Abstract

Background: In recent years, many developers have been driven to develop mobile applications by the tremendous growth rate on the mobile apps market. Google Play Store is swamped every day with thousands of new apps with many professionals working on their own or within a team to succeed.

Objective: This study is aimed to use data analysis and machine learning methods to (1) define a new success metric for every application and (2) predict the success of an android application based on its various features for long run.

Methods: Success metric was defined based on the three important features of an applications i.e. ratings, reviews and number of installs. The k-means clustering algorithm was applied by considering ratings, reviews and number of installs to establish three different clusters as successful, moderately successful and unsuccessful. Feature scaling was done to all these three features before the application of k-means clustering algorithm. Further, the classification model was developed using (1) Naïve Bayes; (2) Random Forests and (3) kNN. Feature importance was calculated using mean decreased Gini Index of every features. Overall classification accuracy, sensitivity and specificity was calculated to compare the performance of both the classification models. Training and testing were implemented by conducting 10-fold cross validation.

Results: k-means clustering algorithm was applied to achieve clusters with different values of k from 5-8. We found that k=7 worked best for our dataset considering the datapoint distribution. Naïve Bayes classification model by considering feature selection and random sampling gave us accuracy of 62%. However, accuracy was increased to 97% by replacing the random sampling with 10-fold cross validation.

Conclusion: We propose a model to classify the success metric of an android applications. Cluster analysis gave us the success metric for individual application. These metrics were then classified on the by considering the important features in testing data with an acceptable level of accuracy.

Introduction

Objectives

The objective of this project is to develop a machine learning model to analyze success of android applications based on its features such as ratings, reviews, number of installations etc. The proliferation of smart phones is driving the mobile app stores' rapid development. Presently, Apple's App Store (for iOS users) and Google Play Store (the official Android OS app store) are the two largest global platforms for app distribution.

We have selected Google Play Store to evaluate its features that are available to us for predicting the success of a particular app. In exploratory analysis, this prediction will be used to find out the most successful category of the applications which will assist any company or developer to launch their new application into a category according to their needs, for instance, to make more revenue, to gain more popularity or for advertisement purposes and so on.

Over the past few years, the tremendous growth rate on the mobile application market has motivated many developers to build mobile applications. Google play store gets swamped with thousands of new applications everyday with lots of professional developers working independently or in a team to make them successful. It's competitive for the app developers, the companies that are creating an app to come up with a unique concept that the end users can certainly purchase. Nevertheless, as the mobile industry is growing rapidly, increased competition often leads to increased chances of failure. With immense competition from around the globe, it is

important for a developer to know that he is moving in a right direction. Therefore, as a huge amount of time, effort and money is spent in the project, developers need to do enough study, as the business cannot afford an app failure. Since, most of the play store applications are free of cost, the revenue model is completely inaccessible and unclear as to how application category, in-app advertising, in-app purchases contribute to an applications' success. Ultimately, the quality of an application is typically determined by the number of downloads and user ratings it has received over its lifespan instead of the revenue is generated.

Because of the market growth and the humongous competition, we chose to do research related to this field, so we will carry out an analysis on our data from Google Play Store and will define our own success metrics which we believe will be a major contribution for developers to know their success rate and will be able to decide which attribute to preserve or which to alter in accordance with the current state of their application. Our success model could help developers predict whether their app will be successful in the long run given the current state of the apps that have slow growth rates or are new to the market.

Literature Survey

In paper [2], Authors have suggested a general framework for developers to process, summarize and evaluate user reviews published on App Stores applications. Their framework automatically extracts relevant features from app reviews (e.g., functionality information, bugs, requirements, etc.) and analyzes the sentiment associated with each. The authors noticed that how app store ratings do not affect the overall store rating even after users rate it after reaching a certain level, they also observed that when an app is revised their rating changes version wisely but is not observed in the store rating. Therefore, the researchers came up with the idea of version rating which could be calculated using store ratings available in App Store. Their approach to calculate version ratings is very efficient to allow developers work on the updates.

Furthermore, in paper [3], The authors conducted a study on the rating-review mismatch problem, they reviewed 8600 reviews from 10 common Android apps manually and found that 20 percent of their dataset ratings were inconsistent with the review. To automatically classify reviews whose rating did not match the sentiments expressed in the review, they implemented two conventional machine learning approaches, it concluded different classifiers like Decision tree, Decision stump, Naïve Bayes Classifier and few other algorithms, and other approach based on deep learning techniques.

Another paper with the same approach [4], Islam states that the numerical rating given by users as in the stars has a huge difference from the reviews provided by them, so he has suggested a rating system that will eliminate the ambiguity created by the discrepancy of the rating and the subsequent review by the same user. It has been seen how much we as users are reliant on others opinion when making any decisions so according to him, people install the app based on the rating provided for that particular app, Here Islam describes the division of the problem into two sub-problems. First, the ambiguity and second, the biasness to the users' summarized rating. In order to solve this problem, he suggested a model that will initially carry out sentiment analysis on the user reviews and obtain numeric rating from the polarity. Therefore, the average rating of the sentiment analysis and the star ratings provided by the users will be the final rating which will be shown to users.

Correspondingly in Paper [5], Bin Fu et al. proposed a system named 'WisCom' that is capable of (a) identifying discrepancies in reviews (b) detecting reasons why users like or hate an app, and offering an interactive, zoomable view of how reviews of users change over time; and (c) providing valuable insights into the broader app market, revealing the major concerns and desires of users of different app types. They extended their investigation to provide a valuable insight to help a mobile app market operator like Google as well as individual app developers and end users. It introduces techniques for summarizing and mining these reviews for different parties such as End users can use these summaries to pick the apps with the best user experience without having to read any reviews, App developers could use these summaries to recognize why end users love or hate their apps and rival apps to improve their quality, and Market players like Google Play can use their techniques to automatically detect

problematic apps to ensure safe and high-quality content and drive the market to greater prosperity. The study of market segments and patterns could help all of these and other stakeholders. More precisely, it analyzes on three scales, firstly on a single review (micro-level) conducted using a regularized linear regression model, secondly on reviews of each user (meso-level) performing LDA as well as performing subject analysis on different time segments to determine how review changes over time, and finally on all applications on the market (macro-level). They outlined all their three level strategies and identified the advantages that end users, developers and the entire mobile ecosystem would benefit from.

