BACHELOR OF SCIENCE IN
COMPUTER SCIENCE AND ENGINEERING

Inspiring Excellence

Exploratory Data Analysis and Success Prediction of Google Play Store Apps

Authors
Abdul Mueez
Khushba Ahmed
Tuba Islam
Waqqas Iqbal

Supervisor
Amitabha Chakrabarty, Ph.D
Associate Professor
Department of CSE

A thesis submitted to the Department of CSE
in partial fulfillment of the requirements for the degree of
B.Sc. Engineering in CSE

Department of Computer Science and Engineering
BRAC University, Dhaka - 1212, Bangladesh

December 2018

Declaration

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any Degree or Diploma.

Authors: Abdul Mueez, Khushba Ahmed, Tuba Islam, Waqqas Iqbal

Abdul Mueez
Student ID: 15101108

Khushba Ahmed
Student ID: 15101020

Tuba Islam
Student ID: 15141002

Waqqas Iqbal
Student ID: 15101109

Supervisor: Amitabha Chakrabarty, Ph.D

Amitabha Chakrabarty, Ph.D
Associate Professor, Department of Computer Science and Engineering
BRAC University

December 2018

The thesis titled: Exploratory Data Analysis and Success Prediction of Google Play Store Apps
Submitted by:
Abdul Mueez Student ID: 15101108
Khushba Ahmed Student ID: 15101020
Tuba Islam Student ID: 15141002
Waqqas Iqbal Student ID: 15101109
of Academic Year ..... has been found as satisfactory and accepted as partial fulfillment of the requirement for the Degree of .....

1. Amitabha Chakrabarty, Ph.D.
Associate Professor
BRAC University

2. Md. Abdul Mottalib, Ph.D.
Chairperson
BRAC University

Acknowledgements

We would first like to express our sincere gratitude to our thesis advisor Associate Professor, Dr. Amitabha Chakrabarty of the School of Computer Science and Engineering at BRAC Univeristy. The door to Prof. Chakrabarty's office was always open whenever we ran into trouble spot or had a question about our research or writing. He consistently allowed this paper to be our own work, but steered us in the right the direction whenever he thought we needed it.

We would also like to express our cordial thanks and utmost gratitude to Dipankar Chaki, Lecturer of the School of Computer Science and Engineering at BRAC University for providing his valuable insight, suggestion and guidance to us.

Lastly, we want to convey gratitude from the depths of our hearts towards the faculty members of the department of Computer Science and Engineering for providing us with all the knowledge throughout our undergraduate life which has immensely helped us in conducting our work.

Abstract

Mobile app distribution platform such as Google play store gets flooded with several thousands of new apps everyday with many more thousands of developers working independently or in a team to make them successful. With immense competition from all over the globe, it is imperative for a developer to know whether he is proceeding in the right direction. Unlike making a movie where presence of popular celebrities raise the probability of success even before the movie is released, it is not the case with developing apps. Since most Play Store apps are free, the revenue model is quite unknown and unavailable as to how the in-app purchases, in-app adverts and subscriptions contribute to the success of an app. Thus, an app's success is usually determined by the number of installs and the user ratings that it has received over its lifetime rather than the revenue it generated. In this thesis, on a smaller scale, we have tried to perform exploratory data analysis to dive deeper into the Google Play Store data that we collected, discovering relationships with specific features such as how the number of words in an app name for instance, affect installs, in order to use them to find out which apps are more likely to succeed. Using these extracted features and the recent sentiment of users we have predicted the "success" of an app soon after it is launched into the Google Play Store.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

BS4	Beautiful Soup 4
CSV	Comma-Separated Values
EDA	Exploratory Data Analysis
GLM	Generalized linear model
HTML	Hypertext Markup Language
LDA	Linear Discriminant Analysis
NLTK	Natural Language Toolkit
PLC	Product Life Cycle
POS	Parts Of Speech
SVM	Support Vector Machine
URL	Uniform Resource Locator
XML	Extensible Markup Language

Chapter 1

Introduction

In this chapter, we briefly described our research work and gave a complete overview of our work including the thesis contribution and the problem statement. Later, other parts of this report contain more information about our findings and analysis.

1.1 Introduction

It has been observed that the significant growth of the mobile application market has a great impact on digital technology. Having said that, with the ever-growing mobile app market there is also a notable rise of mobile app developers; eventually resulting in sky-high revenue by the global mobile app industry. With immense competition from all over the globe, it is imperative for a developer to know that he is proceeding in the correct direction. To retain this revenue and their place in the market the app developers might have to find a way to stick into their current position. The Google Play Store is found to be the largest app market in the world. It has been observed that although it generates more than double the downloads than the Apple App Store but makes only half the money compared to the App Store. So, we scraped data from the Play Store to conduct our research on it. Recently, a good number of researches has been made on this data as we have seen in paper [3][19]. However, we came across very few papers like [14][6][17] where they tried to find out issues that cause the market to have lower revenue even after having a higher number of downloads in the overall market. Hence, we did exploratory data analysis to find out the reason behind these issues and also defined a success parameter for an app and created a model that predicts it, using an app's metadata.

In the very beginning, we collected the URLs of over 5000 apps and then we scraped the data of those URLs from Google Play Store. The dataset that we used had 5223 unique apps and contained over 170000 reviews. For the EDA part, we analyzed our dataset and

created several Bar Charts to visualize the relationship between each attribute. Then we created a heat map of installs with other numeric features from there we could conclude that numeric features such as the number of ratings as well as subjectivity are all uncorrelated with installs. Additionally, we did an analysis on the features App Name, Content Rating, and Type and reached some valid conclusion. Furthermore, we did review analysis on the 170000 reviews obtained from scraping, using those reviews we created a dictionary of words that appeared in the reviews and recorded their occurrences and illustrated that using a Bar Chart. From those Bar Charts, we observed the variance of the first 25 most frequently used word associated with each rating ranging from 1 through 5. To further investigate that, we created a dictionary of positive and negative words manually, using information from the dataset. However, the explanations for all the analysis performed are described in details in Chapter 4. From the EDA performed, we observed the importance of each feature and the correlations between each of them and tried defining our own success metric being inspired by the approach of predicting success in paper [23]. Hence, from the analysis, it can be seen that determining the success rate of an app will play a very important role for developers and can bring certain changes that might affect the lifetime of their app.

For the EDA part we used various Python Data visualization libraries and for predicting the success rate, we used two ensemble learning methods and two machine learning algorithms- XGBoost, Random Forest, KNN and SVM and got the highest accuracy using XGBoost. After comparing all the accuracies by using different algorithms, we performed voting ensemble to get a better accuracy than each individual methods could achieve alone.

1.2 Motivation

Mobile App Industry is booming in a significant way and thus this is giving rise to more competition among the developing community those who create applications. This is kind of a threat to each individual developer as a greater number of competitions will increase the chance of degrading the value of their app in the store. Therefore, even if an app is launched and has reached a standard rating along with a strong number of installs it is not enough to retain their position in the Play store forever because there will be many other apps which will be exactly providing the end-users with same features or even better. It's worth revisiting the case of Dong Nguyen, creator of the ultra-popular "Flappy Bird" game in 2013; although he reportedly earned as much as 50,000 a day from in-app advertising and purchases when the game became a smash hit, app stores were soon flooded with dozens of (often poorly-made) clones. Due to the expansion and competition among the market we chose to do research relevant to this sector so we did a thorough analysis of our data of Google Play Store and

came up with our own defined success metric which we think will be a big contribution for the developers as through this they will get to know their success rate and will be able to decide what feature needs to be maintained or which one needs to be modified according to the current state of their app. Our main motive for this research was to find something meaningful throughout the analysis that might matter to the developing community or to the end users. Many Researchers did research on topics like predicting store rating, sentiment rating, version rating, and many other analyses have been done previously using the data from Google Play Store, however, very few types of research have been conducted based on the topic we have chosen, therefore, we found this topic more convincing for our research work and proceeded with this.

1.3 Problem Statement

The expansion of smart phones is driving the fast development of mobile app stores. Currently, the two largest global platforms for app distribution are Apple's App Store (for iOS users) and Google play store (official app store for the Android OS). For our research, we have picked Google play store and did a thorough analysis of its features that were available to us for predicting the success of a particular app. But the question arises that, is it even necessary to do so? Well, the answer is yes, that is because an average of 6,140 mobile apps is released through the Google Play Store every day, according to the statistics shown by the Statista in their recent report where all the app in together compete for user attention [2]. And the number of available apps in the Google play store is estimated to be around 2.6 million applications in March 2018 subsequent to outperforming 1 million apps in July 2013 [1]. The huge number of apps in the play store and the numbers of the app released every day make it quite competitive for the app developers, the companies who are building an app to come up with a unique idea that will definitely be bought by the end users. Because at the end of the day if the app does not perform well in the android market, then all the hard work behind building the app will go in vain. As the mobile industry is growing rapidly it is increasing the level of competition however, increased competition also leads to increased chances of failure. So, the developers need to do enough research as an enormous amount of time, effort and the money are invested into the process, so business cannot afford an app failure. If we look into the history of the revenue growth of play store, then between 2016 and 2017, play store has seen a revenue growth of 34.2%, with the percentage rise in app downloads was 16.7% [20]. The app that is launched needs to generate revenue so that it can pay off the team of developers and help the company to make a profit and add money to its capital. Well the developer does not get all of the money an app makes, he

gets 70% of the total money and play store keeps 30 % when the app is launched in the play store, and there is no minimum threshold when it comes to the amount that an application is to earn, any amount above 1$ gets credited to the account. Play store works on a 70:30 payment distribution ratio [20] Building an app is therefore not an easy task to deal with, as there are a bunch of developers creating app every now and then. But to reach the level of success that what makes an app stand out in the crowd. According to the State of Mobile App Developers study around one-third of the app developers have less than 10k downloads across all regions, globally only 15% have crossed >1000k download mark [24]. The lower the number of downloads the less it has a chance that it will do great business ahead in future in the android market. This is one big problem that we tried to solve in our research. We looked into apps that failed miserably in the app business in spite of having great ideas [8]. For example, Hailo, a mobile app that is used to hail cabs from your phone, it uses the same concept as Uber but its work was confined within only yellow cabs. Even though it came with a great idea but failed to succeed because of intense competition and a flawed business model; Google Wave (supposed to be the ultimate communication tool) a very few people knew about it, and the app complexity was way too high for end users to grab the idea of how to use it, though the excitement was huge it did not meet the expectation [8]. Success is a thing that does not come to one so easily and for that, we analyzed the features of Google play store and came to a conclusion that will help developers to understand their app success rate using our proposed success parameters.

1.4 Thesis Contribution

Addressing the problems mentioned in the earlier section, we tried to figure out ways to help developers target the right people with the right apps. We performed Exploratory Data Analysis on the Google Play Store dataset that we formed by scraping the Play Store site. It was observed that apps of certain category have significantly more installs than others, and free apps dominate the android market. We further found that developers who use succinct names acquire higher installs than those who fail to do so. Moreover, we analysed more than 170,000 text reviews/comments of 5233 apps made by the users and learned how most people express satisfaction and dissatisfaction in their writing. All these findings can pave way for developers who are planning to enter into the android market and are willing to have a competitive advantage over others. In addition to analysing data, we formulated success of an app in terms of its installs and average rating, justifying our choices and decisions in the most logical way possible. We labelled 5233 apps as “successful” and “not-successful” based on our formulation. The features extracted from data analysis and the basic attributes

of apps were used to train four different classifiers to predict the earlier formulated success. For apps which have stagnant growth rate or are new to the market, our success model can help developers predict whether their app will reach success in the long run given the present state of the app.

1.5 Thesis Outline

Chapter 1: Contains a brief overview of our research, our estimated goal and how our work can help the developers.
Chapter 2: Discussion of the literature review and background study of our thesis, including detailed description of the algorithms we used in our research.
Chapter 3: Description of the process of collecting data, data processing and description of the corresponding features.
Chapter 4: Discoveries about the correlation between the features that we found by performing EDA with visual representations.
Chapter 5: Definition of the success parameter that we proposed as our target class and the results of each model used.
Chapter 6: Conclusion and proposed future work for the system.

Chapter 2

Background Study

In this chapter we present the work of other relevant papers and describe the algorithms that we used to create our models.

2.1 Literature Review

In paper [23], the author scraped data from the Google Play Store to extract as many features as possible using a web crawler 'scrapy' and used the data to train three models to predict the success of an application. To predict success metrics of an app, revenue should be the key feature but as it is not found publicly, the author used the number of installations and average user rating, trained Linear model to categorize whether an app will be successful or not and additionally used linear regression to predict the average rating of a system. The principal component analysis was performed in order to focus on variation and create strong patterns of the dataset by using inputs of GLM and Linear regression models. Using these models, the author concluded that about 35% of the total successful applications has the word 'photo' in the description and about 31% has the word 'share', using these models, the developer can be referred to the genres of application which are mostly liked by end users. For future work, the author suggested using the revenue to predict success metrics and referred companies like AppAnnie to collect such data. From the defined success metric, a developer will be able to find the economic success of an application. This paper acted as our reference paper, we got the idea of how the number of downloads and average rating can play an important role while defining success metric and to find the success rate of any application.

Furthermore, in paper [14], the authors observed that how store-ratings after reaching a certain value does not affect the overall store rating even after users rate it, they also noticed that when an app is updated their rating varies version wise, but it is not observable in the store-rating. Therefore, they came up with the idea of version rating which they could

calculate based on the calculation of store rating which was available in App Store. This idea was proposed in order to help developers gain incentive to develop apps even after the store rating reaches a threshold value and they recommended that every App store owner should display the current version rating. Their approach of calculating version rating to make developers work for the updates is very efficient.

Similarly, in paper [3], the authors tried to represent the review-rating mismatch, through establishing multiple systems that can automatically detect the inconsistency between these two, to prove this mismatch they implemented two machine learning approaches, it included different classifiers like Naive Bayes Classifier, Decision tree, Decision stump, Decision table, and few other algorithms, and the other approach focused on deep learning techniques. They also hosted several surveys in order to learn the opinion of end users and developers regarding this mismatch. The outcome of the survey was quite expected, both the Developer and end users of Android app agreed that rating of an app should match with its corresponding review and they also asserted to have an automated system to detect the mismatch between rating and review if there is any. This paper proposed a wonderful way to represent the review-rating mismatch, we got inspired by their work and gave an effort to create a feature based on the review and rating mismatch and eliminate those which has a higher difference in mismatch but due to time constraint three of our members could manually annotate only 2000 reviews out of 199763 individually, due to this huge variance we could not get a proper result and dropped the idea of doing so.

Another paper working on the same aspect [10], the author states that the numeric rating as in the stars given by users have a huge difference than compared to the reviews given by them thus a rating system has been proposed by the author which will remove the ambiguity created by the mismatch of the rating and respective review by the same user. It has been seen that how much we as users are dependent on others opinion while taking any decision so according to the author people install the app based on the rating that is given for that particular app. Here the author describes the problem dividing it into two sub-problems. Firstly, the ambiguity and secondly it is the biasness to the summarized rating of the users. Further explaining the problem, he added that earlier people used to only extract the rating from the comments rather including the star rating associated with it. To solve the problems, he proposed a system that will initially conduct sentiment analysis on the user reviews and will generate a numeric rating from the polarity. Thus, a final rating will be the average of the rating from the sentiment analysis and the star ratings that are given by the users. This proposal would reduce the confusion of the users and allow them to have a final rating based on both review and star rating. This paper shows a strong relationship between the star rating and the reviews of the users, which helped us of picking up the idea of using the reviews of

the app in our work and we did a thorough research on the reviews that are explained in later sections of this paper.

Correspondingly, in paper [14], Luiz et al. proposed a framework for mobile application developers with which they will be able to bring in modification on features those are found negative based on the end users' evaluation of their application. Their Framework consisted of three main building blocks (i) topic modeling, (ii) sentiment analysis and (iii) summarization interface. This Framework was designed to make developers realize that sentiment rating provides a more accurate value of user feedback for an application than star rating and how important it is to take account of the biasness of the features that affect the overall rating of an application. This paper guided us to find the Sentiment mean by showing how it gives a more accurate value than star rating. We simply used a python library for finding the Sentiment mean rather than using their strategy.

Considering the fact that how important polarity values are, thus from Fu, Lin, and Li work [6], it helped us to get the idea of removing the inconsistent reviews that will basically reduce noise from the dataset and give better performance in sentiment analysis, and therefore will generate polarity value more accurately. Fu, Lin, and Li proposed WisCom, a system that is able to analyze at least ten millions of user ratings and comments in the app markets in three different levels. The first feature of their system is to identify the Inconsistency in reviews; secondly, they looked for the reasons why people do not like a particular app and how the reviews change over time. They extended their analysis to provide a valuable insight into the complete app market providing users with major concerns. It generates techniques for summarizing and mining the reviews, which will help the end users to choose the best app that has the best experience without reading the comment, also app developers can then use the set of result to understand why end users don't like their app which will eventually help the developers to improve their quality of app. More specifically it analysis review on three levels that are firstly on single review (micro level) done using regularized linear regression model, secondly on reviews of each app (meso level) by doing LDA as well as they performed topic analysis on different time segments to judge how review changes over the time, and lastly on all of the apps in the market (macro level). They highlighted all their three-level techniques and described the benefits that will be received by the end-users, developers, and the entire mobile ecosystem.

