matrix, where each cell indicates the **Pearson correlation coefficient** between a pair of features. This visualization helps identify patterns such as:

- **Strongly correlated features**, which may indicate redundancy (e.g., Reviews and Installs)
- **Weak or no correlation**, which suggests feature independence
- **Unexpected inverse relationships**, which may require further investigation

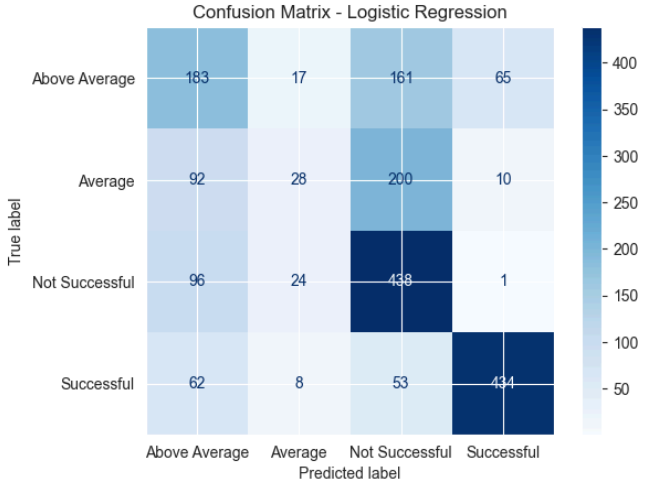


The heatmap also helped guide feature selection and avoid multicollinearity in models such as linear regression and SVM. By identifying highly correlated pairs, we ensured that features included in the model were informative but not redundant.

Techniques

Logistic Regression

Logistic Regression is a statistical method for binary and multiclass classification problems. Unlike linear regression which predicts continuous values, logistic regression predicts the probability that an instance belongs to a certain class.



For our dataset, Logistic Regression was used to classify apps into one of four success levels based on features like number of installs, size, rating, price, and content type. The model learns a set of weights for each feature and applies the logistic (sigmoid) function to estimate class probabilities.

Advantages:

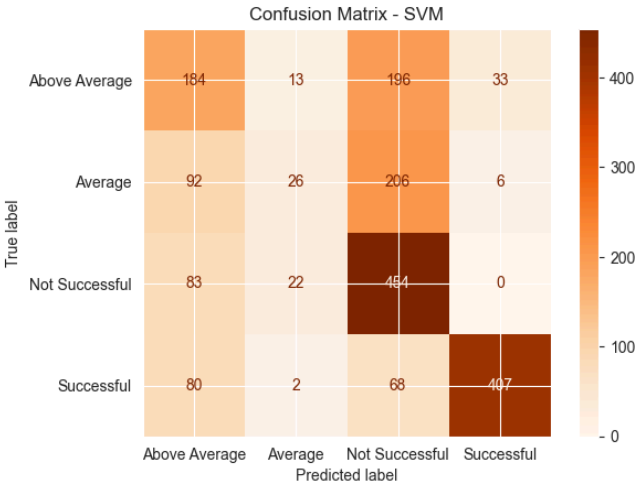
- Simple to implement and interpret.
- Works well when the relationship between features and output is approximately linear
- Computationally efficient and suitable for smaller datasets.

Disadvantages:

- Assumes linear separability between classes, which may not hold in more complex datasets.
- Sensitive to outliers and multicollinearity.
- May not capture interactions between features unless explicitly added.

Support Vector Machine (SVM)

SVM is a powerful and flexible classification algorithm that aims to find the optimal decision boundary (or hyperplane) between classes. It can handle both linear and non-linear problems using different kernels (e.g., linear, polynomial, RBF).



In our case, SVM was used to classify apps based on the same feature set. The key strength of SVM lies in its ability to maximize the margin between classes, which often leads to better generalization, especially in high-dimensional spaces.

Advantages:

- Effective in high-dimensional feature spaces.
- Works well with clear margins of separation.