In Paper [1], Tuckerman scarped Google Play store data to extract as many features as possible using the 'scrappy' web crawler and used the data to train three models to predict the application's success. Revenue should be the key feature to predict an app's success metrics but as it is not publicly accessible, He used the number of installs and average user rating, trained Linear model to categorize whether an application will be efficient and also used linear regression to predict the to predict the average system rating. The key component research was performed to concentrate on variation and create strong data set patterns using GLM and Linear regression models inputs. Using these models, Tuckerman concluded, that about 35% of the total active applications has the word 'photo' in the description and about 31% has the word 'share'. Using these models, developers may refer to the application genres that are mostly accepted by end users. A developer will be able to find an application's economic success from the given success metric. This paper served as our reference document, we got the idea of how the number of downloads and average rating could play an important role in determining success metrics and identifying every application's success rate.

We also looked at papers from different domains but similar work to better understand how reviews vary from one domain to another in order to justify the reasons for choosing each attribute in our dataset will play an important role in predicting success using the success metrics proposed in [1]. Pang and Lee worked on movie reviews in paper [6], transforming their extracted rating into one of three categories: neutral, positive, negative. Using movie reviews as data, they found that standard machine learning techniques ultimately outperform human produced baselines. For the word list, two computer science graduate students were asked to volunteer in a review for the movie to pick good indicator words for positive and negative feeling. From the words they first found levels of accuracy for the overall negative and positive feeling and later made a more condensed list of words using seven terms respectively for positive and negative. They implemented three algorithms: Support Vector Machine, Naïve Bayes, and Maximum Entropy (MaxEnt or ME, in short), they attempted eight different combinations of features with all three algorithms and showed the comparative result, where SVM performed the best and Naïve Bayes the worst. They noticed that adding POS (Parts of Speech) tags improves Naïve Bayes ' accuracy marginally, but it decreases for SVM and remains unchanged for MaxEnt. As movie review is generally different from app review, their work is quite different from ours.

For example, movie reviews are generally longer than app review, according to Bin Fu et al. [5] (average 71 characters per comment noted in [5]).

Accomplishment

In this project we defined our own success metric which will give us the level of how successful or hit the application is in our android market. Based on the defined success metric which was taken as the class label, we implemented a classification model that will help to determine the success metric of an application. This will help the developer to update the application in such a way that the applications shall generate maximum revenue in the long run. We also analyzed the android market data to find the best category based on number of installations and ratings that a new developer to go for before developing a new application.

Data

We have downloaded the Google Store Apps dataset from Kaggle [7] which has about 264k applications. After filtering out the duplicates, this dataset has around 242k unique application names. In the below section, the data has been described in detailed.

A sample data instances are broken down into two tables for better visualization.

App Name	Category	Rating	Review	Installs	Size
Door Dash	FOOD_AND_DRINK	4.5842	305034	5,000,000+	Varies with device
Peapod	SHOPPING	3.656329	1967	10,000,000+	16M
FreshDirect	LIFESTYLE	3.213528	754	100,000+	39M
Chick-fil-A	FOOD_AND_DRINK	4.374691	52526	5,000,000+	19M
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
Pot Shot	GAME_CASUAL	4.441585	4896	100,000+	44M
Ganster Tycoon	GAME_SIMULATION	4.612342	632	50,000+	44M

Table 2.1 Instances in raw format with first 6 features

App Name	Price	Content Rating	Last Updated	Minimum Version	Latest Version
Door Dash	0	Everyone	29-Mar-19	Varies with device	Varies with device
Peapod	0	Everyone	20-Sep-18	5.0 and up	2.2.0
FreshDirect	0	Everyone	27-Mar-19	5.0 and up	6.7
Chick-fil-A	0	Everyone	21-Mar-19	5.0 and up	6.1.0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
Pot Shot	0	Everyone	9-Mar-19	4.1 and up	1.1.9
Ganster Tycoon	$3.50	Teen	1-Apr-19	4.1 and up	1.1.0

Table 2.2 Instances in raw format with remaining features

The column header of the dataset from Table 2.1 and Table 2.2 are briefly explained in the following Table 2.3.

Attributes	Description
App Name	Name of the application
Category	Category or genre e.g. Education, Food and Drinks
Rating	Average rating of the application
Reviews	Number of reviews
Installs	Number of installs
Size	Size of the application in k (kilobyte), M (megabyte)
Price	Price of the application in $ (dollars)
Content Rating	Intended audience or age group targeted

Last Updated	Last updated date
Minimum Version	Minimum android version required to install application
Latest Version	Current version of the application

Table 2.3 Description of the features

In our dataset, we have the missing/blank values, special character, and faulty random values in most of the attributes such as App Name, Category, Rating, Reviews, Installs, Size, Minimum Version, and Latest Version.

As you can see from dataset, we don't have any predefined class information, as a result we defined our own class label that will determine the success level of each application. So, our classification model predicts the success class label for each of the applications. While modeling the dataset, all the features mentioned above in Table 2.3 are used in determining the class label for each application.

Methods and Tools

Preprocessing

Since we have downloaded the dataset from Kaggle, it consists of many missing values, special characters or faulty random values. This raw data will have to be pre-processed to turn it into a valuable information for deriving the accurate machine learning model. In Data cleaning, we had approximately 13k instances with missing values which were handled by deleting those entries. We do have the noisy data (data entry errors) in some of the attributes which were also handled manually by deleting those entries. In Data Transformation, we did feature scaling on some of the attributes such as Number of Installs, and Number of Reviews. Characters such as '+' and ',' were removed from number of installs columns. All the size in KBs was converted into MBs in size column. A new column named as 'Type' was introduced by us for data analysis. Last updated date feature was converted into date format. There were few values as 'Varies with device' in column Minimum android version and Size which was replaced by NA. These all processed attributes will be used to determine the future success metrics for applications according to category.

Clustering

In basic terms, the objective of clustering is to find different groups within the elements in the data. To do so, clustering algorithms find the structure in the data so that elements of the same cluster (or group) are more like each other than to those from different clusters.

In our dataset as we don't have any class label for success matrix, we applied clustering to define our own success matrix. The anticipated algorithm that we used is k-means clustering.