Additionally, we skimmed through the book "Sentiment Analysis and Opinion Mining" [13], from there we gathered important information as we learned about the three different levels of sentiment analysis which consists of the document level, sentence level, and aspect and entity level and the problems associated with it and this book also enhanced our knowledge on how to deal with conditional and sarcastic sentences, we also got to know

how aspects can be extracted using Supervised Learning algorithms and how those extractions can be grouped into categories. Furthermore, we learned about the dictionary-based approach why it is formed how it can help people to find a large number of sentiment words with their orientations, we did some similar type analysis in our EDA in the Section of Review analysis [4.2]. This book basically introduced the field of sentiment analysis and opinion mining and surveyed the current state-of-the-art.

We also looked in papers that are of different domains but of similar work to understand better that how reviews vary from one domain to other so that we can justify the reasons of chossing each features in our dataset that played an important role of predicting the success using the success metrics proposed in [23]. In paper [17], Pang and Lee worked on movie reviews; they converted their extracted rating into one of the three categories: neutral, positive, negative. For the list of words, they asked two computer science graduate students to volunteer for choosing good indicator words for positive and negative sentiment in a review for the movie. From the given words they initially found accuracies for the overall negative and positive sentiment and later made a more compressed file of words using seven words for positive and negative respectively. They used three algorithms: Naïve Bayes, Support vector machine, and Maximum Entropy (MaxEnt or ME, for short), with all three algorithms they tried eight different feature combinations and shown the compared result, where SVM performed the best and Naïve Bayes the worst. They found that adding POS (Parts of speech) tags slightly increase the accuracy for Naïve Bayes, but declines for SVM and for MaxEnt it remains unchanged. Their work is quite different as that of ours, as movie review are generally different from app review [6]. For example, according to Fu et al. [6], movie reviews are generally longer, as compared to app review (average 71 characters per comment stated in [6]).

On the other hand, Jong [11] worked with yelp dataset consisting of one hundred fifty thousand reviews and their corresponding ratings for restaurants. According to his findings [8], he stated text review holds the more quantitative value that a star rating thus combining the star rating with a list of restaurant text review will provide a rich quantitative estimation of the service satisfaction rating. The average rating of his dataset was 3.6, so he placed all ratings above this to positive sentiment and below this to negative sentiment. Similar to the work of Pang and Lee he also used Naïve Bayes and SVM, but he used another different learning algorithm which is composed of two processes: capturing semantic similarities and modeling after sentiment. Using seven different training set sizes they predicted rating with respective sentiment analysis of twenty thousand unique reviews. In their research, the Naive Bayes classifier algorithm performed the best out of the three where in our research it is the opposite scenario.

All the papers mentioned above helped us to get all the ideas that we incorporated in our work, which made our work more organized and eventually helped us to produce a better analysis of the features that are present in our dataset.

2.2 Supervised Learning

2.2.1 What is Supervised Learning?

Supervised learning is the data mining task of inferring a function from labeled training data.The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances [21]. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

Supervised learning is divided two broad categories: classification and regression.

- In classification, the goal is to assign a class (or label) from a finite set of classes to an observation. That is, responses are categorical variables. Applications include spam filters, advertisement recommendation systems, and image and speech recognition. Predicting whether a patient will have a heart attack within a year is a classification problem, and the possible classes may be true and false.
- In regression, the goal is to predict a continuous measurement for an observation. that is, the responses variables are real numbers. Applications include forecasting crypto-currency prices, energy consumption, or disease incidence and many other.

Basic work-flow of Supervised Learning:

- Prepare Data
- Choose an algorithm
- Fit a Model
- Choose a validation method
- Examine Fit and Update Until Satisfied
- Use Fitted Model for Predictions

2.2.2 Algorithms

Random Forest

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method. Ensembling is nothing but a combination of weak learners (individual trees) to produce a strong learner.

Definition: A random forest is a classifier consisting of a collection of tree structured classifiers $h(x,\Theta_k), k = 1,...$ where the Θ_k are independent identically distributed $(i.i.d)$ random vectors and each tree casts a unit vote for the most popular class at input [4].

Random Forest Algorithm: The following are the basic steps involved in performing the random forest algorithm:

- Pick N random records from the dataset.
- Build a decision tree based on these N records.
- Choose the number of trees you want in your algorithm and repeat steps (i) and (ii).
- In case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

Figure 2.1 shows different trees labelling the class differently. What ensemble does is take the mode (maximum occurring class) of the output produced by n different trees to create a better model. To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

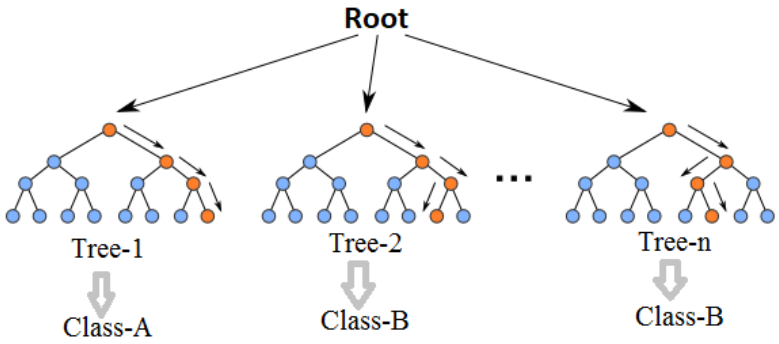

Fig. 2.1 Multiple decision trees [12]

Even though decision trees are pretty intuitive and easier to understand, they can be very noisy. Few changes in the data can lead to different splits and completely different models. The instability of the tree makes it unrealistic as a prediction model by itself. A single decision tree is insufficient and generally overfits the data, that is it can capture the structure of the in-sample data very well, but it tends to work poorly out-of-sample. In the context of statistics, decisions trees have low bias (as it can fit the data well) but high variances (the predictions are noisy).

Understanding the working principle of decision trees is imperative in the understanding of Random Forest Algorithm. The most popular algorithm for decision trees is ID3 algorithm. It finds the best attributes/features that best classifies the target attribute. One of the most commonly used way to figure out the best attribute is by calculating Information Gain which is, in turn, calculated using another property called Entropy.

The calculation of entropy of a system is done as follows:

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{2.1}$$

Here, c is the total number of classes or attributes and p_i is number of examples belonging to the i^{th} class. Information gain is simply the expected reduction in entropy caused by partitioning all our examples according to a given attribute. Mathematically, it is defined as:

$$Gain(S,A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.2}$$

S refers to the entire set of examples that we have. A is the attribute we want to partition or split. |S| is the number of examples and $|S_v|$ is the number of examples for the current value of attribute A. The attribute with the highest information gain sits at the root node, and the tree is first split based on that attribute.

XGBoost

XGBoost is another ensemble learning method. As it is almost never sufficient to reply upon the results of just one model, it combines the predictive powers of multiple learners to reach a conclusion. The base learners are weak learners in which the bias is high, and the predictive power is just slightly better than random guessing. But each of these weak learners add some vital information for prediction, resulting in a strong learner by effectively combining these weak learners. The final strong learner brings down both the bias and the variance.

The tree ensemble model consists of a set of classification and regression trees (CART). Figure 2.2 shows a simple example of a CART that classifies whether someone will like an

app or not. The original figure from [5] had been modified to paint a better picture of our dataset.

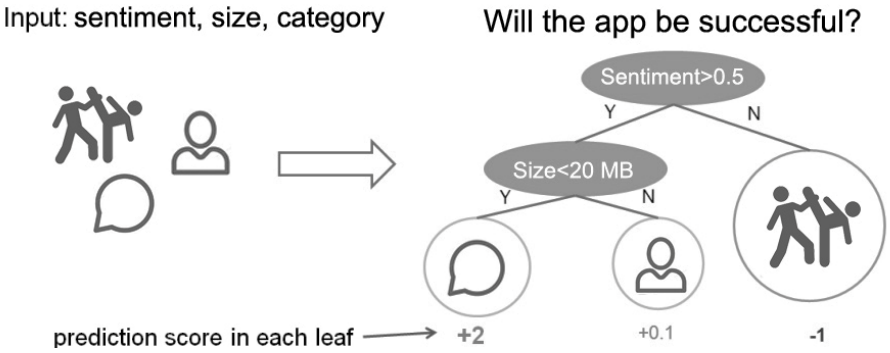

Fig. 2.2 CART Model Representation

Suppose, the many app categories available are classified into different leaves and assigned a score on the corresponding leaf. Unlike decision trees, in which the leaf only contains decision values, in CART, a real score is associated with each of the leaves, which gives a better interpretation.

The task of training the model involves finding the best parameters θ that best fit the training data x_i and labels y_i. This is done via the objective function which measures how well the model fits the training data. Objective functions are composed of two parts: training loss and regularization term which can be denoted by:

$$obj(\theta) = L(\theta) + \Omega(\theta) \tag{2.3}$$

where L is the training loss function, and Ω is the regularization term. The regularization term controls the complexity of the model, helping to avoid overfitting.

While trees are built in a parallel manner in bagging, boosting builds trees sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Figure 2.3 perfectly illustrates the concept. Due to each tree learning from its predecessors and updating the residual errors (difference between an observed y-value and the corresponding predicted y-value), the tree that grows next in the sequence will always learn from an updated version of the residuals. This is known as an additive strategy where what has already been learned is fixed, and a new tree is added one at a time.

The boosting process in its absolute basic can be broken down into the following steps [22]:

- Fit a model to the data: $F_1(x) = y$
- Fit a model to the residuals: $h_1(x) = y - F_1(x)$

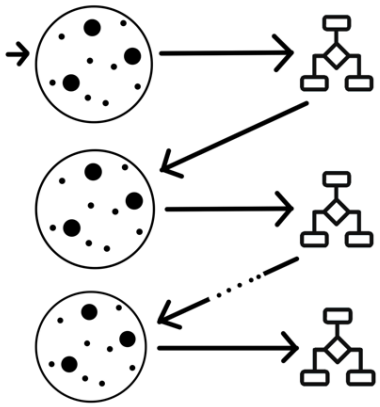

Fig. 2.3 Sequential Tree Structure

- Create a new model: $F_2(x) = F_1(x) + h_1(x)$

By creating more models that correct the errors of the previous models, this can be generalized to:

$$F(x) = F_1(X) \rightarrow F_2(x) = F_1(x) + h_1(x) \ldots .. \rightarrow F_M(x) = F_{M-1}(x) + h_{M-1}(x). \tag{2.4}$$

At each step, the residual would also need to be calculated: $h_m(x) = y - F_m(x)$ where $h_m(x)$ can be any model, but in our case, it is a tree-based learner. With this in mind, suppose that instead of training h_0 on the residuals of F_0, we train h_0 on the gradient of the loss function, $L(y, F_0(x))$ with respect to the prediction values produced by $F_m(x)$. With samples in h_m grouped into leaves, an average gradient can be calculated and then scaled by some factor, γ, such that $F_m + \gamma h_m$ minimizes the loss function for the samples in each leaf. In practice, a different factor is chosen for each leaf. For iteration m = 1 to M:

- Calculate the gradient of L at the point s^{m-1}
- "Step" in the direction of greatest descent (the negative gradient) with step size γ. That is, $s^m = s^{m-1} - \gamma L(s^{m-1})$. If γ is small and M is sufficiently large, s^M will be the location of L 's minimum value.

Most of these are true for all previous gradient boosting algorithms that came before XGBoost, but what really separates it from the others is [22]:

- Regularization: XGBoost can penalize complex models through both L1 and L2 regularization which helps prevent over-fitting.

- Handling sparse data: Missing values or data processing steps like one-hot encoding can make data sparse. XGBoost incorporates a sparsity-aware split finding algorithm that can handle different types of sparsity patterns in the data.
- Weighted quantile sketch: Most existing tree based algorithms can find the split points when the data points are of equal weights (using quantile sketch algorithm). However, they can not handle weighted data. XGBoost has a distributed weighted quantile sketch algorithm that can effectively handle weighted data.
- Block structure for parallel learning: For faster computing, XGBoost can utilize multiple cores on the CPU. Unlike other algorithms, this enables the data layout to be reused by subsequent iterations, instead of computing it again.
- Cache awareness: In XGBoost, non-continuous memory access is required to get the gradient statistics by row index. Hence, XGBoost has been designed to make optimal use of hardware.
- Out-of-core computing: This feature optimizes the available disk space and maximizes its usage when handling huge datasets that do not fit into memory

K-Nearest Neighbours

The k-nearest neighbor (k-NN) algorithm is one of the data mining techniques considered to be among the top 10 techniques for data mining [16]. The k-NN method uses the well-known principle of Cicero pares cum paribus facillime congregantur (birds of a feather flock together or literally equals with equals easily associate). It tries to classify an unknown sample based on the known classification of its neighbors. Let us suppose that a set of samples with known classification is available, the so-called training set. Intuitively, each sample should be classified similarly to its surrounding samples. Therefore, if the classification of a sample is unknown, then it could be predicted by considering the classification of its nearest neighbor samples. Given an unknown sample and a training set, all the distances between the unknown sample and all the samples in the training set can be computed. The distance with the smallest value corresponds to the sample in the training set closest to the unknown sample. Therefore, the unknown sample may be classified based on the classification of this nearest neighbor.

Figure 2.4 shows the k-NN decision rule for k = 1, k = 2 and k = 3 for a set of samples divided into 2 classes X and Y. In Figure 2.4(a), an unknown sample is classified by using only one nearest neighbour and which would label the ? as X; in Figure 2.4(b) two known neighbours are used and since its a tie-breaker, the unknown test instance may be labeled as X or Y. In the last case in Figure 2.4(c), the parameter k is set to 3, so that the closest three

neighbours are considered for classifying the unknown one. Two of them belong to the same class, whereas only one belongs to the other class. Thus the unknown sample ? is labeled as Y.

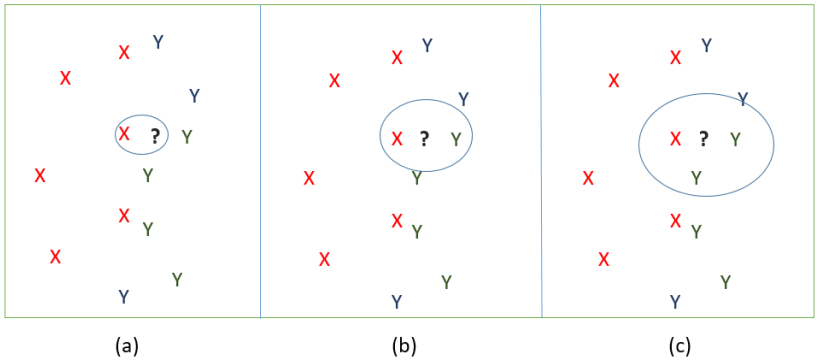

Fig. 2.4 (a) 1-nearest neighbour; (b) 2-nearest neighbours; (c) 3-nearest neighbours

The following Figure 2.5 shows an algorithm in understandable terms.

```
for all the unknown samples UnSample(i)
   for all the known samples Sample(j)
      compute the distance between UnSamples(i) and Sample(j)
   end for
   find the k smallest distances
   locate the corresponding samples Sample(j1),..,Sample(jk)
   assign UnSample(i) to the class which appears more frequently
end for
```

Fig. 2.5 K-NN Algorithm [16]

Support Vector Machine

A support vector machine (SVM) is a type of supervised machine learning classification algorithm which outputs an optimal hyperplane that categorizes new examples given labeled training data [15]. SVMs were introduced initially in 1960s and were later refined in 1990s. However, it is only now that they are becoming very popular, owing to their ability to achieve outstanding results.

Simple SVM: In case of linearly separable data in two dimensions, as shown in Figure 2.6, a typical machine learning algorithm tries to find a boundary that divides the data in such a way that the misclassification error can be minimized. If you closely look at Figure 2.6, there can be several boundaries that correctly divide the data points. The two dashed lines as well as one solid line classify the data correctly.

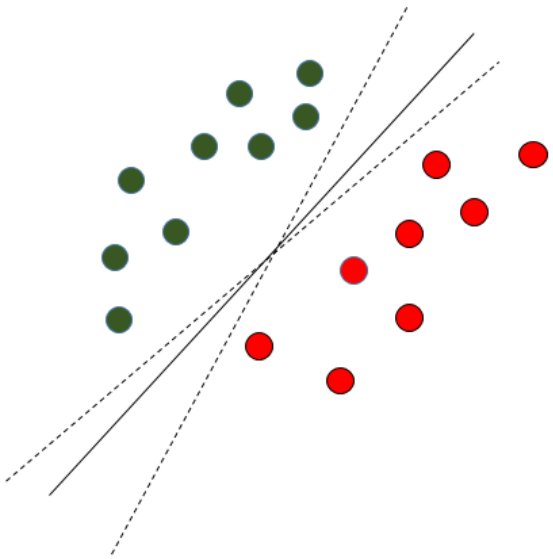

Fig. 2.6 Multiple Decision Boundaries

SVM differs from the other classification algorithms in the way that it chooses the decision boundary that maximizes the distance from the nearest data points of all the classes. An SVM doesn't merely find a decision boundary; it finds the most optimal decision boundary. The most optimal decision boundary is the one which has maximum margin from the nearest points of all the classes. The nearest points from the decision boundary that maximize the distance between the decision boundary and the points are called support vectors as seen in Figure¬2.7. The decision boundary in case of support vector machines is called the maximum margin classifier, or the maximum margin hyper plane.