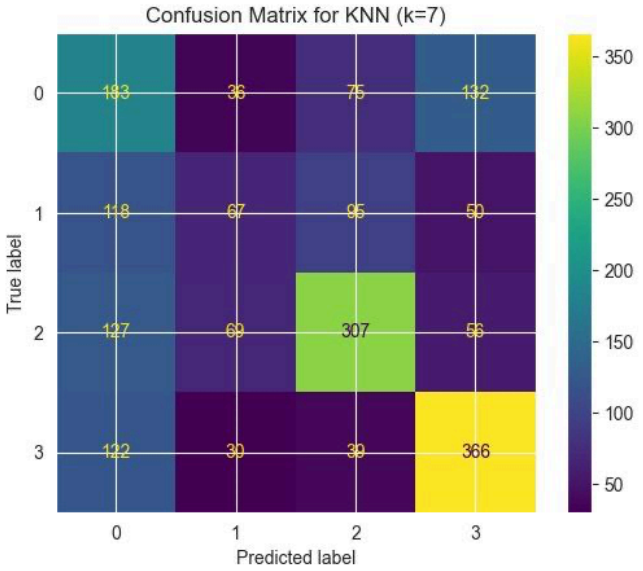
- Robust against overfitting, especially with proper regularization.

Disadvantages:

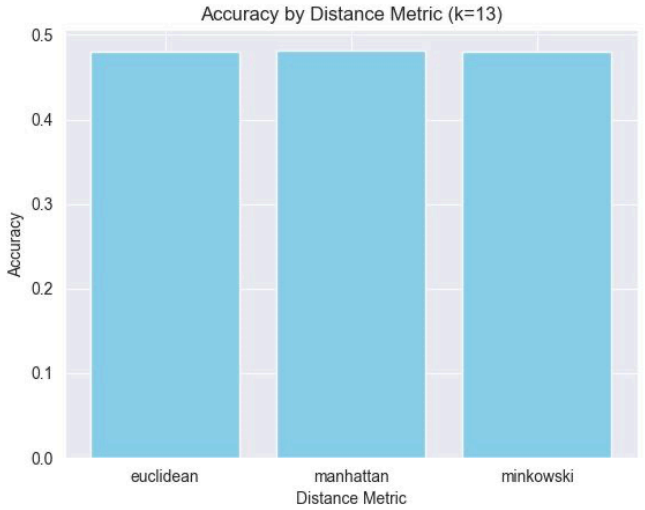
- Can be computationally intensive for large datasets.
- Choosing the right kernel and tuning hyperparameters can be complex.
- Does not directly output probabilities (although this can be approximated).

Both models are suitable for classification tasks like the one in our dataset. Logistic Regression provides a good baseline with interpretability, while SVM often performs better when the decision boundaries are not strictly linear.

To further evaluate the performance of the K-Nearest Neighbors (KNN) classifier, we analyzed its **confusion matrix**, which provides a detailed view of the model's classification results across the four success categories.



In the K-Nearest Neighbors (KNN) algorithm, the choice of distance metric plays a critical role in determining how neighbors are selected. We evaluated KNN performance using different distance measures, including **Euclidean**, **Manhattan**, and **Minkowski** distances. Each of these metrics calculates the distance between feature vectors differently, which can impact classification accuracy, especially in high-dimensional or scaled data.



Challenges

One of the major challenges we encountered was during the data cleaning phase. Due to missing or improperly formatted values in critical columns such as Installs, Rating, and Size, we were forced to discard approximately 2,000 entries from the original dataset. While this step was necessary to ensure the quality and consistency of the remaining data, it also reduced the dataset size and limited the diversity of app types available for training. This loss of data may have affected the balance across target classes, particularly for the "Successful" category, which already had fewer examples.

Conclusion

This project demonstrates the feasibility of predicting app success on the Google Play Store using only app metadata. Our model provides a scalable method for developers and analysts to estimate app performance without relying on user reviews or post-launch metrics. Key insights: Success levels correlate strongly with app type, reviews, and pricing model Feature engineering and preprocessing significantly improve model performance Ensemble models like Random Forest yield strong, interpretable results Future work may include incorporating textual features from user reviews, app descriptions, or temporal dynamics such as version update frequency.