K-Means algorithms are extremely easy to implement and very efficient computationally speaking. The k-Means algorithms aim to find and group in classes the data points that have high similarity between them. In the terms of the algorithm, this similarity is understood as the opposite of the distance between datapoints. The closer the data points are, the more similar and more likely to belong to the same cluster they will be. On our dataset, k-means clustering is applied by considering the features no of installs, Ratings and no of reviews as these three columns gave us the highest feature importance. No of installs and reviews were scaled based on Ratings range before implementing clustering. We have tried different values of k such as k=5,6,7,8 which gave us different set of clusters. We finalised the value of k=7 as that gave us the good distribution of data points among all clusters, which did not lead to any class imbalance. Also, the combinations of the features selected for clustering worked well with k=7. We

found average value of each feature value after clustering with different values of k such as k=5,6,7,8 respectively to find the best set of clusters.

Since we found our class label i.e. success metric using clustering, we trained the model by a supervised learning approach. Training supervised model consists of a variety of training examples to educate the user about a feature from marked training data. For instance, a pair of input entities (typically a vector) and a desired output value in supervised learning. A supervised learning algorithm analyzes the training data and constructs a method which can be used to generate new examples. Two main sections of supervised learning are Classification and Regression.

Classification

In classification, the goal is to assign an observation to a class (or label) from a finite set of classes That is, categorical variables are the results. Applications include spam filters, recommendation systems for marketing, and identification of image and language. The probability of a person suffering from depression within each year is a classification problem and the possible classes may be true and false.

In our dataset, we will be using classification methods to determine each application's success metric. The anticipated algorithms that we used are as follows:

1. Random Forest Algorithm

Random forest is a tree-based algorithm which involves the construction of several trees (decision trees) and then the integration of their output to enhance the model's generalization. The way trees are coupled is known as an ensemble method. Assembly is only a mixture of weak students (single trees) which produces a strong learner.

Definition: A random forest is a classifier consisting of a collection of tree structured classifiers h(x, Θk), k = 1,... where the Θk are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input. In our dataset, we defined our own class label which was the success metric. Thus, Random forest will help to merge these multiple decision trees so that a more accurate and stable classification can be achieved. Before applying random forest, we split the data into ratio of 60:40. We trained the random forest model by considering 60% of the dataset and classified the value for test data using the model. We achieved an accuracy of 95% by splitting the data into 60:40.

2. Naive Bayes Algorithm

Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posterior decision rule in a Bayesian setting. It can also be represented using a very simple Bayesian network.

For our dataset we implemented Naïve Bayes with random sampling and 10-fold cross validation. For building model with Naïve Bayes, we calculated the mean Gini Index on the training sample to get the important features. Two models were generated by using random sampling of 60:40 of the training dataset and by 10-fold cross-validation. The generated model was used with the testing data sample to classify the value of success metric of individual application. By using the random splitting (60:40) we achieved an accuracy of 81.86% and by using cross validation accuracy was reached to 84.99%.

3. K-Nearest Neighbour

KNN (K-Nearest Neighbors) is one of many (supervised learning) algorithms used in data mining and machine learning, it's a classifier algorithm where the learning is based "how similar" is a data (a vector) from other. It is called Lazy algorithm because it does not need any training data points for model generation. All training data is used in the testing phase which makes training faster and testing phase slower and costlier.

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 as k=11. The highest accuracy of 98.85% was obtained with k=11.

Validation

Validation of information is a tool of ensuring the reliability and performance of your results, usually done before importation and storage. Also, a type of data cleaning can be called. Software verification guarantees that your software is accurate (no empty or zero value), special and that the set of values are compatible with what you anticipate. The three basic steps in data validation are determining the data sample, validating the sample dataset and validating the data format. The method that we will be using for data validation is as follows:

1. 10-fold Cross Validation

Cross validation is a method used for testing machine learning models on a set of limited data. The term k refers to the number of categories that a given sample dataset can be partitioned into. This strategy includes separating the set of observations into roughly 10 categories or folds. The first plug is regarded as verification, and the remaining 9 folds are modified to the system. Cross-validation is primarily used to estimate the strengths of a machine learning model for unidentified information in applied algorithm learning. It involves a minimal sample to evaluate how the system is usually expected to perform when predicting data which are not used during model learning. This method will lead us to generate a less biased model to prevent the overfitting which will eventually lead to a high variance model. In short, each discovery in the sample of information is assigned to a category and stays for the duration of the process in the community. It allows that test the ability to be used in the set time and to train the 9 method.

Tools

The language that we will be using is R programming for which the environment used will be R studio. The version of R that will be used for statistical computing and data analysis is R-3.6.1. Below is the list of libraries that might be required during analysis.

Caret: To streamline the model building and evaluation process, as well as feature selection and other techniques.

Lubridate: Tools that make working with dates and times easier.

randomForest: Random forest methods from machine learning.

Dplyr: For all the manipulation related to dataset.

e1071: Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support vector machines, shortest path computation.

Stringr: Simple and consistent wrappers for common string operations.

Magrittr: Offers a set of operations that will promote semantics which leads to code improvement.

Highcharter: Charting library.

Rpart: Can be used for dataset partitioning process.

Tidyverse: used for data manipulation, exploration and visualization that share a common design.

Xts: Provide for uniform handling of R's different time-based data classes.

caretEnsemble: It is the preferred way to construct list of caret models in this package, as it will ensure the resampling indexes are identical across all models.

psych: A package for personality, psychometric, and psychological research.

Amelia: Multiple Imputation of Incomplete Multivariate Data.

Mice: Creating multiple imputations as compared to a single imputation (such as mean) takes care of uncertainty in missing values.

GGally: 'ggplot2' is a plotting system based on the grammar of graphics.

Normalr: provides an optimal transformation for non-normal variables.

Cluster: used for clustering solutions.

HSAUR: A Handbook of Statistical Analyses Using R.

Fpc: consists of Flexible Procedures for Clustering

klaR: used for Classification and Visualization. Miscellaneous functions for classification

MASS: Functions and datasets to support Venables and Ripley, ``Modern Applied Statistics with S''

Class: Various functions for classification, including k-nearest neighbour, Learning Vector Quantization and Self-Organizing Maps.

Results

Clustering

Clustering of the dataset was implemented by considering the features as no of installations, reviews and ratings. We tried clustering with 4 different values of k i.e. k=5,6,7,8. K-means clustering algorithm showed increase in average values of ratings, reviews and no of installs for higher values of k.

For k=5, it is observed that cluster 4 showed the maximum average value which derives the most successful cluster follow by cluster 3, 2,5 and 1. Hence the level of success metric for 5 clusters is 4,3,2,5,1 which is from most successful to unsuccessful cluster. Figure 4.1 shows the plot for average values of ratings, reviews and no of installs for 5 clusters.