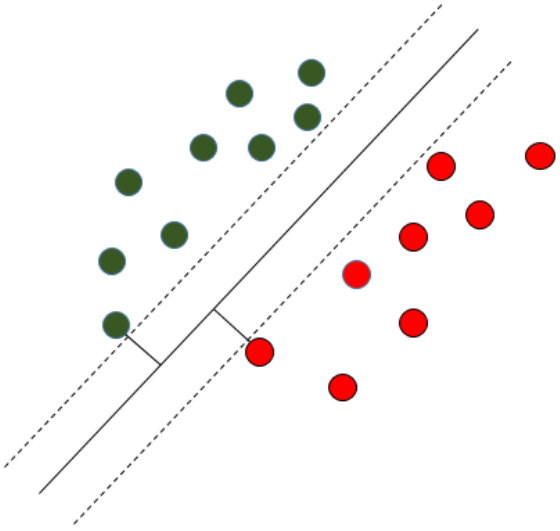

Fig. 2.7 Decision Boundary with Support Vectors

Kernel SVM: In the previous two figures Figure 2.6 and Figure 2.7 it was shown how the simple SVM algorithm can be used to find decision boundary for linearly separable data. However, in the case of non-linearly separable data, such as the one shown in Figure 2.8, a straight line cannot be used as a decision boundary.

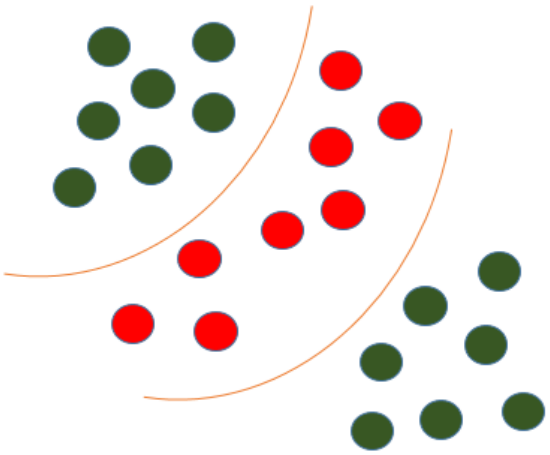

Fig. 2.8 Non-linearly Separable Data

In case of non-linearly separable data, the simple SVM algorithm cannot be used. Rather, a modified version of SVM, called Kernel SVM, is used. Basically, the kernel SVM projects the non-linearly separable data lower dimensions to linearly separable data in higher dimensions in such a way that data points belonging to different classes are allocated to different dimensions.

2.3 Sentiment Analysis

Sentiment analysis is the measurement of favorable, unfavorable and neutral language. It can evaluate how a person feels by their emotions, opinions and attitudes. It can also be referred to as opinion mining. It basically examines the problem of studying texts, like posts and reviews, uploaded by users on various platforms, forums and electronic businesses, regarding the opinions they have about a product, service, event, person or idea by extracting the subjective information of the source material. From the book "Sentiment Analysis and Opinion Mining"[4], we learned about the three different levels of sentiment analysis which consists of the document level, sentence level and aspect and entity level and the problems associated with it and also enhanced on how to deal with conditional and sarcastic sentences, we also got to know how aspects can be extracted using Supervised Learning algorithms and how those extractions can be grouped into categories. Furthermore, we learned about the dictionary-based approach why it is formed how it can help people to find a large number of sentiment words with their orientations, we did some similar type analysis in our EDA in the Section of Review analysis [4.2]. This book basically introduced the field of sentiment analysis and opinion mining and surveyed the current state-of-the-art. It is a very important measurement as it can help you expose to the likes and dislikes of your product, service, or event by the customers. It is therefore considered as a crucial ingredient for managing the customer experience. We collected the URLs of over 5000 apps and then we scraped the data of those URLs from Google Play Store. The dataset that we used had 5223 unique apps and contained over 170000 reviews. As we skimmed through relevant papers like [7,10,11], we observed that in most of the papers after their feature extraction Sentiment value is considered as one of the main features for any sort of prediction analysis of any app. As a result, after scraping the reviews, we preprocessed the reviews and then calculated the sentiment mean for each review using TextBlob,which is a Python library for processing textual data. It has more human like interface, provides more advanced modules and is much easier to understand than NLTK's functionality, considering the Sentiment_subject and Sentiment_polarity column. After the preprocessing of the review column only two lines of code is required to find the sentiment mean using textblob.

2.4 Voting Ensemble

The Ensemble Vote Classifier is a meta-classifier for combining similar or conceptually different machine learning classifiers for classification via majority or plurality voting [18]. Ensemble learning is primarily used to improve the (classification, prediction, function

approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one. Usually, ensemble gives a boost in the accuracy of a model.

2.4.1 Majority Voting / Hard Voting

Hard voting is the simplest case of majority voting. Here, we predict the class label y^2 via majority (plurality) voting of each classifier Cj. The Figure 2.9 illustrates how it all works.

$$\hat{y} = mode\{C_1(x), C_2(x), ..., C_m(x)\} \tag{2.5}$$

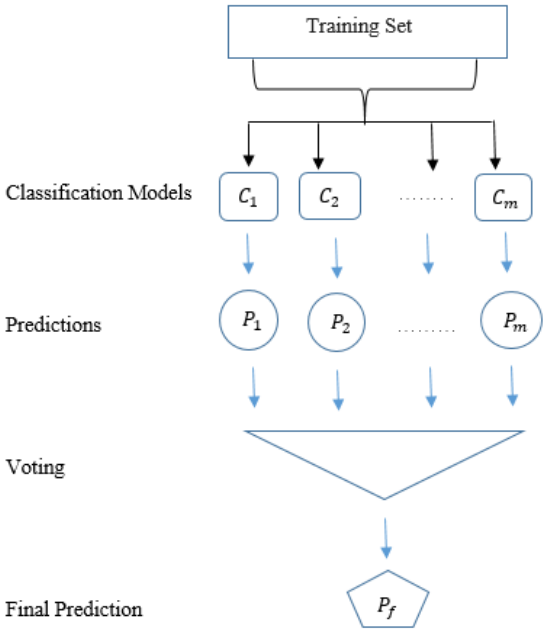

Fig. 2.9 Majority Voting Ensemble

Assuming that we combine three classifiers which classify a training sample as follows:

- classifier 1 = Class 1
- classifier 2 = Class 1
- classifier 2 = Class 0

The output of the voting ensemble would be:

$$\hat{y} = mode\{1, 1, 0\} = 1 \tag{2.6}$$

2.4.2 Weighted Majority Vote

In addition to the simple majority vote as described earlier, we can compute a weighted majority vote by associating a weight w_j with classifier C_j:

$$\hat{y} = arg\max_i \sum_{j=1}^{m} w_j X_A C_j(x) = i \tag{2.7}$$

where X_A is the characteristic function $[C_j(x) = i \in A]$, and A is the set of unique class labels.

Continuing with the example from the previous section

- classifier 1 = Class 1
- classifier 2 = class 1
- classifier 3 = class 0

assigning the weights {0.2, 0.2, 0.6} would yield a prediction $\hat{y}$=0

$$\hat{y} = arg\max_i [0.2 \times i_1 + 0.2 \times i_1 + 0.6 \times i_0] = 0 \tag{2.8}$$

Chapter 3

Data Collection and Processing

The dataset [7] that we first downloaded from Kaggle had about 10,000 apps, but only about 600 of them had corresponding reviews. Since we planned on using sentiment to influence our model, having the details of only 600 apps was insufficient to build a model that holds some ground truth. Thus, we filtered 6,233 app names from the original kaggle dataset, having atleast 100 reviews and extracted information of those apps from the Play Store ourselves. This chapter looks at the process via which it was accomplished.

3.1 URL Collection

A major hurdle with forming our own dataset was fetching URLs via app name from the original dataset downloaded from Kaggle. A URL of an app from the Google Play Store typically looks like the following: https://play.google.com/store/apps/details?id=com.ID For instance, the popular messaging app, "WhatsApp Messenger" has the following URL: https://play.google.com/store/apps/details?id=com.whatsapp At first glance, it might seem very obvious, that ID is the first word of the app name, but that is not the case. If we look at "Messenger – Text and Video Chat for Free", another popular messaging app, the URL looks like the following: https://play.google.com/store/apps/details?id=com.facebook.orca As can be seen above, the URL's ID has a word 'orca' that is not present anywhere in the app name. Hence, forming Play Store app URLs solely from app names is not possible as there is no fixed relation.

The only way to collect URLs ourselves, without any paid third-party service seemed like doing it manually, getting a human to sit behind a computer and look up an app in the Play Store, click on the app, and then paste the link in an excel file accordingly. And that is exactly what we did, only that we removed the human part. The use of a library called pynput helped us accomplish that. Pynput is a Python library that allows one to control and

monitor input devices such as a keyboard and mouse. For instance, after creating a mouse object via:
from pynput.mouse import Button, Controller as MouseController
mouse = MouseController()
we could obtain the current pointer position using: *print("Current pos: ",mouse.position)*
and set pointer position to a coordinate (x,y) using: *mouse.position = (x, y)*
A left and right click could also be simulated via:
mouse.press(Button.left)
mouse.press(Button.right)

With such a powerful tool in our hands, we noted down the coordinates for the following components of the Google Play Store site:

- Play Store logo
- Search bar
- Search button
- First app shown in the results
- URL bar
- Copy option

While pynput's keyboard controller has a function called "type(text)" that allows entire sentences (in our case, app names) to be typed in one go, it was being problematic at the time. Hence, we had to fetch an app name and and process every single character individually using:

- keyboard.press('a')
- keyboard.release('a')

We, also had to convert the name entirely to lowercase and look for "&" in the name and replace it with "and" because the keyboard controller was encountering difficulty with processing that. Then we had the script activate the search bar in Google Play Store site, fill in the name, click on the search button, wait for 1s for the page to load, click on the first app, wait for another 1s for the next page to load, activate the URL, copy the contents, collect the clipboard data using a "tkinter" library function called "clipboard_get", append the CSV file and repeat the process until all the URLs had been collected. Initially, we also realized that we

were encountering timeout errors and collecting invalid URLs in the process. As a result, the previous search results remained and a new app name was appended to the previous one in the search bar, resulting in consequent queries returning invalid URLs. To mitigate that we put a safety in place, checking if a URL contained *https://play.google.com/store/apps/details?id=* in its head before adding it to our CSV file. If not, we had to discard that URL and refresh the page by having the script click on the Play Store logo. The flowchart in Figure 3.1 illustrates this process.

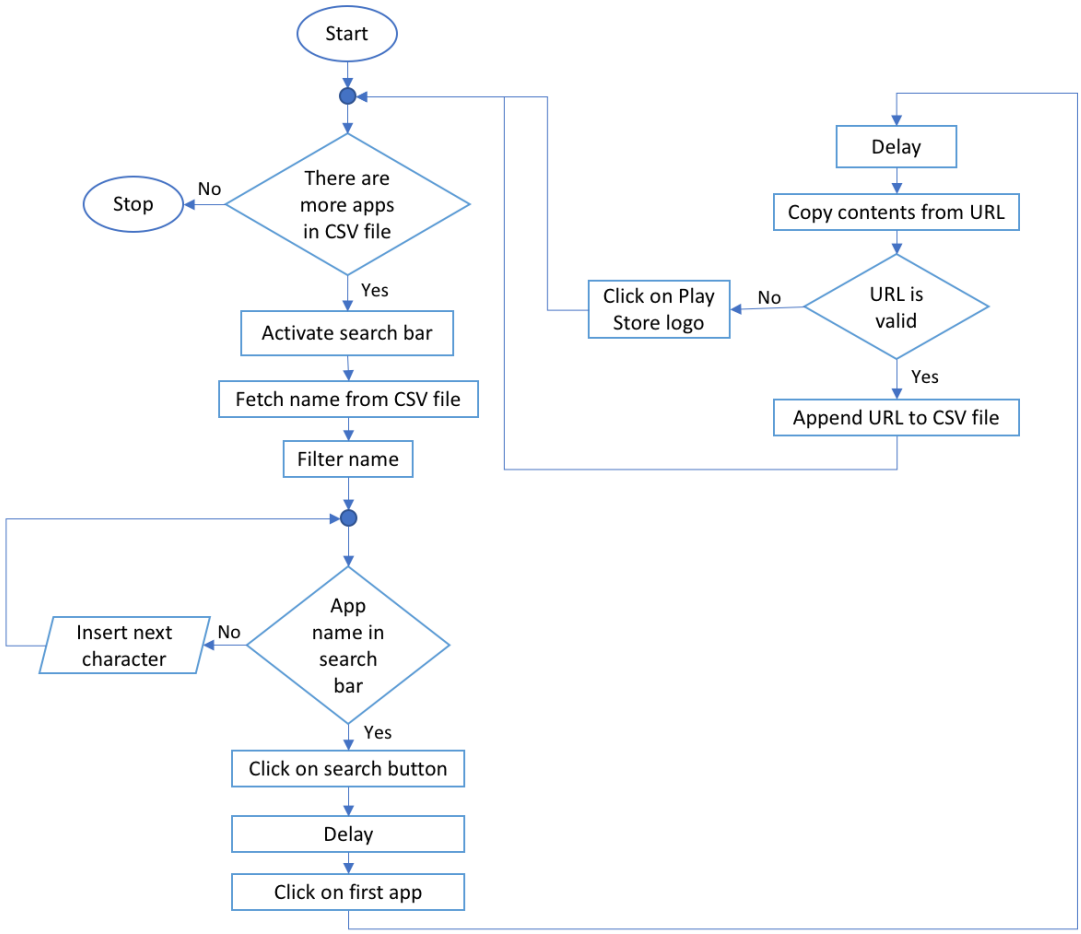

Fig. 3.1 Flowchart of the URL collection process

By this way, we were able to replicate the actions of a human as described before, only much faster, collecting some 5568 URLs overnight.

A likely concern with this approach maybe that the first app shown in the results may not be the app that we were specifically looking for due to another similarly named app being vastly more popular, but it should not be, because later in the scraping process while forming our own dataset, we collect every information from scratch, including the app name.

3.2 Data Collection

In the above section, the method of how we collected the URLs of over 5000 apps has been described. The URLs were saved in a spreadsheet file to be used for web-scraping of Google Play Store App details.

3.2.1 Web Scraping

Web-scraping is a method of data mining from web sites that uses software to extract all the information available from the targeted site by simulating human behaviour. This method mostly focuses on the transformation of unstructured data (HTML format) on the web into structured data (database or spreadsheet). A lot of unstructured data is available on the internet. Once gathered and put into a structured & meaningful format, this data can be used to perform analytics and derive meaningful insights.

There are several ways to extract information from the web. Use of APIs being probably the best way to extract data from a website. Almost all large websites like Twitter, Facebook, Google, Twitter and Stack-Overflow provide APIs to access their data in a more structured manner. If one can get what he needs through an API, it is almost always preferred approach over web scrapping. However, APIs do not offer all the data or one's desired data, much like in our case. Google Play Store Apps has text reviews along with the star ratings that are not accessible using the APIs available and hence the need for scraping the Play Store site.

We used Beautiful Soup 4 (BS4) to extract data from Play Store site. Beautiful Soup is a Python library for pulling data out of HTML and XML files. Python has several other options for HTML scraping in addition to BeatifulSoup. Here are some others: mechanize, scrapemark, scrapy. However, Beautiful Soup 4 is easy and intuitive to work on. It is an incredible tool for pulling out information from a webpage. It can used to extract tables, lists, paragraphs and can also put filters to extract information from web pages. Understanding the basics of HTML tags is preliminary to scraping using BS4. Every <tag> in the HTML documents serves as a block inside the webpages. HTML tags also come with id or class attributes. The attribute specifies a unique id for an HTML tag and the value must be unique within the HTML document. The class attribute is used to define equal styles for HTML tags with the same class. These IDs and CLASS were used to locate the data we wanted.

3.2.2 HTML layout of Play Store Apps

The following syntax shows part of the HTML elements in play store webpage of the app "Sketch – Draw & Paint".

```
<!DOCTYPE html>
<html lang=”en” dir=”ltr”>
  <head>. . . </head>
    <body>
      .
      .
      .
      <div class=” xyOfqd”>
        <div class=” hAyfc”>
          <div class=” BgcNfc”>Installs</div>
          <span class=” htlgb”>
            <div>
              <span class=”htlgb”>100,000,000+ </span>
            </div>
          </span>
        </div>
          .
          .                                .
      </div>
      .
      .
      .
    </body>
</html>
```

The div and span classes hold the same information for all apps. For example, if you want to know the number of installs of another app say “WhatsApp Messenger”, you can directly search in the span class named “htlgb”. The reason behind the consistent use of div and span classes by web developers is usually to maintain the structure of the webpage and make the website responsive. Like the installs count, other attributes such as app name, size, average rating, total number of ratings, rating distribution, date of last update, category, content rating of the app were also located in a similar manne. The text reviews/comments along with their corresponding star rating were not contained in any such such classes or divs and were extracted in a different way. This is because the review/comments of the users are rendered using JavaScript and BeautifulSoup does not execute Javascript so any data delivered or rendered via JS is not available if scraped with BeautifulSoup. However, Beautiful soup allows one to pull the whole text of the webpage using the command “soup.getText()” and

then make the whole text "pretty" using the command "soup.prettify()" to help read the HTML webpage better. So we extracted all the text from the page. Upon observation, a consistent pattern was found to exist in the text where the review-text and its corresponding rating lied. A python script was aimed to fetch the review-text and its corresponding rating and append into a python list. As requesting data too aggressively with our program may be considered as spamming and may break the website, we relaxed our program to behave in a more reasonable manner and extracted only 4 reviews per second, which resulted in at most 40 reviews per app and took about 10 seconds to scrape at most 40 recent most-up-voted review texts of the app. After identifying the location of the meta-data of the apps, the scraper was fed with over 5000 URLS. A loop was iterated to extract information of each app and store all the information in a pandas dataframe and finally into a CSV (comma-separated-values) file to be used for further analysis. Just like Microsoft Excel, Pandas DataFrame provides various functionalities to analyse, change, and extract valuable information from a given dataset. The raw data that was scraped contained 180,438 rows having 5226 unique apps. The limitation of our scraping techniques let us extract at most 40 reviews/comments for each app.

3.2.3 Data

A sample of the data head is Table 3.1, Table 3.2 and Table 3.3. The 15 attributes are broken down into three tables for better visualization.

App	Category	Content Rating	Installs	Rating	Last Updated
Toca Life: City	Education	Everyone	500,000+	4.7	July 6, 2018
Toca Life: City	Education	Everyone	500,000+	4.7	July 6, 2018
Toca Life: City	Education	Everyone	500,000+	4.7	July 6, 2018
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
Infinite Painter	Art & Design	Everyone	1,000,000+	4.1	November 9, 2018
Infinite Painter	Art & Design	Everyone	1,000,000+	4.1	November 9, 2018

Table 3.1 Attributes in raw format

App	#Ratings	#Rating1	#Rating2	#Rating3	#Rating4	#Rating5	Type
Toca Life: City	32,181	1,160	416	1,089	2,439	27,077	Paid
Toca Life: City	32,181	1,160	416	1,089	2,439	27,077	Paid
Toca Life: City	32,181	1,160	416	1,089	2,439	27,077	Paid
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Infinite Painter	38,955	4,450	1,839	3,503	6,100	23,063	Free
Infinite Painter	38,955	4,450	1,839	3,503	6,100	23,063	Free

Table 3.2 Other attributes in raw format

App	Rating Text	Review Text	Size
Toca Life: City	4	Lower the prices please...	24M
Toca Life: City	5	I love Toca Life games so much...	24M
.	.	.	.
.	.	.	.
.	.	.	.
Infinite Painter	2	I would give 5 stars for its efficient...	Varies with device
Infinite Painter	4	I like the app. It's the first...	Varies with device

Table 3.3 Rest of the attributes in raw format

3.3 Data Description

The column header of Table 3.1, Table 3.2 and Table 3.3 are briefly explained in the following Table 3.4.

Attribute	Description
App	Name of the app
Category	Category or genre, e.g. Education, Tools
Content Rating	Suitable content for the audience
Installs	Number of installs e.g. 1+, 5+, 10+,... 1000+
Rating	Average rating of the app
Last Updated	Date of last update
#Ratings	Total number of ratings submitted
Rating Distribution	Number of 1-star, 2-star, ... , 5-star ratings
Type	Free or Paid
Rating Text	Rating submitted by a user
Review Text	The comment text entered by a user
Size	Size of app in k (kilobyte), M (megabyte), ...

Table 3.4 Description of the attributes

3.4 Data Processing

Since we scraped all the app details ourselves, there were no null/missing values in the dataset. However, The raw data extracted needed to be pre-processesed to turn it into some valuable information. To be able to perform EDA and run algorithms on our dataset, installs, total number of ratings, rating distribution were converted to integers. Sizes of app having kilobytes were converted to megabytes to have consistent units. Apps with varying sizes (those that vary with device) were set to the average size of the rest of the apps. Later the categorical attributes like categories/genre, content rating and type were label encoded, for instance free apps were labeled as 1 and paid apps were labeled as 0.

Further, to find out the sentiment of the users, the review text was processed using TextBlob. Polarity and subjectivity of each of the reviews were found with polarity values between -1 and 1 and subjectivity values between 0 and 1. A polarity value of 1 means a strongly positive feedback by the user and a polarity value of 0 means otherwise. On the other side, a higher value of subjectivity means the review is more subjective and hence is not good.

The recent 40 review of each app were grouped and their median sentiment was calculated, similarly the rating that came along the review were grouped for each app and their mean was calculated. These are later used as features for future success prediction and are discussed extensively in later chapters.

Chapter 4

Exploratory Data Analysis

In this chapter we analyze our dataset to summarize their main features with visual representations to see what the data can tell us beyond the formal modeling.

4.1 App Information

This section contains findings from the research conducted on the first dataset that we formed by scraping information from the Google Play. It contains information of 5223 apps.

4.1.1 Category vs App

The Category column in our dataset has 47 different types of categories. Figure 4.1 shows the bar chart of the number of categories. A closer look at it reveals that there are two separate categories for education, namely 'Education' and 'Educational'. So, we merged them under the single label 'Education'. Also, from the figure it is apparent that the Tools category is the most dominant in the Google Play Store. The category includes apps like SHAREit - Transfer & Share, Google Translate, Gboard - the Google Keyboard, etc. It also came to our attention that games were not classified under a single label, instead they were very specific. The following contains all the different types of games:

- Action
- Casual
- Arcade
- Puzzle

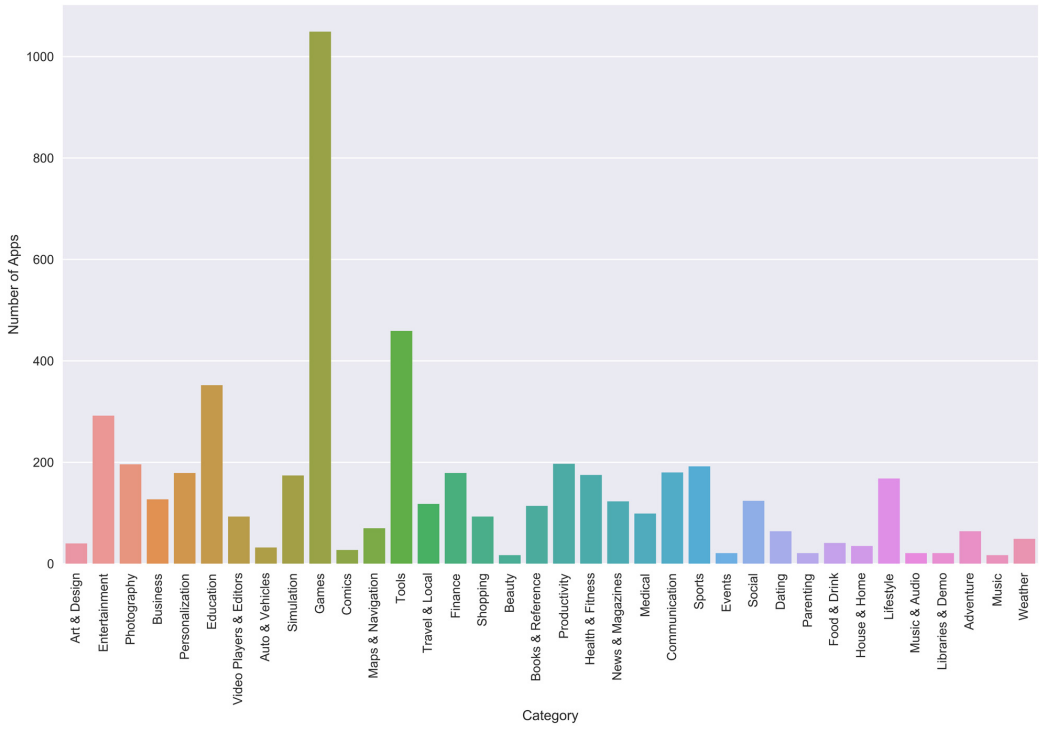

Fig. 4.1 Bar chart of category against number of apps

- Racing
- Role Playing
- Board
- Card
- Casino
- Word
- Trivia
- Strategy

Combining them all under the category, 'Games' resulted in it being the most dominant, with the total number of categories now being 36, down from the original 47. Table 4.1 shows the top ten most dominant categories by number of apps in descending order after this merge.

From Table 4.1 it is clear that Games rule the Play Store followed by Tools, Education and Entertainment.

Place	App	Frequency
1	Games	1049
2	Tools	459
3	Education	352
4	Entertainment	292
5	Productivity	197
6	Photography	196
7	Sports	192
8	Communication	180
9	Personalization	179
10	Finance	179

Table 4.1 Top 10 most dominant categories by number of apps

4.1.2 Category vs Installs

In contrast to our previous results, comparing the top ten most dominant categories by installs tells a slightly different story. Figure 4.2 shows that while Games still rule the Play

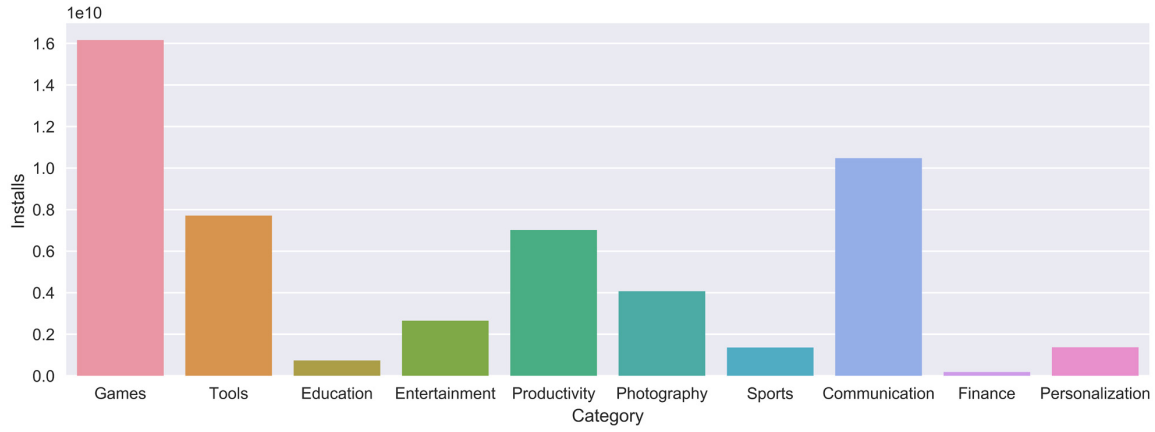

Fig. 4.2 Bar chart of top 10 categories by installs

Store, Tools, previously the 2nd most dominant by app numbers, has been dethroned by Communication which was the 8th most dominant. Similarly, the Education category which previously occupied the 3rd spot has now fallen by six places to the 9th spot. Hence, looking at categories by apps might be misleading for a developer. A developer wanting to attract a large user base should pick a category based on the number of installs and not by the number of apps in the Play Store. Figure 4.3 compares all the categories against the number of installs.

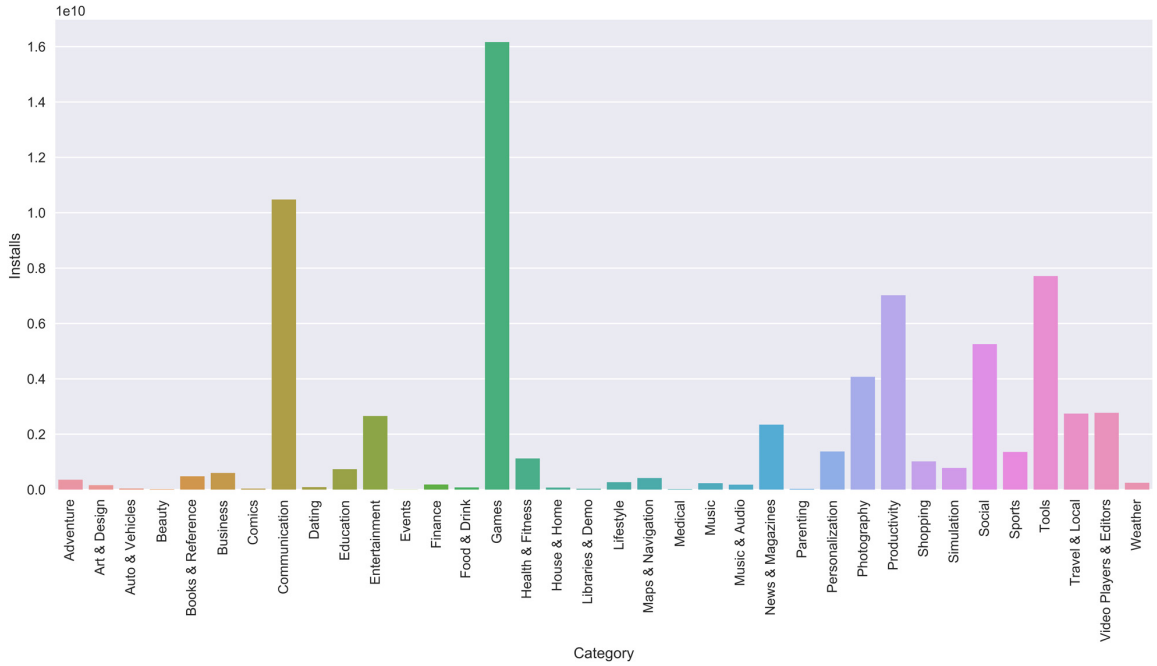

Fig. 4.3 Bar chart of all categories by installs

4.1.3 Installs Correlation

The user base is the most important statistic of an app. Therefore, we wanted to see installs' correspondence with the other features in our dataset. Table 4.2 and Figure 4.4 shows the correlation of installs with other numeric features in the form of values and a heatmap respectively. From there we can conclude that numeric features such as number of ratings,

Correlation of installs versus	Correlation value
Installs	1.000000
No_of_Ratings	0.141699
Size	0.060767
Rating	0.058889
Subjectivity_Mean	0.032624
Sentiment_Mean	-0.042139
Recent_Rating_Mean	-0.045481

Table 4.2 Correlation between installs and other numerical features

overall rating, the mean of the recent ratings, mean of the sentiment of the recent reviews, as well as subjectivity are all uncorrelated with installs. The closest feature is number of ratings as it is likely that the higher the install count, the greater the number of times the app has been rated, but even then, it is far from convincing.

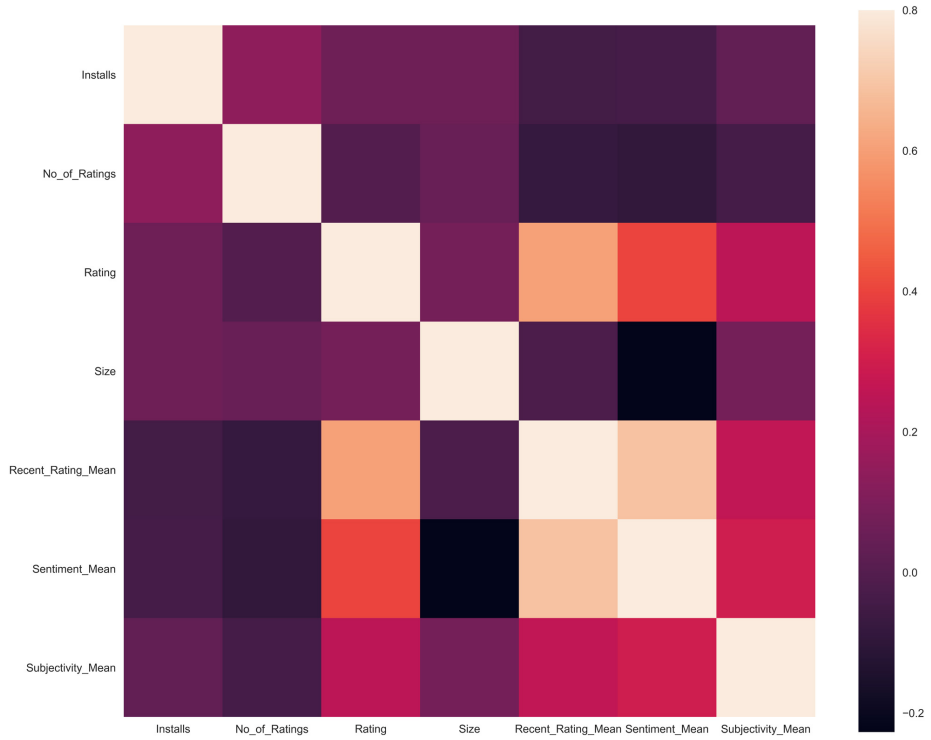

Fig. 4.4 Heat Map of correlation of installs with numerical features

4.1.4 App Name

Next, we identify if there's any correlation with the number of words in an app name and other numeric features by creating a new column in our dataset called "Words in App Name". However, due to the unpromising nature of the results, we have chosen not to include them. Instead, we decided to see how apps with a threshold on the number of words used in their names fared against those without any. Figure 4.5 shows that apps with less than or equal to three words in their names account for almost half the number of installs. When the threshold is increased to 4 words we see the margin increasing significantly in Figure 4.6. It seems that most of the installs in the Play Store are contributed by apps having less than or equal to 4 words in their names. Hence, it is better that developers use a concise word or words to name their app.

4.1.5 Content Rating

A content rating rates the suitability of an app to its users. While previously plotting Category vs Apps and Category vs Installs returned dissimilar results, that is not the case with content rating as Figure 4.7 and Figure 4.8 show both of them having roughly the same distribution.