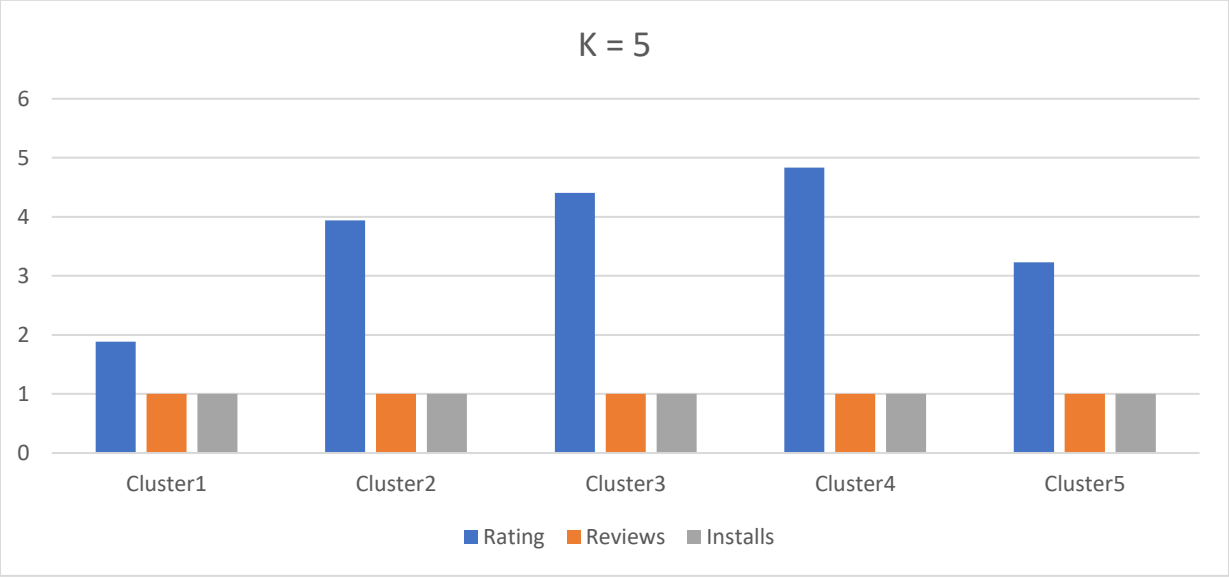

Fig.4.1 Average feature values for k=5

For k=6, it is observed that cluster 4 showed the maximum average value which derives the most successful cluster follow by cluster 3, 2,5,6 and 1. Hence the level of success metric for 6 clusters is 4,3,2,5,6,1 which is from most successful to unsuccessful cluster. Figure 4.2 shows the plot for average values of ratings, reviews and no of installs for 6 clusters.

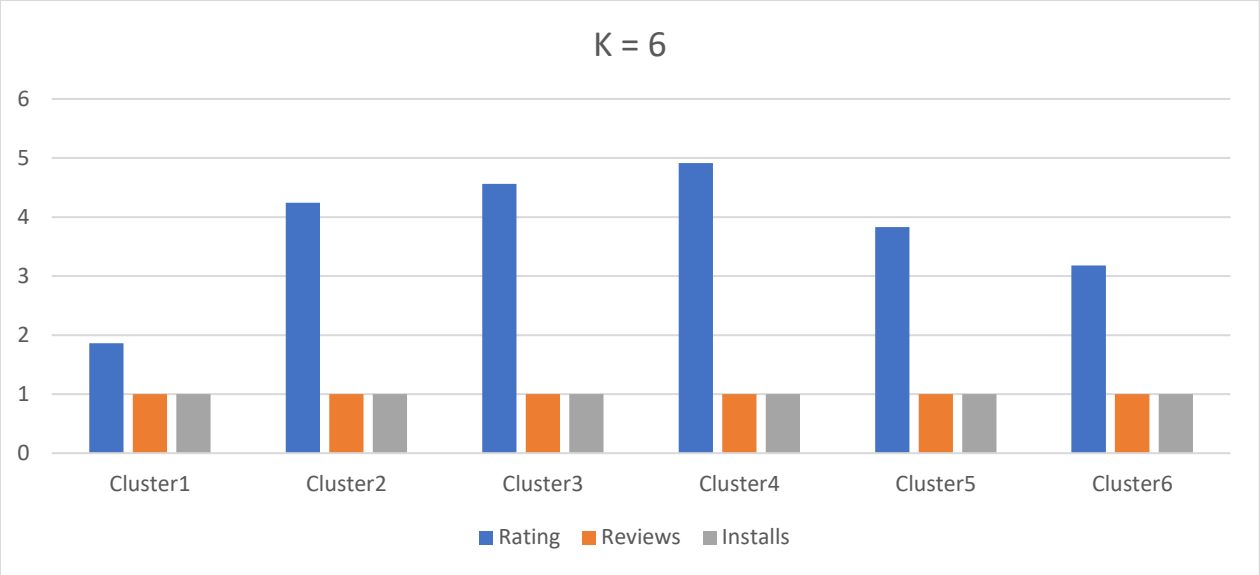

Fig.4.2 Average feature values for k=6

For k=7, it is observed that cluster 4 showed the maximum average value which derives the most successful cluster follow by cluster 3, 2,5,7,6 and 1. Hence the level of success metric for 7 clusters is 4,3,2,5,7,6,1 which is from most successful to unsuccessful cluster. Figure 4.3 shows the plot for average values of ratings, reviews and no of installs for 7 clusters.