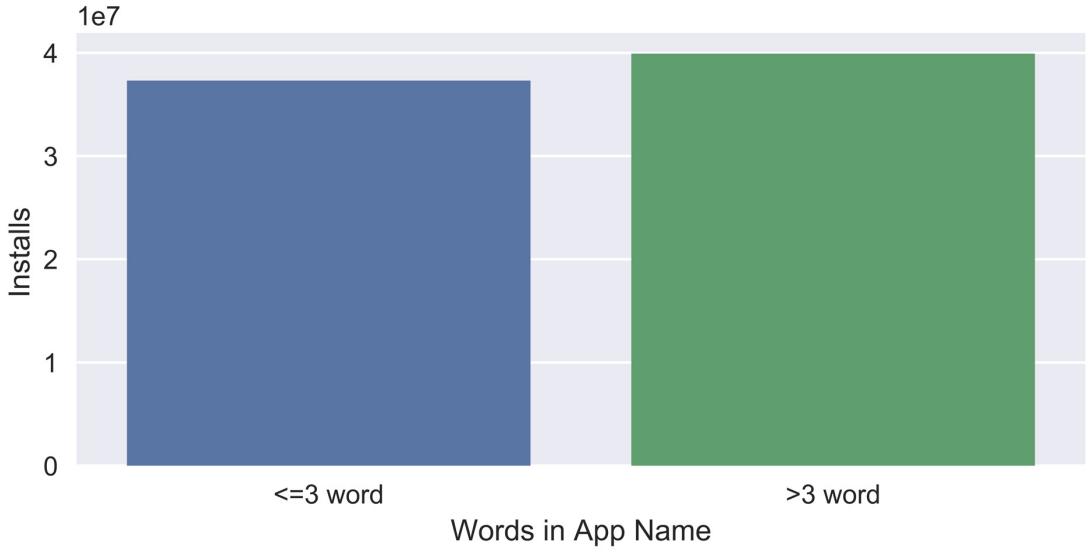

Fig. 4.5 Bar chart of installs versus apps with less than or equal to 3 words and more

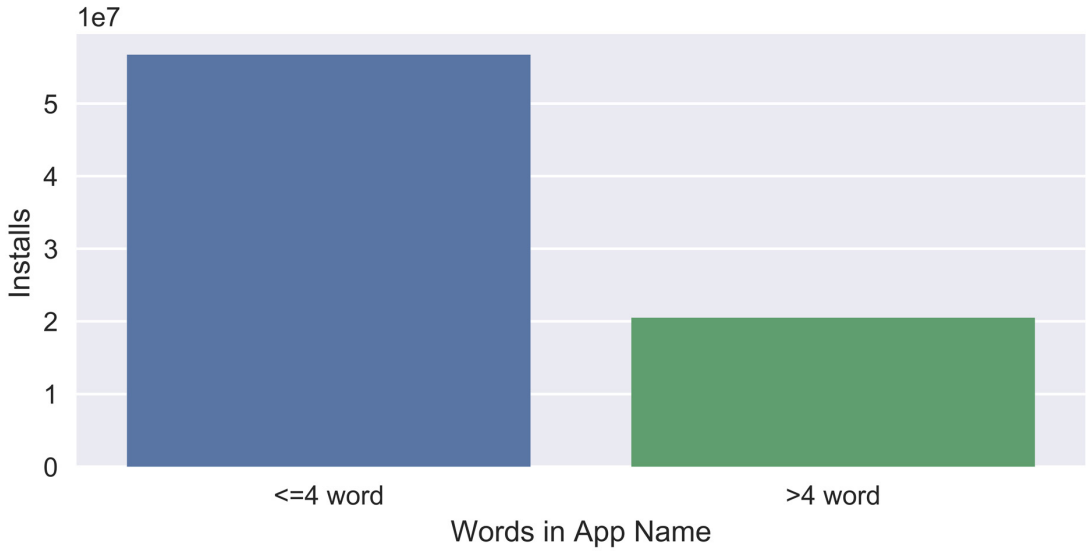

Fig. 4.6 Bar chart of installs versus apps with less than or equal to 4 words and more

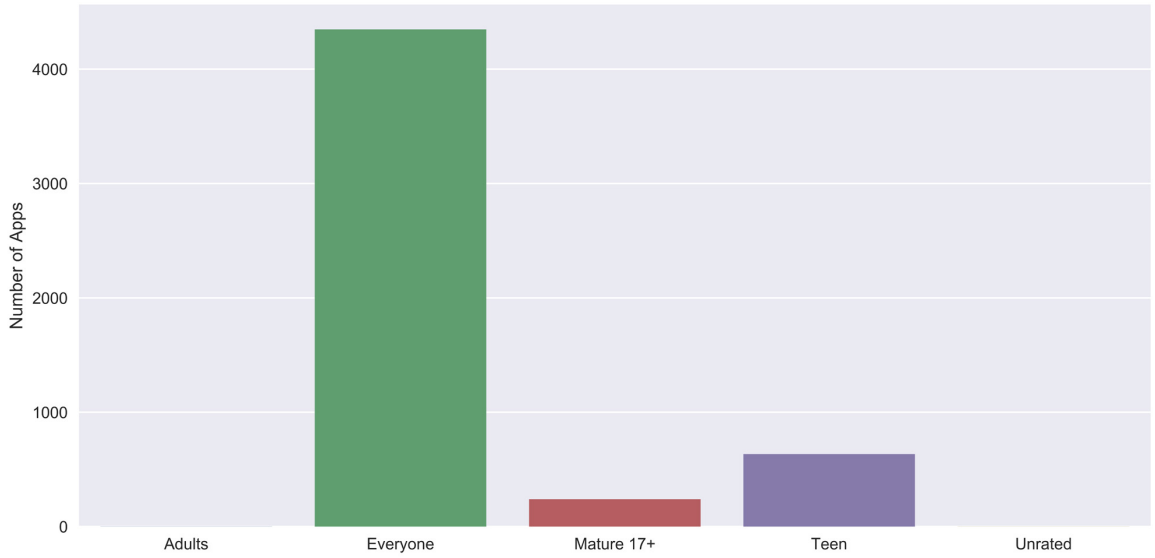

Fig. 4.7 Bar chart of content rating against number of apps

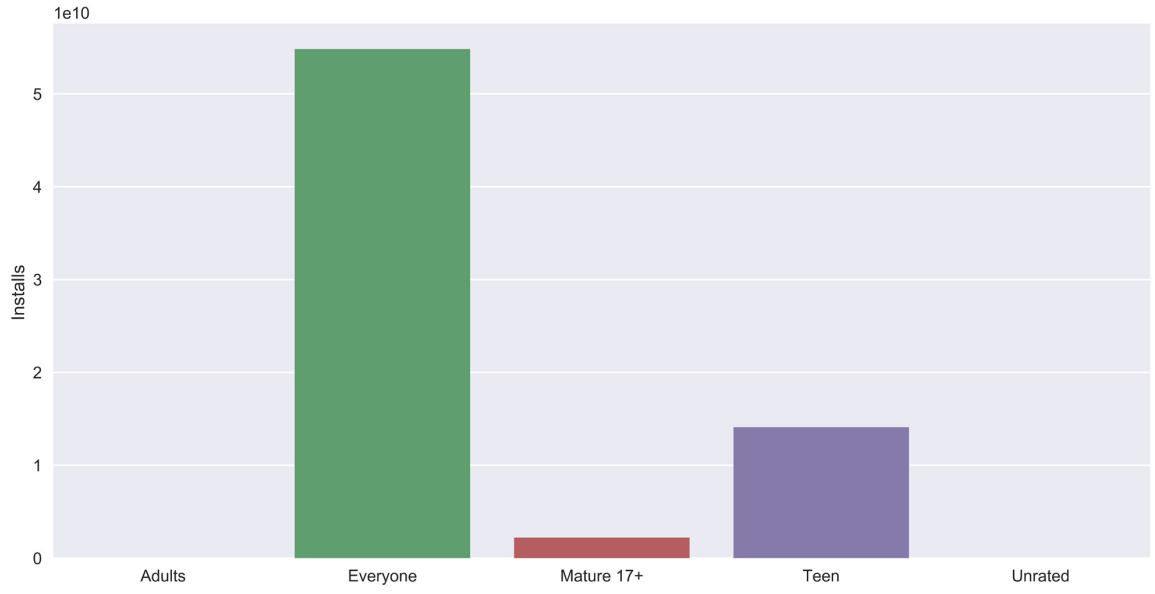

Fig. 4.8 Bar chart of content rating against installs

Adult and Unrated apps are virtually invisible in both Figure 4.7 and Figure 4.8. A reason why this could be is because we worked with 5223 apps and with these two ratings being very low to begin with, our scrapping process missed out on them. But the important thing to note here is that 'Everyone' should be the content rating of choice for a developer, seeking to maximize user base.

4.1.6 Type

Here, in Figure 4.9 and Figure 4.10 we see that free apps crush paid apps, both in terms of number of apps and installs. However, not all of these apps are actually free. Some of them

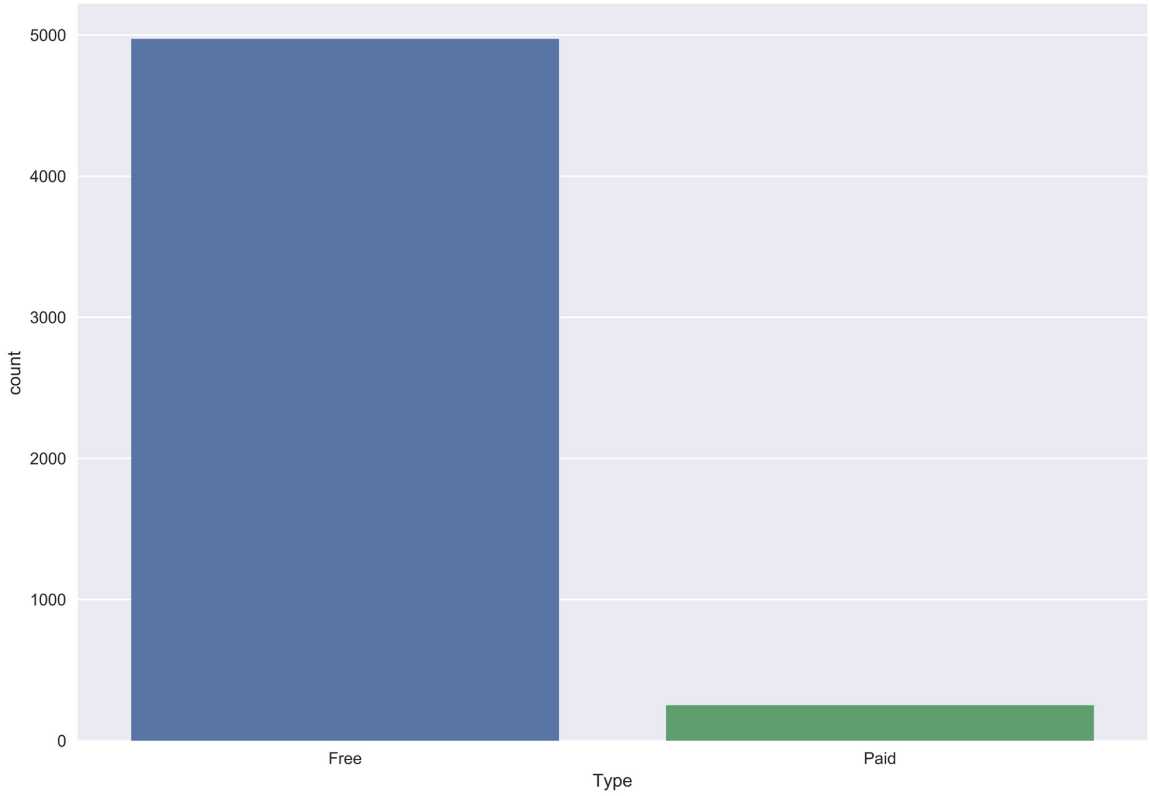

Fig. 4.9 Bar chart of type against apps

do have some form of cost associated with in real life, whether it be in the form of advertising or in app purchases. Since we could not extract those features, we were limited by only the type of an app in our analysis. The outcome of a pie chart of the type (free or paid) of an app with respect to the top four categories by number of apps is shown in Figure 4.11.

While Figure 4.9 and Figure 4.10 confirmed the dominance of 'free' apps, we can see that Tools, Games and Education – all have a small fraction of paid apps, but Entertainment is a

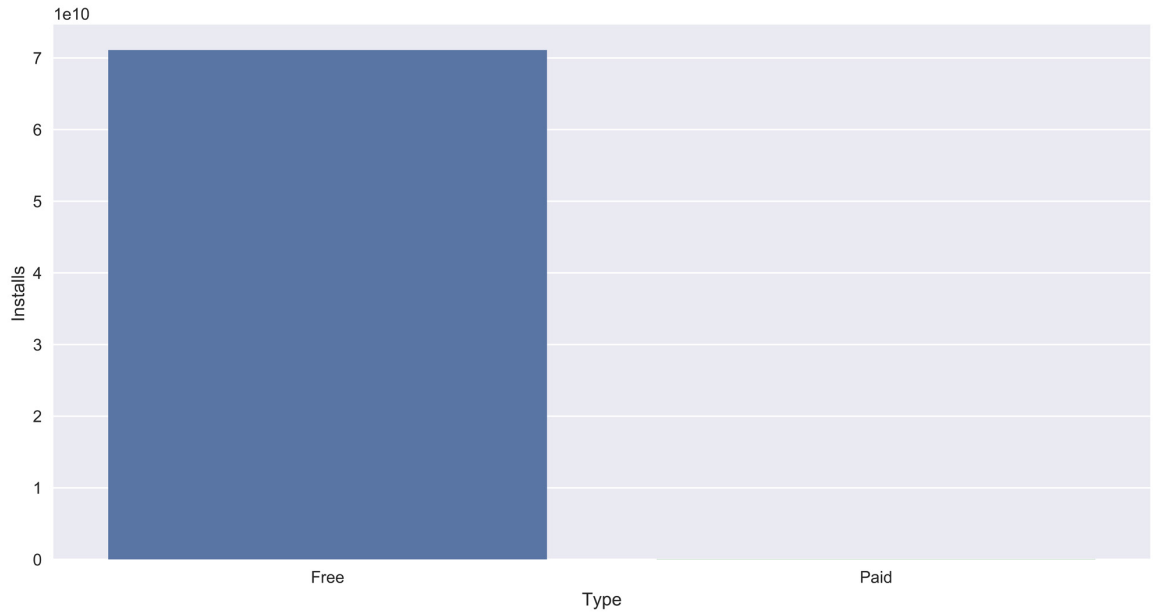

Fig. 4.10 Bar chart of type against installs

different case with barely any visible slice in the paid segment of the pie chart in Figure 4.11. This could likely be due to the fact that entertainment apps are free to download, but require a subscription to use, such as Netflix, Crunchyroll, etc.

4.2 Review Analysis

This section takes a closer look at the reviews and their associated rating. We wanted to see if specific words in a review resulted in a specific rating. The dataset used for the following analysis has 5223 unique apps with a combined total of 174955 reviews.

4.2.1 Dictionary of Words

With an abundance of reviews in our hand, we built a dictionary of words that appeared in the reviews and recorded their occurrences. We began with a fixed list of stop words. Then, we converted every review text to lower case, so as to avoid a word starting with an uppercase or having all uppercase letters being recorded as a different word compared to one with all lower cases later. For instance, this allows all possible case combinations of the word - Awesome, AWESOME and awesome to be treated equally. Next, we used regular expression to remove punctuations in a sentence and only store the words in a list. From there, we scanned the list for duplicate words and removed them. This is so that we do not have the same word more

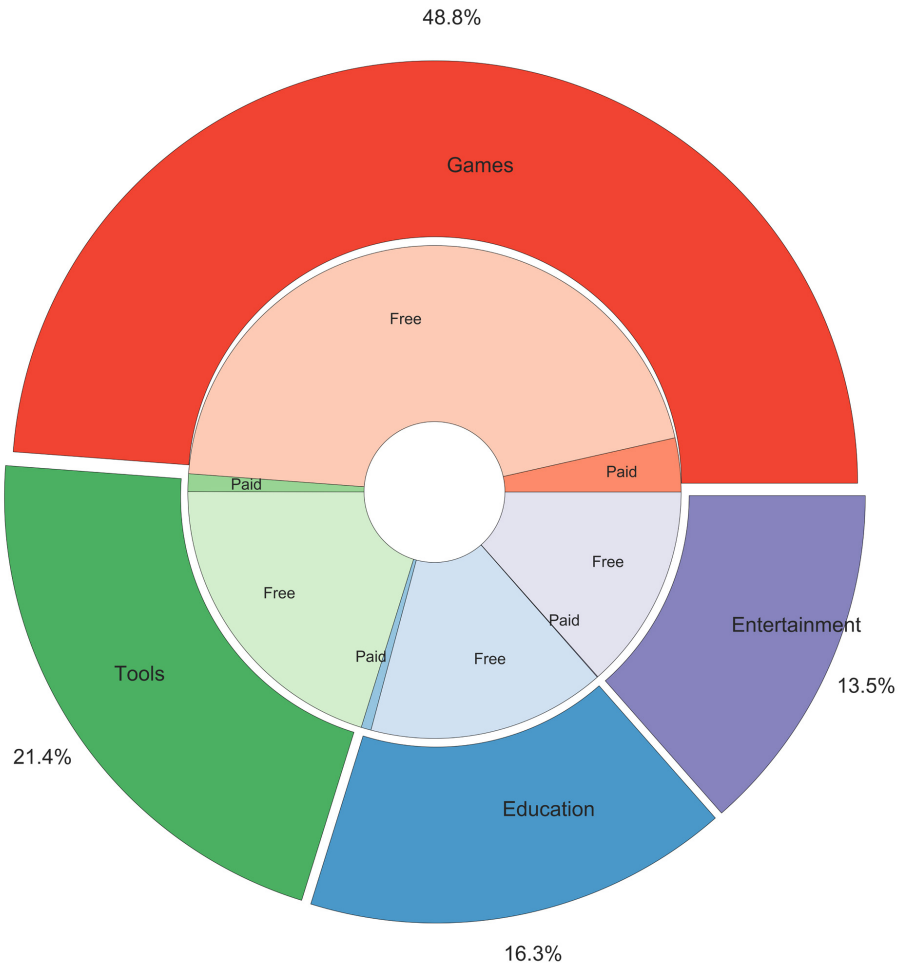

Fig. 4.11 Pie chart of type of app in relation to category

than once for one review. Then, we removed any stop words present in the list. Finally, for every word in that list we either added them to the dictionary if they were not already there and assigned a frequency value of 1 or increased the frequency of a specific word by 1 if it was already present. This process is illustrated in the flowchart in Figure 4.12.

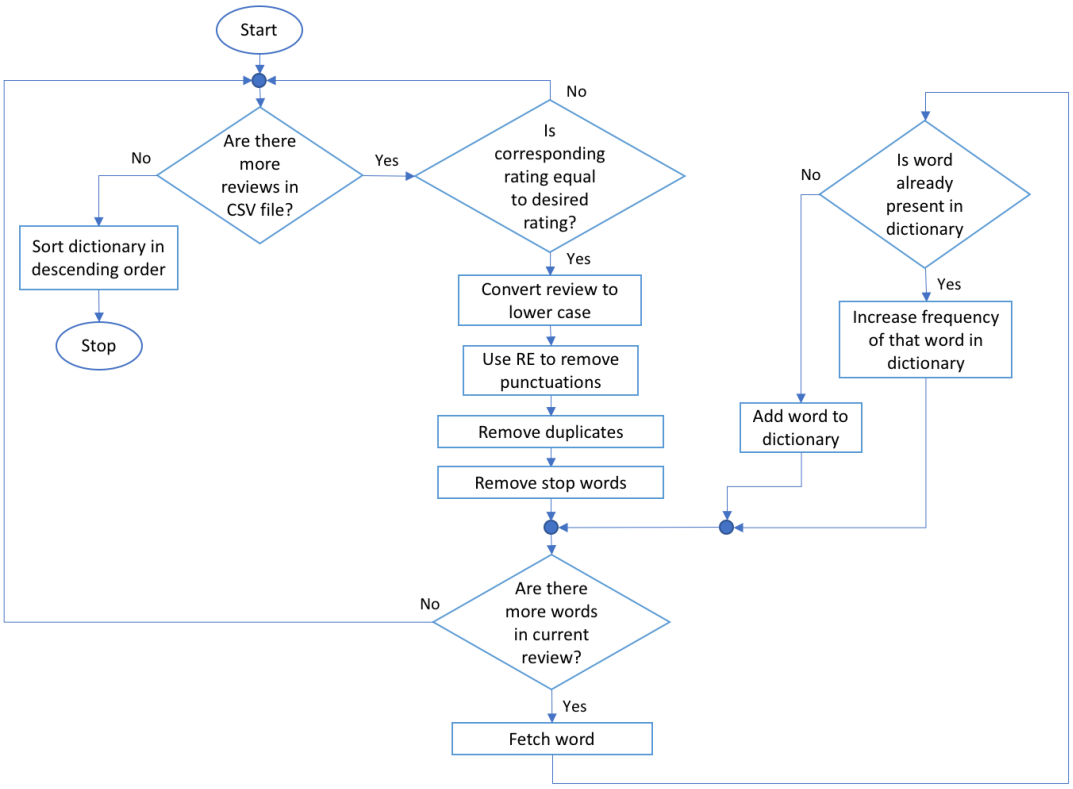

Fig. 4.12 Sorted dictionary of words

We carried this out five times for the five different ratings that one can give to an app in the Play Store and had five different dictionaries, each containing the number of times a word has appeared for a specific rating. Sorting the dictionary in descending order by its value, we noticed one key trend – the words app and game were the top two most frequently used for all the five different ratings which makes sense given that one-fifth of our dataset is composed of apps belonging to the game category. Also, the word bears no relevance to the rating given. Hence, we filtered them from all the five dictionaries to get a more accurate picture of the words that actually impact a rating. The first 25 most frequently used word associated with ratings 1, 2, 3, 4, 5 are illustrated in the Figures 4.13, 4.14, 4.15, 4.16, 4.17 respectively.

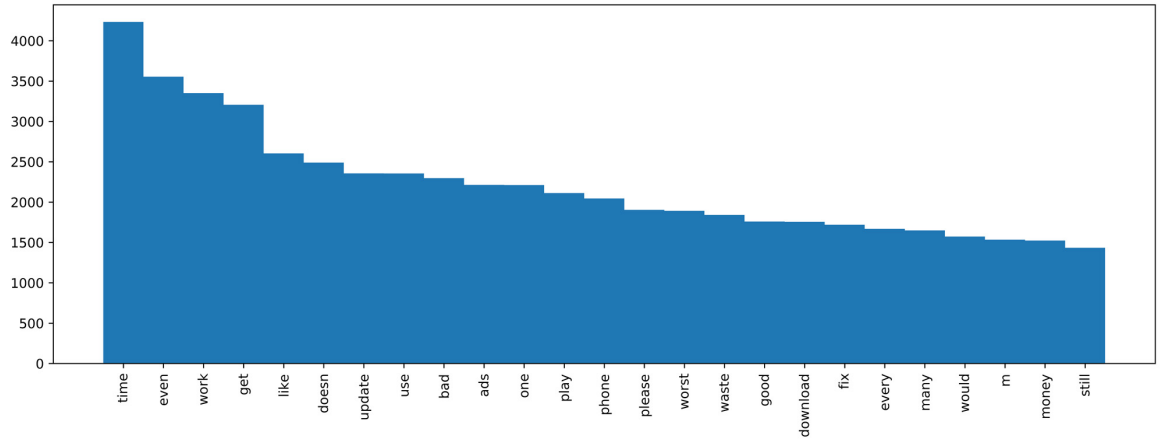

Fig. 4.13 Frequency distribution of words for rating 1

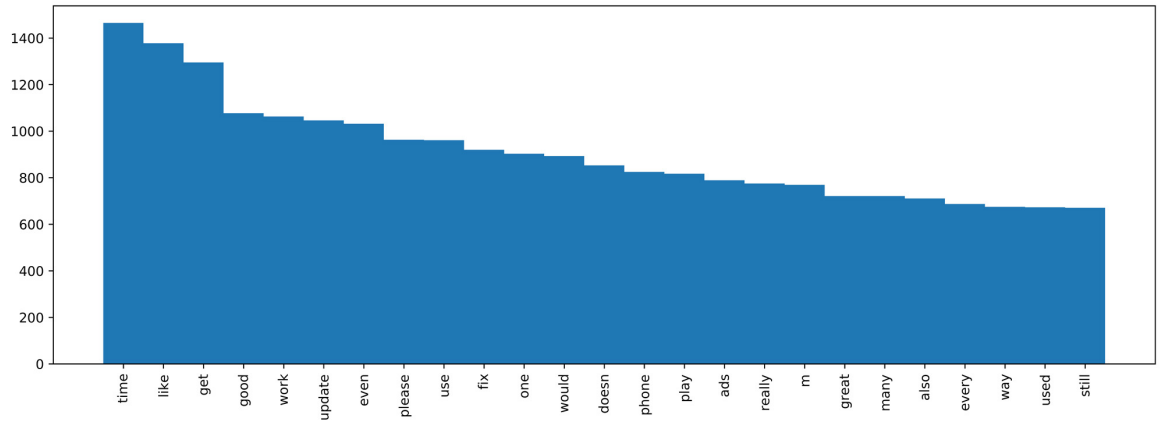

Fig. 4.14 Frequency distribution of words for rating 2

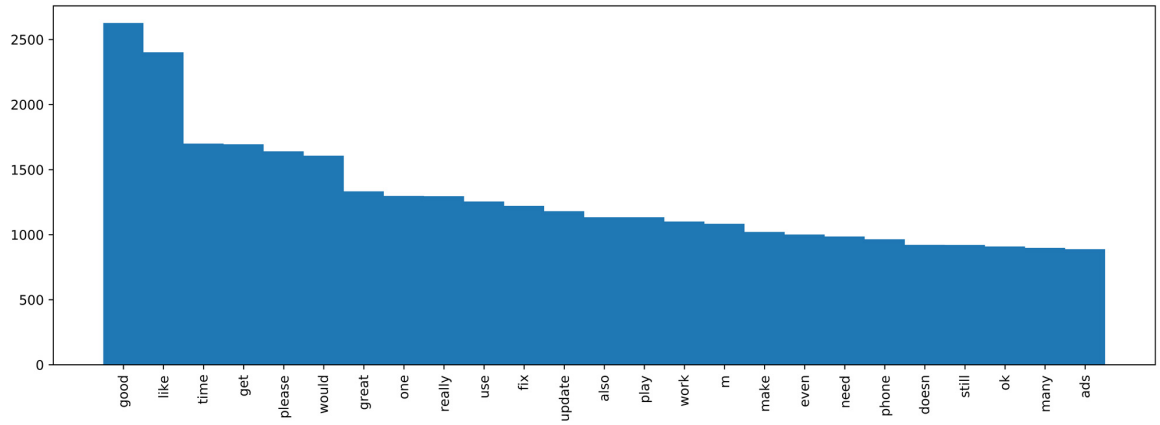

Fig. 4.15 Frequency distribution of words for rating 3

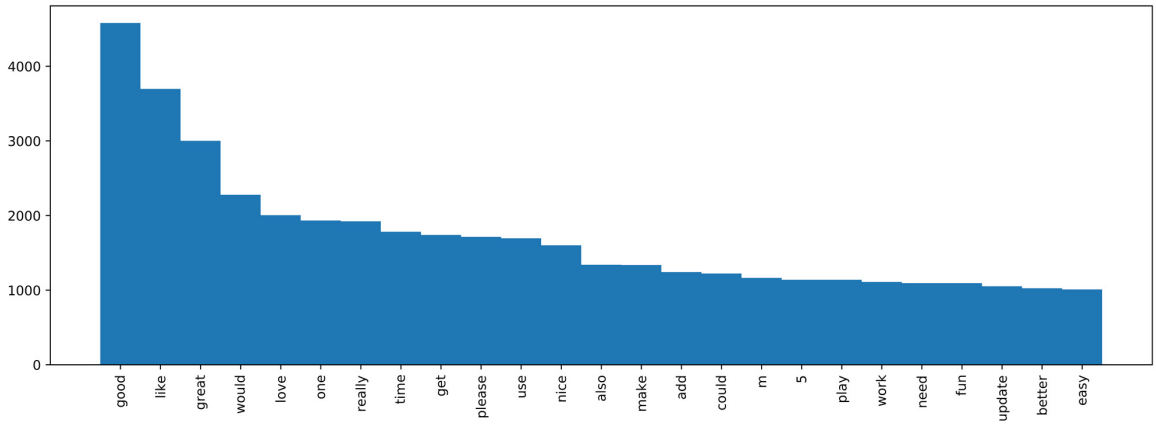

Fig. 4.16 Frequency distribution of words for rating 4

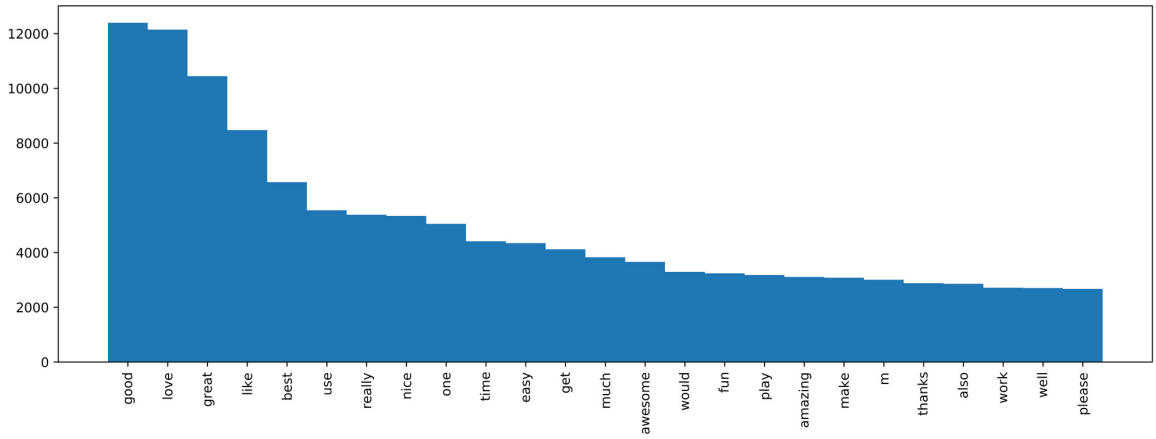

Fig. 4.17 Frequency distribution of words for rating 5

Sentiment	Review
0.7	Post the Oreo update on my Honor 8, the app started to freeze and keeps getting stuck. Have used it through almost all updates till date with no issue... Seems that app is not compatible with Oreo.... Spoilt an otherwise good app...
0.3	After update it's not so good , photo automatically rotate just after capture and changed shape on my honor 7x . Want to get back the old version.
0.4	This app has really been good but .. since it got updated it has really given me so much problem, not letting me get match,and also match finished automatically and when it get minimize it show again by itself ... please you guys should do something about it ok

Table 4.3 Reviews reflecting satisfaction in the past

4.2.2 Rating Features

The bar charts in Figures 4.13, 4.14, 4.15, 4.16, 4.17 of the previous section show that some words overlap among the different ratings. For instance, the word good is prevalent among ratings with three stars, four stars and five stars. Even with a rating of 1 and 2, we see it making it to the top 25 most frequently used word. Querying the dataset for sentiment and reviews with the word good in it and have a corresponding rating of 2 returned results shown in Tables 4.3, 4.4, 4.5.

Tables 4.3 suggests that some users use a positive word like good to describe the previous version of an app that they found satisfactory. Such keywords in reviews corresponding to low ratings could also indicate that a new update has not been well received by an app's user base. Therefore, we first looked for the keyword update and if present, checked if a review had a word from the positive dictionary in it. For instance, there are 2356 reviews corresponding to rating 1, containing the word 'update' and from those reviews, 921 (39.1%) contain at least one word from the positive word list. Similarly, with rating 2 the 'update' word appeared in 1046 reviews and 570 (54.5%) of them had a word from the positive list. Now, for an app with a small user base this could wreak havoc. Even a handful of people, on a specific version of android, encountering issues and therefore giving the app a bad rating would plummet it to the ground, regardless of how good it has been in the past. Therefore, if a user changes his rating to one that is less than say, 2 stars compared to his previous rating, the Play Store can implement a system where that effect would not be seen instantly. Instead, if a specific percentage of users repeat this, the Play Store could alert the developer and give him a week's time before enforcing those ratings into the average rating in order to let the developer undo the changes and gain back the high ratings. Hence, the time period for which potential users would be swayed away because of the poor ratings could be averted.

Sentiment	Review
0.7	The app crashes 2 minutes in. Good app if it would work
0.9	The quality isn't very good.
0.9	Not so good !
-0.3	Not good app

Table 4.4 Reviews reflecting sentiment mismatch

Sentiment	Review
0.7	Good
0.7	Thank u everthing is good .

Table 4.5 Reviews reflecting rating mismatch

However, if we look at the sentiment values on the first column of Table 4.4 and match with the corresponding review, we can see that "Not so good !" has been wrongly assigned a highly positive sentiment of 0.9, whereas a similar phrase "Not good app" has been correctly given a negative sentiment of -0.3. Thus, we can conclude that the sentiment values obtained from the TextBlob API isn't very accurate, as it has clearly mislabeled some reviews with a high sentiment value when it should have been very low or negative. We found further examples of such discrepancies. Moving forward, taking a closer look at the results obtained from the initial query where we only fetched reviews containing the word good and being associated with a 2 star rating, we also found examples of reviews presented in Table 4.5 where we can clearly observe rating mismatches. For instance, a review containing the words "Thank u everthing is good ." has a corresponding rating of 2 when it should have had a rating of 4 or 5. To further investigate this, we created a dictionary of positive and negative words manually, using informing from this dataset, with the hopes of identifying potentially mismatched reviews. Figure 4.18 contains the distribution of the ratings according to the number of positive words they contain.