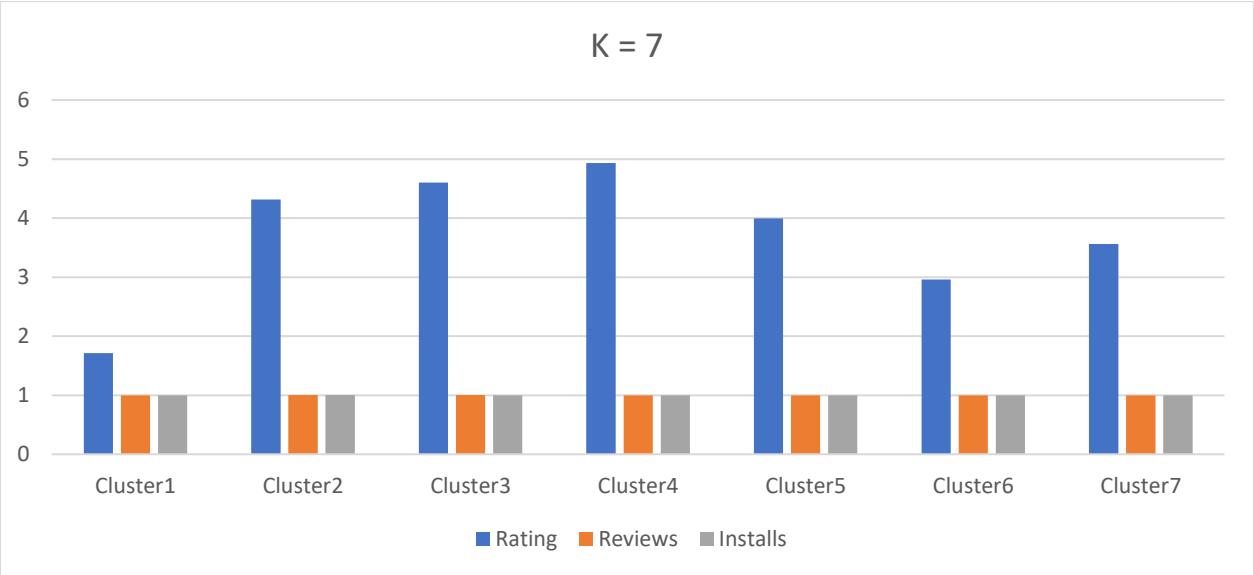

Fig.4.3 Average feature values for k=7

For k=7, it is observed that cluster 4 showed the maximum average value which derives the most successful cluster follow by cluster 3, 2,8,5,7,6 and 1. Hence the level of success metric for 8 clusters is 4,3,2,8,5,7,6,1 which is from most successful to unsuccessful cluster. Figure 4.4 shows the plot for average values of ratings, reviews and no of installs for 8 clusters.

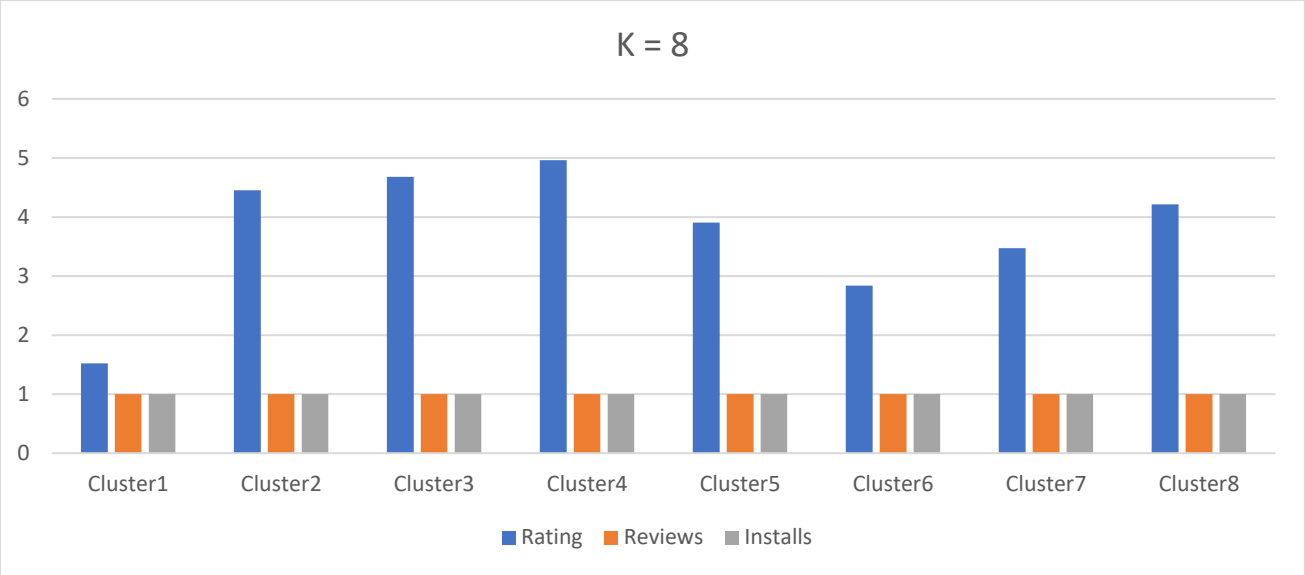

Fig.4.4 Average feature values for k=8

The summarized graph shows the average value of ratings, reviews and no of installs for all values of k and every k cluster. The summarized plot for all implemented values of k is shown in figure 4.5.

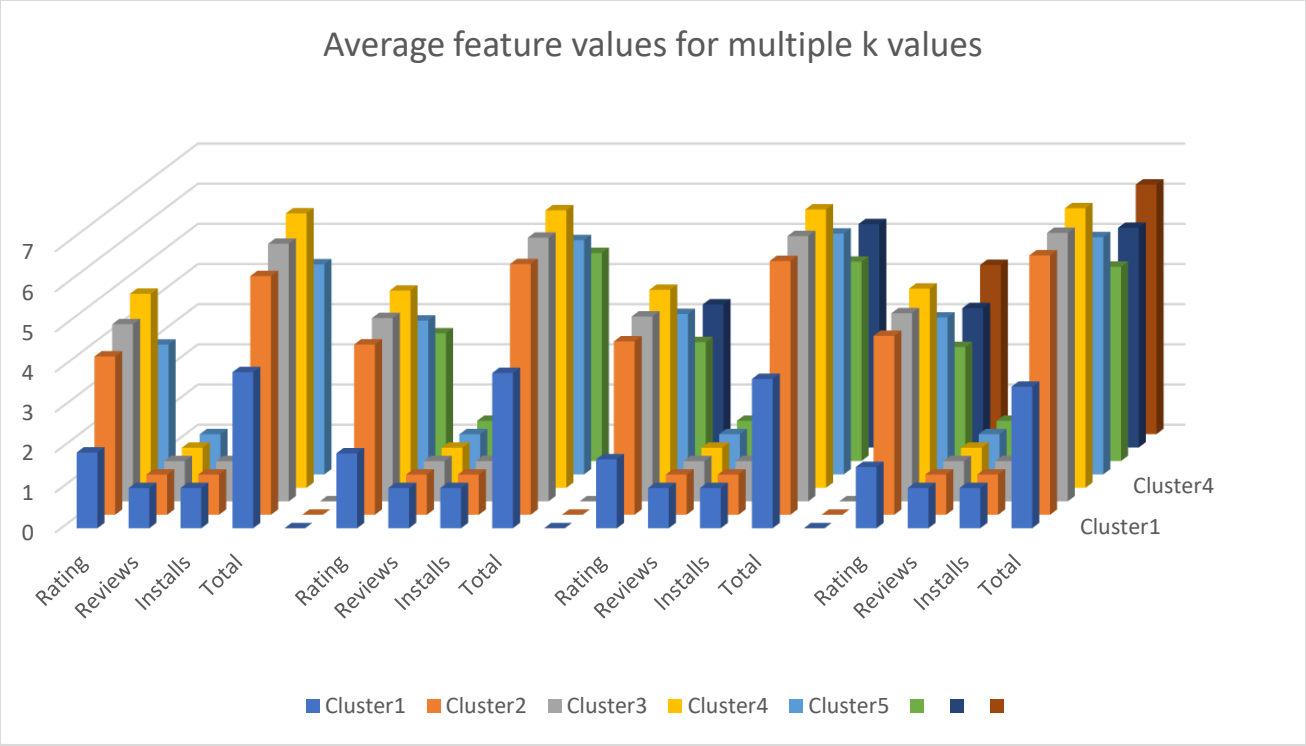

Fig.4.5 Average feature values for k=5,6,7,8

The summarized plot for total average values of ratings, reviews and no of installs is shown in figure 4.6.

The plot shows the total value of features for different values of k by considering every k cluster.

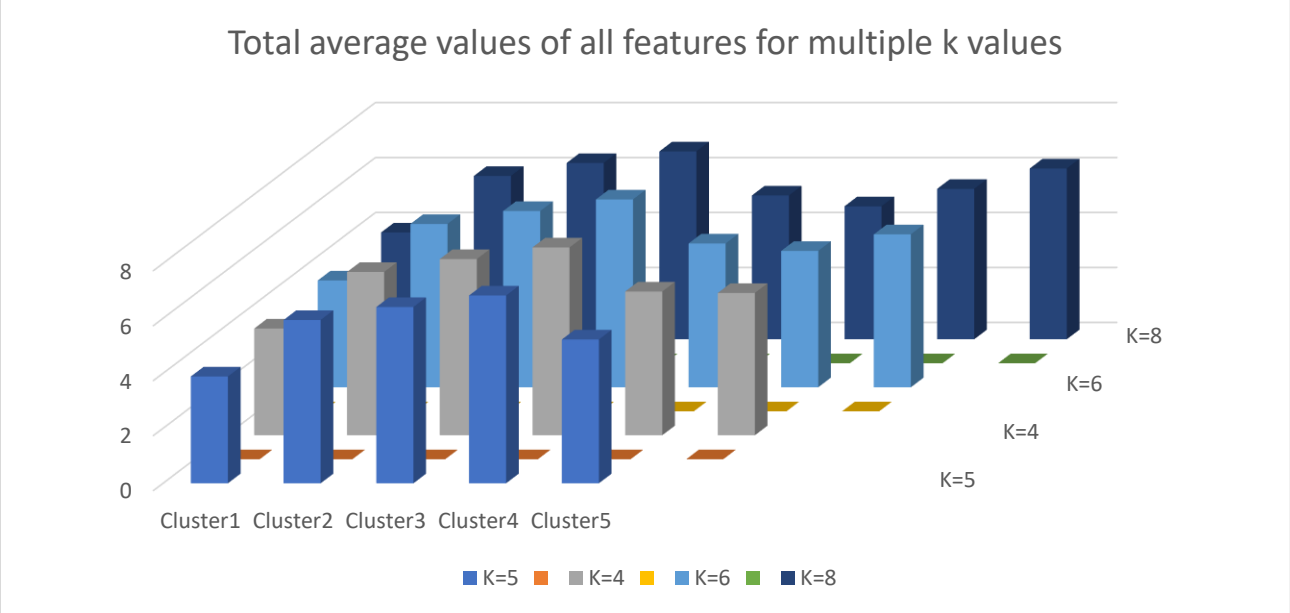

Fig.4.6 Total Average feature values for k=5,6,7,8

Thus, the summarized plots show that for k=7 all the datapoints are distributed evenly compared to other k values. Hence, choosing no of clusters as 7 will not lead to the class imbalance issue during building the classification model. Also, the total average values are higher for k=7 which will lead us to a most successful group of applications.

Thus, we implemented the classification models by considering 7 different values of our class label i.e. success metric.

Naïve Bayes

We tried to implement Naïve Bayes with k=3 and k=7 for comparison of accuracy results. We implemented Naïve Bayes with random sampling of 60:40 ratio and 10-fold cross validation of data. Accuracy of 81.86% and 84.99% was achieved by random sampling and 10-fold cross validation respectively. Table 4.1 shows the confusion matrix with random sampling of dataset. Table 4.2 shows the confusion matrix by using 10-fold cross validation on dataset.

Prediction	1	2	3	4	5	6	7
1	8613	6	8	0	3221	4	0
2	0	5820	1700	4	0	0	37
3	0	72	10515	0	1026	0	0
4	0	3	0	2120	0	4	671
5	28	0	258	0	9302	0	0
6	0	0	6	299	2	693	2
7	0	2476	15	29	0	0	4023

Table 4.1 Confusion matrix for Naïve Bayes

Prediction	1	2	3	4	5	6	7
1	17148	0	0	0	0	0	0
2	0	16935	0	0	0	0	0
3	0	0	25283	0	1	0	0
4	0	0	0	4869	0	3	1
5	0	0	1	0	26951	0	0
6	0	0	0	0	0	1370	0
7	0	1	0	0	0	0	9350

Table 4.2 Confusion matrix for Naïve Bayes with 10-fold cross validation

Random Forest

We built random forest model by considering 7 values of our class label and random sampling of dataset into 60:40 ratio. There was overfitting issue observed in our model which lead us to a very high accuracy of the model. Accuracy of 98% was observed after predicting the values of testing data using random forest model. Table 4.3 shows the confusion matrix for random forest model.

Prediction	1	2	3	4	5	6	7
1	8472	0	0	0	0	0	0
2	0	8582	0	0	0	0	0
3	0	0	12770	0	0	0	0
4	0	0	0	2305	0	1	0
5	0	0	0	0	13444	0	0
6	0	0	0	0	0	715	0
7	0	0	0	0	0	0	4668

Table 4.3 Confusion matrix for random forest

k-Nearest Neighbor

We built kNN model by considering 7 values of our class label and random sampling of dataset into 60:40 ratio. As kNN only takes numeric values, we were unable to pass the nominal categorial features to the model while training. Hence, a high accuracy of 98.85% was observed after predicting the values of testing data using kNN model. Table 4.4 shows the confusion matrix for kNN model.