From Figure 4.18, we can see clearly that the negative word dictionary has not done a superb job differentiating between ratings 1 and 5, as we rating 5 having approximately half as many negative words as rating 1. So, we moved on to the positive word dictionary to see how it affected the ratings. The rating distributions for the positive words are shown in Figure 4.19 where we can see far more consistent results. Rating 5 is clearly the dominant entity here by far as expected, followed by rating 4. However, we also see rating 1 in the mix, but this time the unexpected entity has a much lower percentage. Now, from the previous findings, we can deduce that these ratings are due to either an update that has not sit well with the users, resulting in them expressing how good the previous version of the app was relative to the current one, or it can be a rating mismatch. The latter can be because the developers

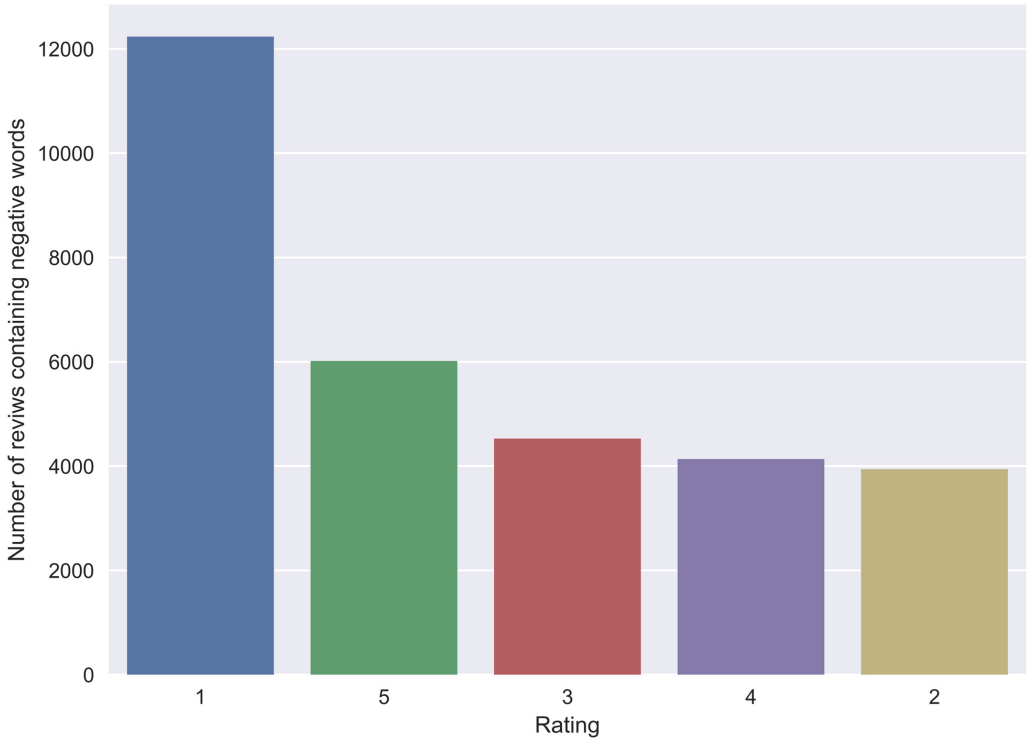

Fig. 4.18 Rating distribution of reviews containing negative words

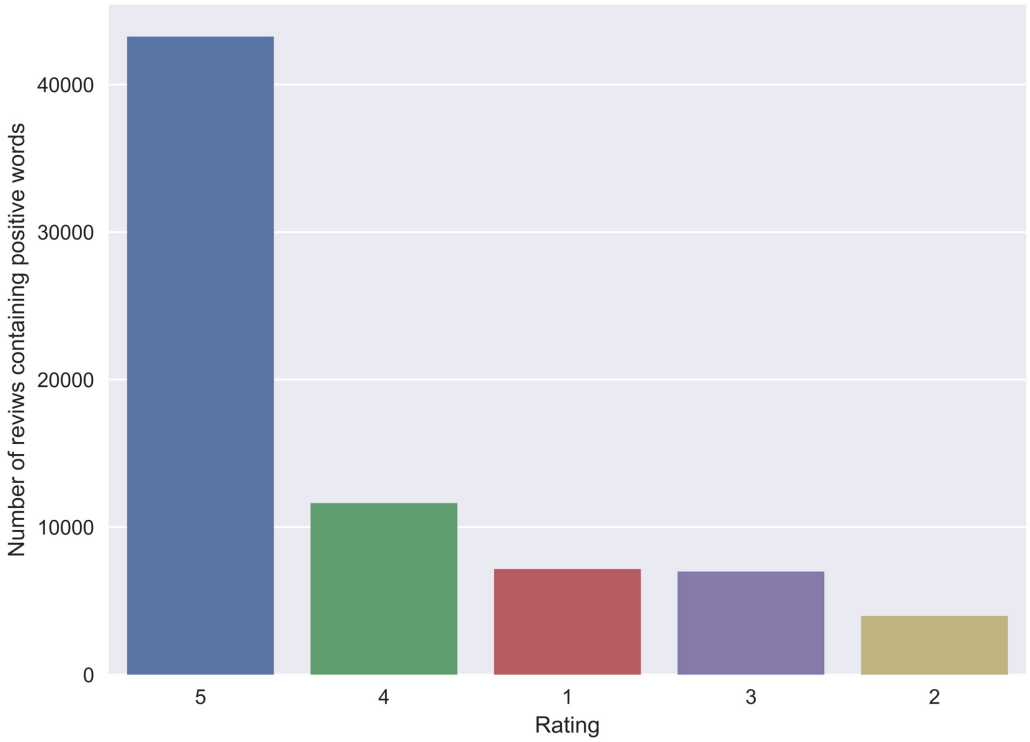

Fig. 4.19 Rating distribution of reviews containing positive words

have added a feature that users have been longing for or fixed existing problems, but while the user has let them know via their positive feedback in the form of posting a review, they have forgotten to update the existing rating for which they were previously unsatisfied. Now, this is a major issue. An app can have an inflated or deflated average rating due to rating mismatches. Such inconsistent ratings can also affect user downloads, as typically users look at the average star ratings while deciding whether to download an app or not [9].

Apart from positive and negative words there are other features associated with a given rating. According to [3] where 8600 reviews had been manually annotated, the following patterns were observed:

1. If a review has all its content in uppercase letters, they indicate frustration and disappointment, resulting in a low star rating.

2. If a review's nature is to question something, it can also be an indication of unhappiness and therefore low star ratings. This can be identified via words like why, when, where, what and question marks.

3. If a review has exclamation(s), they tend to be expressing satisfactoriness and hence correspond to a high star rating.

Therefore, we wanted to see if such patterns would be limited to just one dataset or extrapolate to ours as well. We scanned all the reviews in our dataset and created a new binary column that was assigned a value of 1 or 0 based on whether a review had all capital lettered words or not respectively. Then, plotting a bar chart of all reviews containing only uppercase letters against their corresponding rating gave us Figure 4.20 where we see a completely different result. The reviews in our dataset follow the complete opposite pattern. Instead of reviews with capital lettered words representing a low star rating, the majority of them actually represented a 5 star rating. Rating mismatch could be an issue here, but definitely not big enough to cause such a massive gap. Therefore, such a feature is highly dependent on the dataset used.

Next, we take a look a look at questions. We follow our previous approach of creating a binary column assigning it 0 or 1 based on whether a review contains questions words or question marks and plot a bar chart to see its relation with ratings. Again, in Figure 4.21 we see the opposite pattern, but this time it is not as drastic as the previous one. Rating 1 has once again come second when it should have come first with a significant lead. Therefore, whether a review has questions or not and their corresponding rating is also not representative of all datasets.

Finally, we look at the final feature which is to see if a review has exclamation(s) or not. We follow our usual approach of creating a binary column and plotting the distribution to see

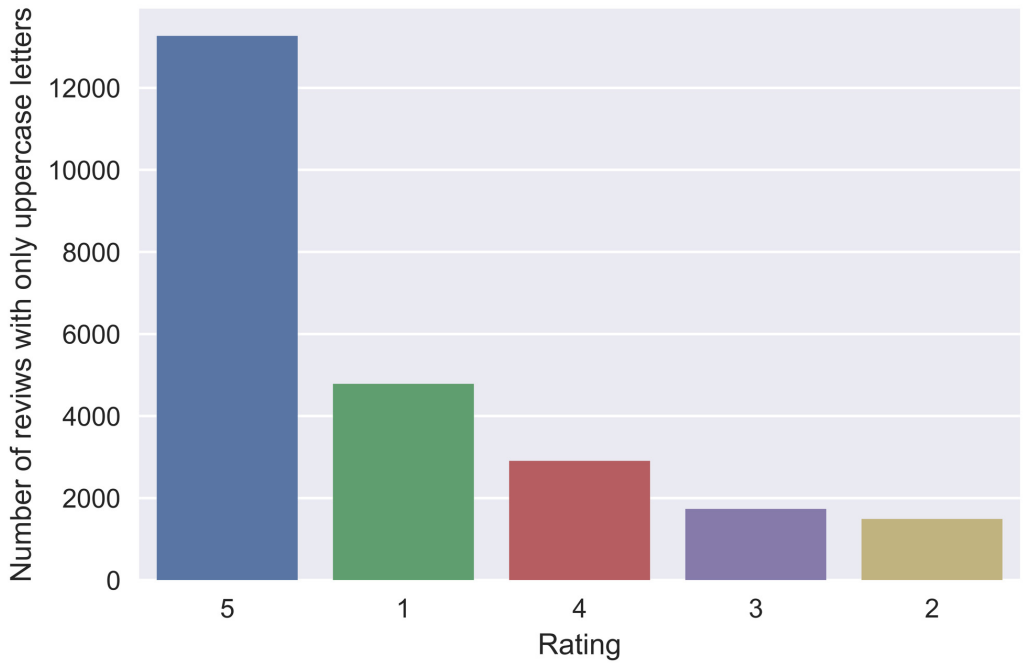

Fig. 4.20 Rating distribution of reviews with all uppercase letters

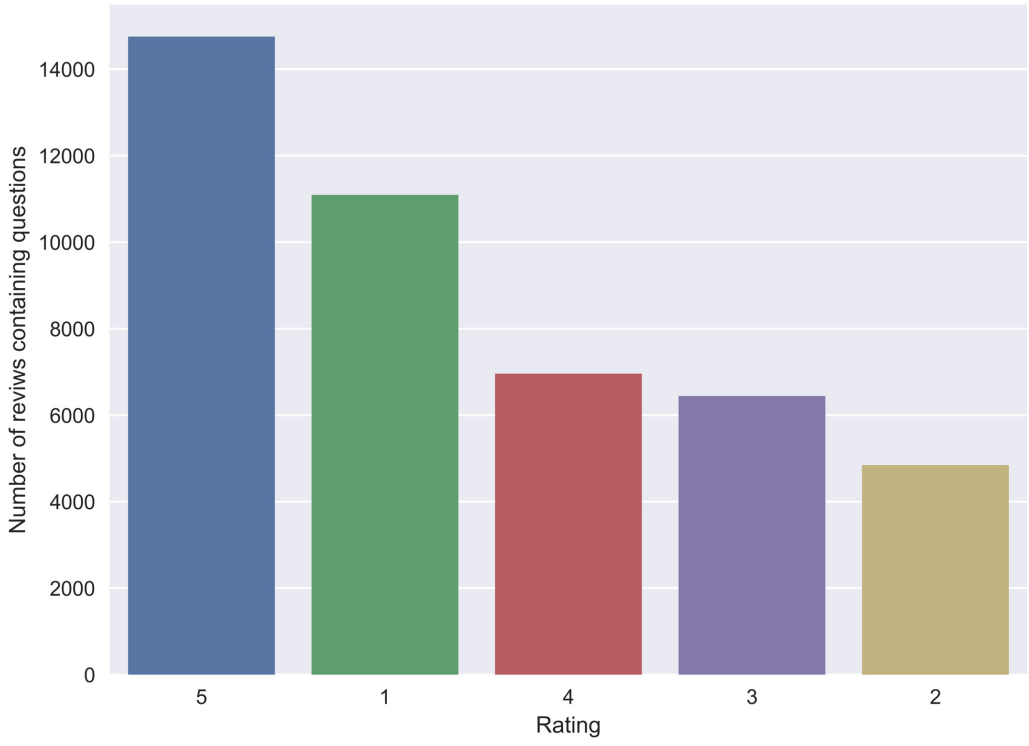

Fig. 4.21 Rating distribution of reviews containing questions

the relation. At last, we see a presumed pattern in Figure 4.22. Reviews with exclamations

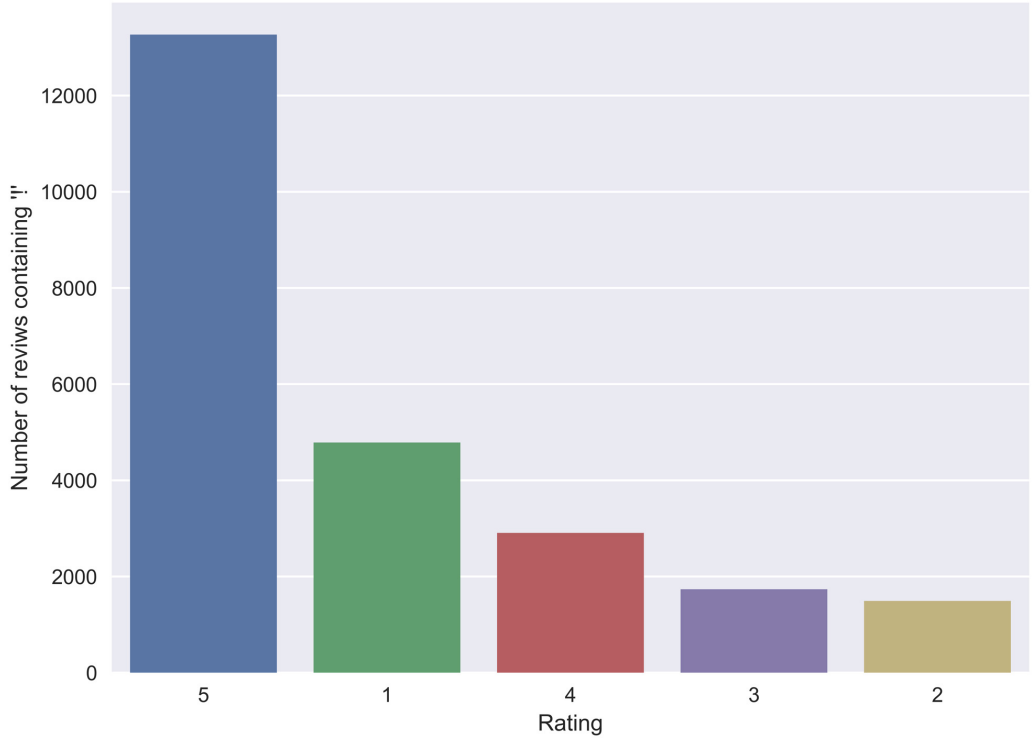

Fig. 4.22 Rating distribution of reviews containing exclamations

do tend to have a 5 star rating in contrast to ones with 1 star. Therefore, this may be a feature that can be extrapolated to other datasets as well.

Chapter 5

Results and Analysis

This chapter explains the target class in our dataset and explores the accuracy of each model created along with the classification report.

5.1 Defining Success

In the paper [23] authored by a Stanford Computer Science student, the formula used by the respective author to define success was the dot product of the average rating and the total number of installs. The average rating being greater than or equal to 4.5 and install count of greater than or equal to 50,000.

$$Success = \begin{cases} 1 & if\ R_{avg} \geq 4.5\ \&\ I_c \geq 50000 \\ 0 & else \end{cases} \tag{5.1}$$

However, the problem with this definition is that there are many apps with a good overall rating and at least 50,000 installs but with a very few raters. And thus the overall rating can be overthrown by some new raters who have a different opinion from the previous ones. Let's take the example of the following app.

App	Installs	Total Rating Count	Avg Rating
Sad Poetry Photo Frames 2018	100000	224	4.5

Table 5.1 Example of a particular App

According to the definition of success defined in [23], this app is more than successful as it has more than 50,000 installs and an average rating of 4.5. However, on a closer look at the total rating count, it is seen that only 224 people have rated the app. And with only another

224 or even less people rating the app poorly say 1 star or 2 star or 3 star, the overall rating of the app would fall drastically. Suddenly the app would become an unsuccessful one.
So we formulated success like this:

$$Success = \begin{cases} 1 & if\ R_{4,5} \geq 75\% * T_r\ \&\ R_{T_r,I_c} \geq 1\%\ \&\ I_c \geq 50000 \\ 0 & else \end{cases} \tag{5.2}$$

where $R_{4,5}$ is the sum of rating 4 and rating 5 in total rating distribution, T_r is the total number of ratings and I_c is the total installs count and R_{T_r,I_c} is the ratio of the total rating count and total installs count multiplied by one hundred.

However, it was much later that we realized that we have excluded apps that are mature and stable and undoubtedly successful but are labeled wrongly as unsuccessful due to the 1% constraint we put in our definition as it can be seen in the Table 5.2 that the total rating count is slightly less than 1% of total installs.

App	Installs	Total Rating Count	Avg Rating
imo free video calls and chat	500000000	4982152	4.3
Maps - Navigate & Explore	1000000000	9633402	4.3
Jewels Star: OZ adventure	5000000	30001	4.5

Table 5.2 Example of a well established app

For apps having millions of installs, one cannot expect to have 1% of raters, so a generic condition does not suffice in determining the success of all apps. So we finally formulate success in the following manner.

$$Success = \begin{cases} 1 & if\ I_c \geq 5000000\ \&\ R_{T_r,I_c} \geq 0.6\%\ \&\ R_{avg} \geq 4 \\ 1 & elseif\ I_c \geq 1000000\ \&\ R_{T_r,I_c} \geq 0.7\%\ \&\ R_{avg} \geq 4.1 \\ 1 & elseif\ I_c \geq 500000\ \&\ R_{T_r,I_c} \geq 0.8\%\ \&\ R_{avg} \geq 4.2 \\ 1 & elseif\ I_c \geq 100000\ \&\ R_{T_r,I_c} \geq 0.9\%\ \&\ R_{avg} \geq 4.3 \\ 1 & elseif\ I_c \geq 50000\ \&\ R_{T_r,I_c} \geq 1\%\ \&\ R_{avg} \geq 4.4 \\ 0 & else \end{cases} \tag{5.3}$$

In this way, we have 2797 apps labeled as 'successful' and 2426 apps labeled as 'not-successful'.

5.2 Results

5.2.1 Random Forests

In our data set of Google Play Store Apps, Random Forest Algorithm gave good competition to the other two classification algorithms XG Boost and K-NN. Random Forest was used for classification as it works well with both categorical and numerical features which are both present in our dataset. This algorithm is very stable in the sense that if a new data point is introduced in the dataset the overall algorithm is not affected much since new data may impact one tree, but it is very hard for it to impact all the trees. In general, Random Forest is fast to train, but quite slow to create predictions once it is trained. A major disadvantage of random forests lies in their complexity. They require much more computational resources, owing to the large number of decision trees joined together. However, due to our dataset being not very large and having around 5300 rows, each row containing the meta-data of an app, the use of large number of trees (estimators) was not a problem. With the default hyper-parameters, random forest was able to produce about 82.23% accuracy in the prediction of the "success" of an app. The maximum accuracy obtained was 85.79% by using a train-test split of 30% that too with just 100 estimators. However, the highest **accuracy of 84.13%** was obtained when the optimal number of estimators was found and used to be 100 with cross-validation of 10 folds. Usually, the more the number of estimators, the better is the accuracy. The trade-off being the time taken to predict when more number of trees/estimators are combined. One of the big problems in machine learning is overfitting, but the chances of overfitting is reduced by taking number of estimators equal to 100, i.e. large number of estimators and by the cross-validation. The sorted order of feature importance obtained with a train-test split method from the random forest classifier is shown in Table 5.3.