Prediction	1	2	3	4	5	6	7
1	8323	17	0	0	0	0	0
2	95	8279	31	1	0	0	0
3	10	109	12555	131	9	0	0
4	0	0	0	2210	0	3	0
5	0	1	4	11	13640	5	70
6	0	0	0	0	0	620	0
7	0	0	0	0	0	89	4744

Table 4.4 Confusion matrix for kNN model

Feature Importance

Feature importance was determined by calculating the mean Gini decrease of the features by using random forest library. Table 4.5 shows the list of important features with its respective mean gini decrease values.

Features	MeanDecreaseGini
Category	1382.87263
Rating	138768.69644
Reviews	1442.95171
Installs	1669.50528
Size	162.76144
Content.Rating	22.20774
Last.Updated	294.11663
Type	23.12899
Min.Android.ver	104.86199

Table 4.5 Mean Gini Decrease of all Attributes

Classification Results

Thus, for our dataset Naïve Bayes model performed the best with cross validation by achieving the accuracy of 84.99%. Table 4.6 shows the comparative analysis of the classification results by considering the 3 classification models.

Algorithm	Accuracy	Sensitivity	Specificity
Naïve Bayes	81.86%	84.19%	96.95%
Naïve Bayes with cross validation	84.99%	87.25%	97.49%
Random forest	99.99%	99.99%	1.00%
K nearest neighbor	98.85%	96.54%	99.79%

Table 4.6 Comparative Analysis

Exploratory Data Analysis

Figure 4.7 shows the most popular category, by number of installs:

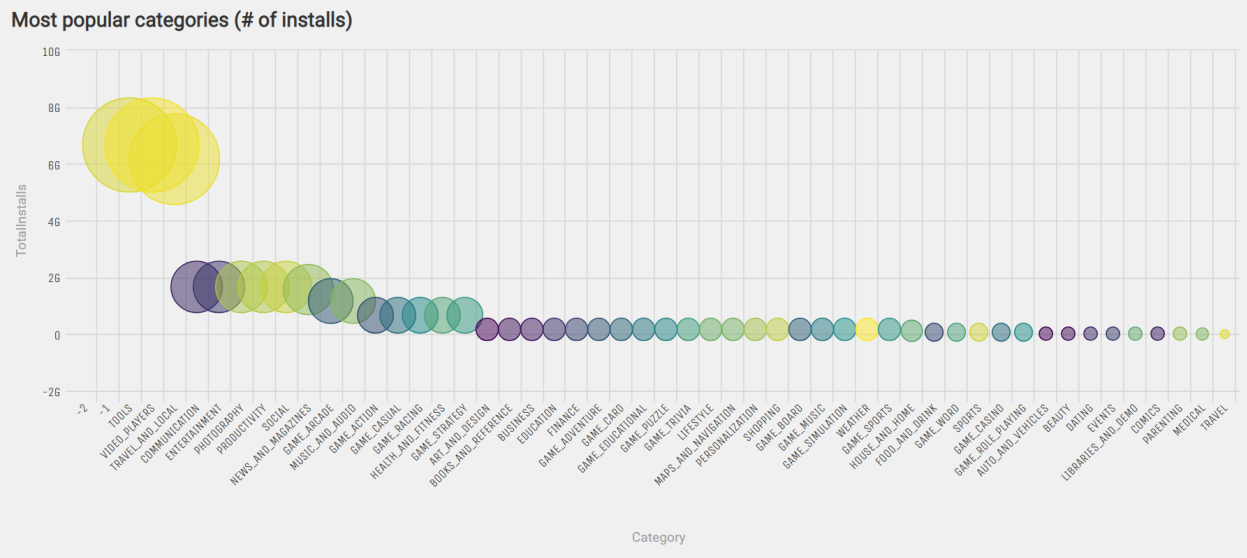

Fig.4.7 Popular Category vs no of installs

Figure 4.8 shows the most popular category, by average rating:

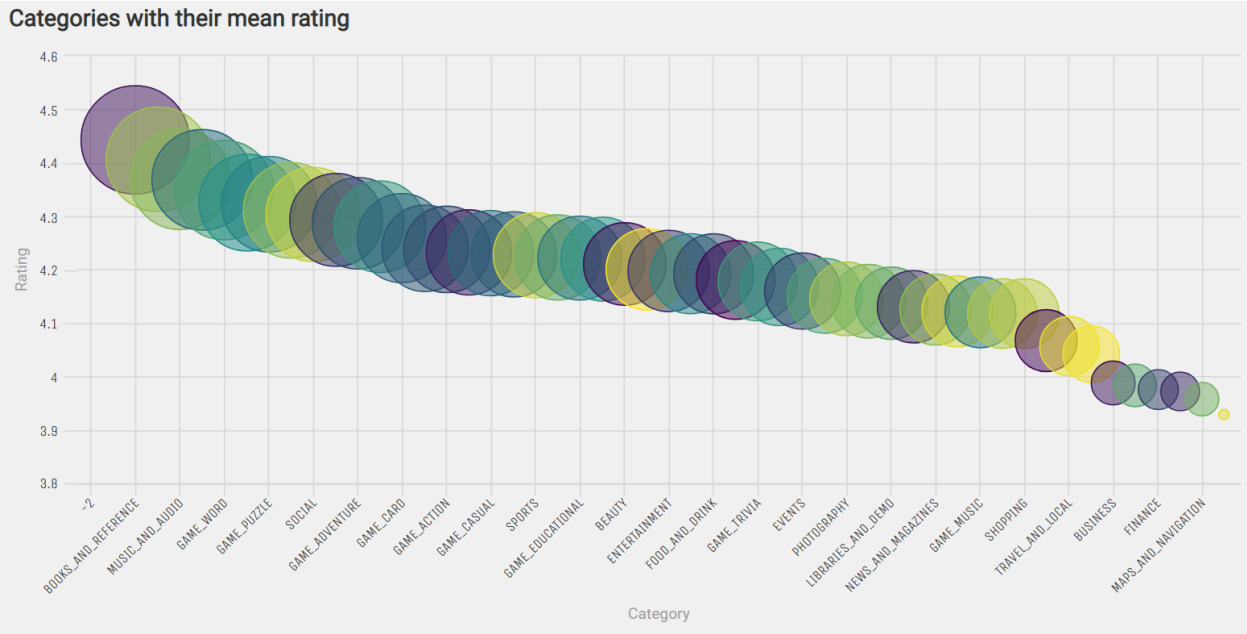

Fig.4.8 Popular Category vs average ratings

From above 2 graphs, we can say that if an application is having the highest rating does not implies that it is popular or should have higher number of installations.