Column	Importance
No of Ratings	0.48
Last Updated	0.1
Recent Rating Mean	0.09
Size	0.08
Category (Label Encoded)	0.06
Words in App Name	0.04
Sentiment Mean	0.03
Subjectivity Mean	0.03
Sentiment Median	0.03
Subjectivity Median	0.03
Content Rating (Label Encoded)	0.0
Type (Free – One Hot Encoded)	0.0
Type (Paid – One Hot Encoded)	0.0

Table 5.3 Feature importance according to Random Forest

So, the decision trees that form when running random forest classifier gain the most information (information gain) from the 'Number of Ratings' of the App to split the tree, then 'Recent Rating Mean' and so on.
The confusion matrix with CV=10 folds is shown in Table 5.4:

	Predicted 'Not Successful'	Predicted 'Successful'
Actually 'Not Successful'	1966	460
Actually 'Successful'	369	2428

Table 5.4 Confusion Matrix of Random Forest

The classification report of Random Forest classifier is shown in Table 5.5

Label	Precision	Recall'	F1-Score	Support'
'Not Successful'	0.84	0.81	0.83	2426
'Successful'	0.84	0.87	0.85	2797
Average/Total	0.84	0.84	0.84	5223

Table 5.5 Classification Report of Random Forest

5.2.2 XGBoost Classifier

The XGBoost Python API provides a function that plots decision trees within a trained XGBoost model. With the graphviz library installed, we were able to plot the n_{th} tree in our model, showing the features and feature values for each split as well as the output leaf nodes. Figure 5.1 shows the 100th with default parameters. Here, we can see the max depth is 3.

Fig. 5.1 100th boosted tree with default parameters

Also, the root node in our first tree is the 'Last Updated' feature. The split decisions within each node and the different colors for left and right splits allows for clear understanding. However, the default parameters were not optimally set for the model, given the dataset, as we got an accuracy of 84.22%. Tuning the hyperparameters using the RandomizedSearchCV function with 1000 iterations built into sklearn, we were able to obtain optimal parameter values which are presented in Table 5.8. From those parameters we obtained an **accuracy of 85.09%**. Figure 5.2 shows the 100th boosted tree generated by those parameter values

where we can clearly see the number of level (max depth) has increased by 1 in contrast to the previous tree. Furthermore, Table 5.6 shows the confusion matrix of the XGBoost

Fig. 5.2 100th boosted tree with optimal parameters

model with class labels. Here, we can see that the model predicted success correctly 2503 out of 2797 times (89.5%) and predicted not success 1941 out of 2426 times (80%). From the classification report in Table 5.7, we can see these values being reflected in the recall column. Here, we can see that the model has predicted success and not success with a recall of 0.89 and 0.80 respectively. Table 5.9 shows the weight given to the features in our dataset by this model.

	Predicted 'Not Successful'	Predicted 'Successful'
Actually 'Not Successful'	1941	485
Actually 'Successful'	294	2503

Table 5.6 Confusion Matrix of XGBoost

Label	Precision	Recall'	F1-Score	Support'
'Not Successful'	0.87	0.80	0.83	2426
'Successful'	0.84	0.89	0.87	2797
Average/Total	0.85	0.85	0.85	5223

Table 5.7 Classification Report of XGBoost

Hyperparameter	Value used
min_child_weight	1
gamma	2
subsample	0.4997
colsample_bytree	0.9225
max_depth	4
n_estimators	596
Learning_rate	0.00578

Table 5.8 Hyperparameter values used in XGBoost

Column	Importance
No of Ratings	0.31
Recent Rating Mean	0.2
Last Updated	0.11
Size	0.09
Words in App Name	0.07
Category (Label Encoded)	0.07
Type (one hot encoded Free column)	0.08
Sentiment Median	0.04
Sentiment Mean	0.03
Subjectivity Median	0.03
Content Rating (Label Encoded)	0.0
Type (Free – One Hot Encoded)	0.01
Type (Paid – One Hot Encoded)	0.0

Table 5.9 Feature importance according to XGBoost

5.2.3 K-Nearest Neighbour

The K-NN algorithm was run with varying values of k, where k is the number of neighbours voting on the test instance class. As the optimum value for k varies for different models, the optimum value of k was determined by running a loop from k=10 to k=100. The highest accuracy of **82.44%** was obtained with k=55 and with CV=10 folds. The following Figure 5.3 shows how the optimum value of k was determined.

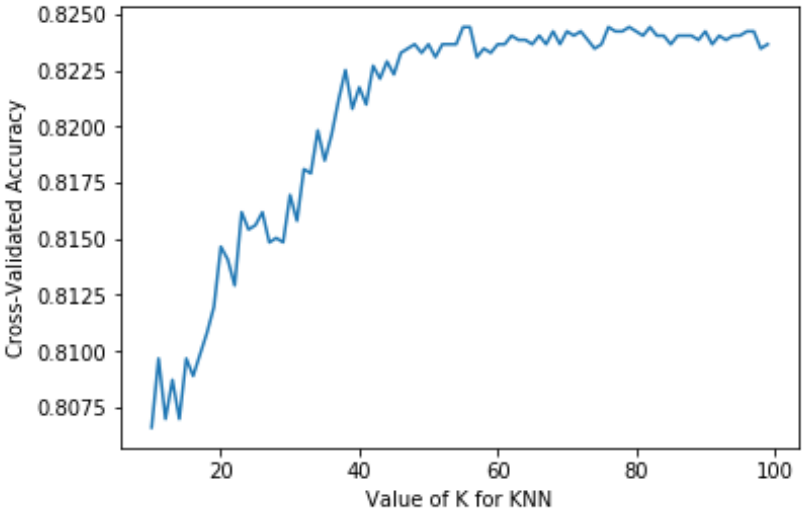

Fig. 5.3 Accuracy for different value of K

Table 5.10 shows the confusion matrix of the K-NN model with class labels. Here, we can see that the model predicted success correctly 2411 out of 2942 times and predicted not successful correctly 1895 out of 2281 times. From the classification report in Table 5.11, we can see these values being reflected in the precision column. Here, we can see that the model has predicted success and not successful with a precision of 0.82 and 0.83 respectively.

	Predicted 'Not Successful'	Predicted 'Successful'
Actually 'Not Successful'	1895	531
Actually 'Successful'	386	2411

Table 5.10 Confusion Matrix of K-NN

Label	Precision	Recall'	F1-Score	Support'
'Not Successful'	0.83	0.78	0.81	2426
'Successful'	0.82	0.86	0.84	2797
Average/Total	0.82	0.82	0.82	5223

Table 5.11 Classification Report of K-NN

5.2.4 Support Vector Machine

At first, simple SVM was fit with the training data. However, due to the non-linearly separable nature of the data, the accuracy obtained was slightly better than a mere guess. Thus, kernel

SVM was run with kernel='poly' and degree=3 to obtain an accuracy of **81.33%** with CV of 10 folds. This means a cubic polynomial separated the two classes better than a linear hyperplane. Higher degrees of polynomial did not output any better accuracy than this, rather the accuracy dropped below 70%.

Table 5.12 shows the confusion matrix of the SVM model with class labels. Here, we can see that the model predicted success correctly 2297 out of 2772 times and predicted not successful correctly 1951 out of 2451 times. From the classification report in Table 5.13, we can see these values being reflected in the precision column. Here, we can see that the model has predicted success and not successful with a precision of 0.83 and 0.80 respectively.

	Predicted 'Not Successful'	Predicted 'Successful'
Actually 'Not Successful'	1951	475
Actually 'Successful'	500	2297

Table 5.12 Confusion Matrix of SVM

Label	Precision	Recall'	F1-Score	Support'
'Not Successful'	0.80	0.80	0.80	2426
'Successful'	0.83	0.82	0.82	2797
Average/Total	0.81	0.81	0.81	5223

Table 5.13 Classification Report of SVM

5.2.5 Voting Ensemble and Overall Results

The majority/hard voting and weighted voting was done on the earlier four classifiers. Majority/Hard Voting is nothing but taking the mode of the estimators. The more occurring prediction is chosen by the voting ensemble. We did weighted voting since XGBoost performed the best among other classifiers and hence was given a higher weight (twice) than the other three classifiers producing an ensemble that results in the highest accuracy of **86.3%**. The overall results are shown in Figure 5.4, all the classifiers were implemented with train-test split of 30% and cross-validation of 10 folds.

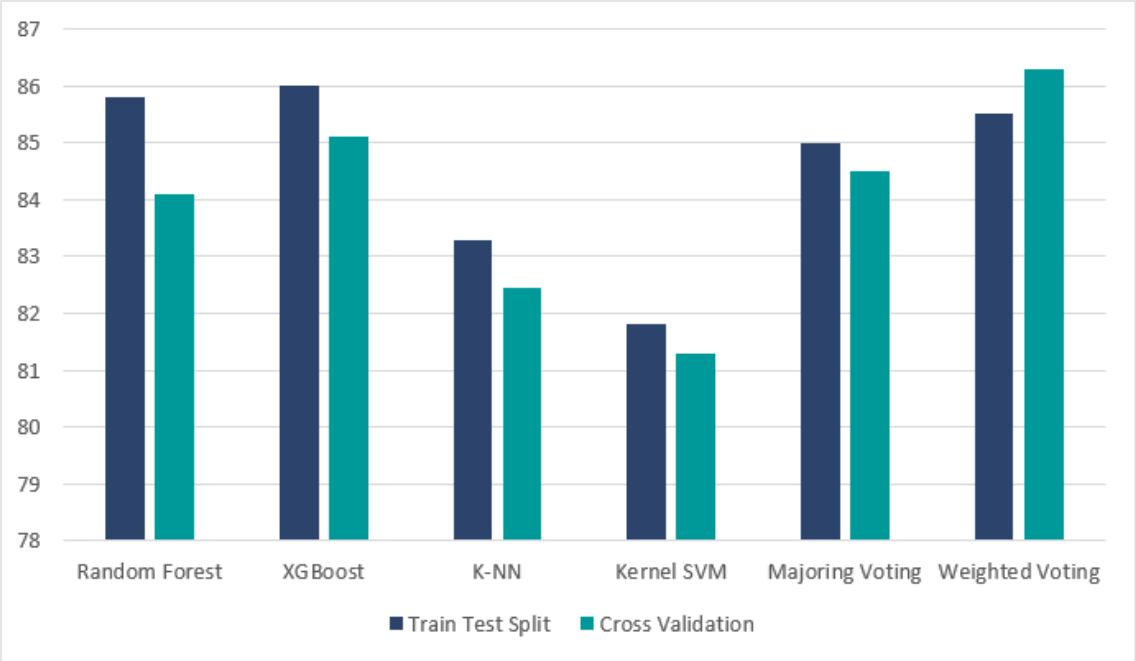

Fig. 5.4 Overall Results

Since almost all the classifiers are able to provide more than 80% accuracy in the prediction of success based on the present features available, this prediction model might be a useful tool for the developers to know if their app, if not successful already, has a possibility of reaching success or not.

5.3 Threats to validity

Let's have a look at the first 10 apps in our dataset in Table 5.14. As obvious as it may seem that the higher the total rating count, the higher the number of installs and hence higher chances of success, it is not quite the real picture as can be seen from the above table. If we only look at the first app "Photo Editor & Candy Camera & Grid & ScrapBook" and the last app shown in the table "Kids Paint Free - Drawing Fun", the former app has installs of only 10000+ but has 683 raters while the latter has higher installs of 50000+ but only 179 raters. So the idea that there is a strong positive correlation between total rating count and number of installs does not hold which helps us to conclude that it is not humanly possible to predict the number of installs given the total rating count. Again to put weight to our conclusion, we can observe the app "Infinite Painter" and the immediately next app "Garden Coloring Book". The former app has total rating count, i.e. total number of people who have the rated

App	Installs	Total Rating Count	Avg Rating
Photo Editor & Candy ...	10000	683	4.2
Coloring book moana	1000000	1780	3.9
U Launcher Lite – FREE Live...	5000000	119937	4.7
Sketch - Draw & Paint	100000000	238616	4.5
Pixel Draw - Number Art...	500000	2127	4.4
Logo Maker For Me ...	100000	1142	4.1
I Smoke Effect Photo ...	500000	1298	4.2
Infinite Painter	1000000	38955	4.1
Garden Coloring Book	1000000	14835	4.4
Kids Paint Free - Drawing...	50000	238616	4.5

Table 5.14 First 10 Apps in our Dataset

the app, of 38,955 and the latter has 14,835. Despite the latter app having less than half the total number of raters, the number of installs is same for both.

.

Chapter 6

Conclusion and Future Work

In this chapter we conclude our findings and suggest related work that could be carried out later.

6.1 Conclusion

The Google Play Store is the largest app market in the world. It generates more than double the downloads of the Apple App Store, but makes only half the money as the App Store. So, we scraped data from the Play Store to conduct research on it. There are a couple of issues with the Play Store that we have identified from exploratory data analysis. A inclusion of a feature that lets users notify developers if they are unsatisfied with an update could go a long into not plummeting an app with low installs to the ground. There is also the problem of rating mismatch on a smaller scale. If this issue could be mitigated, the Play Store would provide a more accurate representation of user sentiment which in turn could help developers make adjustments to their app accordingly. We also defined a success parameter for an app based on the number of installs, distribution of ratings 4 and 5 relative to the overall number of ratings and installs to rating ratio. Using features other than the ones used to create that parameter, we created a model that predicts it with 82.92% accuracy.

6.2 Future Work

In our exploratory data analysis we formed a dictionary of positive words and observed that reviews associated with low star ratings containing those words tend to reflect two things:

- User dissatisfaction with a new update

- Rating mismatch

We were also able to confirm that reviews containing exclamations tend to be associated with a high star rating. We could analyse all the reviews from top to bottom which will allow us to observe certain patterns in the dataset that could be translated into features. Using them we could try and build a model that could potentially identify if a review will result in a rating mismatch, or represent user dissatisfaction. If successful, this model could prevent developers from having an inflated or deflated view of how their app is performing. Also, our initial dataset had only about 5000 apps with around 170000 reviews. We could try the same research with fewer apps, but more reviews per apps, which could paint a different picture of the review patterns of an app.

References

[1] (2018). Google play store: number of apps 2018. [online] https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[2] (2018). Number of daily android app releases worldwide 2018 | statistic. [online] https://www.statista.com/statistics/276703/android-app-releases-worldwide/.

[3] Aralikatte, R., Sridhara, G., Gantayat, N., and Mani, S. (2018). Fault in your stars: an analysis of android app reviews. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 57–66. ACM.

[4] Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

[5] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

[6] Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J., and Sadeh, N. (2013). Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1276–1284. ACM.

[7] Gorman, B. (2017). A kaggle master explains gradient boosting | no free hunch. http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/?fbclid=IwAR1rloE2-AdSIzySTuxkLZPO_fkLrI9vpT--V6OTkpOTrMMTIfKvsHIh68g. (Accessed on 11/25/2018).

[8] Grover, S. (2015). 3 apps that failed (and what they teach us about app marketing). [online] https://blog.placeit.net/apps-fail-teach-us-app-marketing/.

[9] Harman, M., Jia, Y., and Zhang, Y. (2012). App store mining and analysis: Msr for app stores. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 108–111.

[10] Islam, M. R. (2014). Numeric rating of apps on google play store by sentiment analysis on user reviews. In *2014 International Conference on Electrical Engineering and Information Communication Technology*, pages 1–4.

[11] Jong, J. (2011). Predicting rating with sentiment analysis. [online] http://cs229.stanford.edu/proj2011/Jong-PredictingRatingwithSentimentAnalysis.pdf.

[12] Koehrsen, W. (2017). Random forest simple explanation – william koehrsen – medium. https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d?fbclid=IwAR3ZFCD6zUVMePPFIOFNv45VDbuuJv_hqoPziRKELAcjGbYjx5jzEb0ax8M.

[13] Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167.

[14] Luiz, W., Viegas, F., Alencar, R., Mourão, F., Salles, T., Carvalho, D., Gonçalves, M. A., and Rocha, L. (2018). A feature-oriented sentiment rating for mobile app reviews. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1909–1918. International World Wide Web Conferences Steering Committee.

[15] Malik, U. (2018). Implementing svm and kernel svm with python's scikit-learn. https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/?fbclid=IwAR0r_x1BKv7u3A21L2-baw9ayyafG_7jjjoCLAeDzPv6M7oedYbTQBg4VIY.

[16] Mucherino, A., Papajorgji, P. J., and Pardalos, P. M. (2009). *k-Nearest Neighbor Classification*, pages 83–106. Springer New York, New York, NY.

[17] Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics.

[18] Raschka, S. (2014). Ensemblevoteclassifier. [online] http://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/?fbclid=IwAR0bO5WkTAYBcP7dNjgxjrSZrwsmNT2GU0XNEjuqhoj7qyR9Ua8kJVqY304.

[19] Ruiz, I. J. M., Nagappan, M., Adams, B., Berger, T., Dienst, S., and Hassan, A. E. (2016). Examining the rating system used in mobile-app stores. *IEEE Software*, 33(6):86–92.

[20] Saxena, P. (2018). How much money can you make with your app. [online] https://appinventiv.com/blog/how-much-money-can-you-earn-through-your-mobile-app.

[21] Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York.

[22] Sundaram, R. B. (2018). Understanding the math behind the xgboost algorithm. https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/?fbclid=IwAR2WPrUwPFsAVXPSl-wLdJ0Mj3ddu0GDOj2bEhUyqFdqWKcaEVNuZzFYERg. (Accessed on 11/25/2018).

[23] Tuckerman, C. (2014). Predicting mobile application success.

[24] Valentine, A. (2017). 4 mobile app developer success stories. [online] https://blog.proto.io/4-mobile-app-developer-success-stories/.