Figure 4.9 shows the application size distribution in MB.

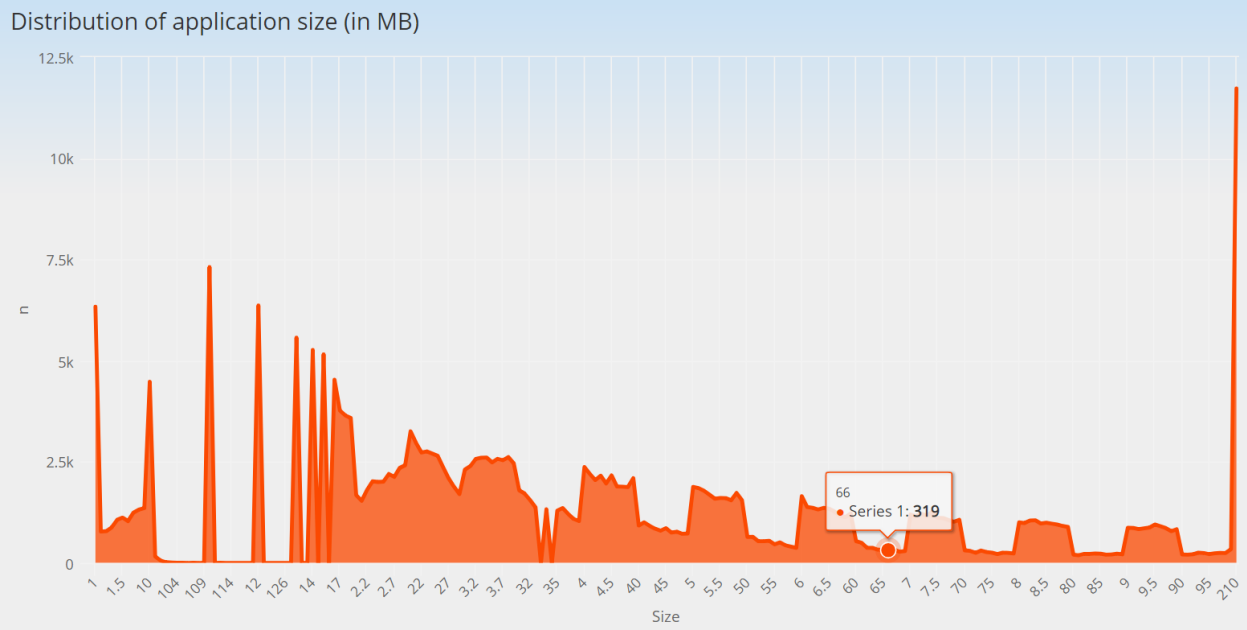

Fig.4.9 Application size distribution

In this dataset, most of the applications have 210MB size.

Applications in this dataset range from 0 to 10^{9} installations. If we divide this range by groups of 10k, we can see the distribution shown in figure 4.10.

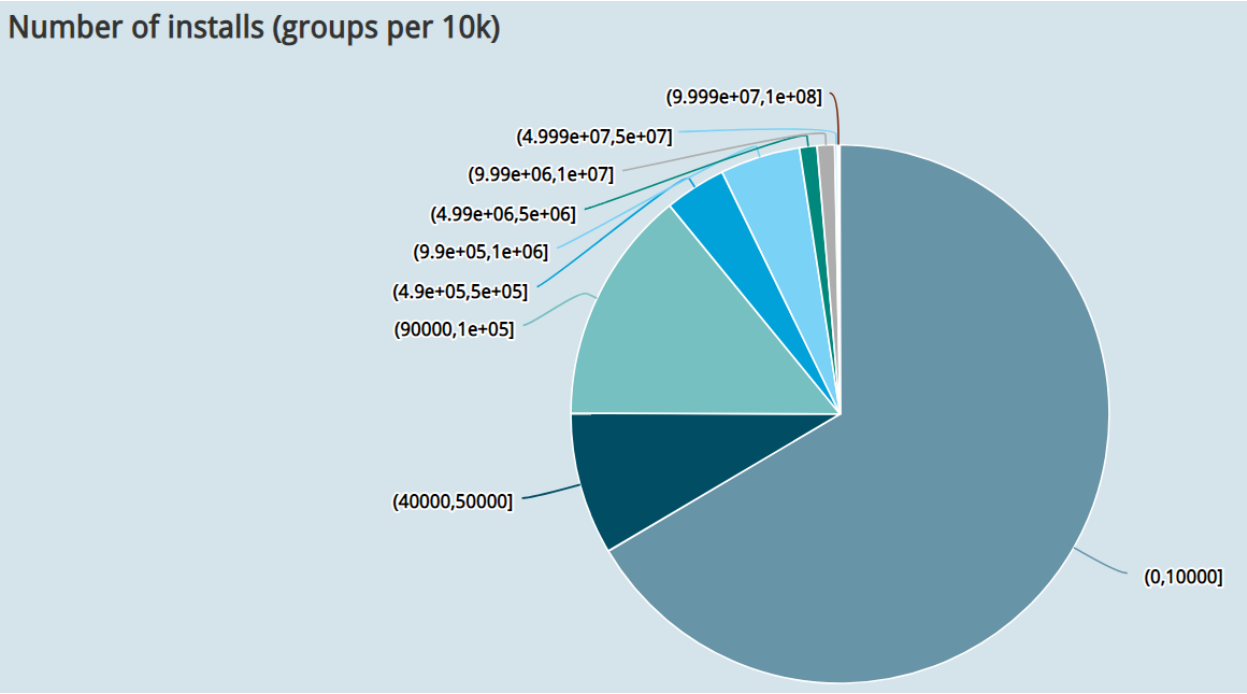

Fig.4.10 No of installs per 10k group

The largest group is made of applications with up to 10k downloads. Over 75% of the applications had less than 50k installs.

Figure 4.11 shows the total split between free and paid applications.

Fig.4.11 Ratio of paid vs free apps

It is observed that 4% of applications are paid.

Figure 4.12 shows how this ratio changes per category.

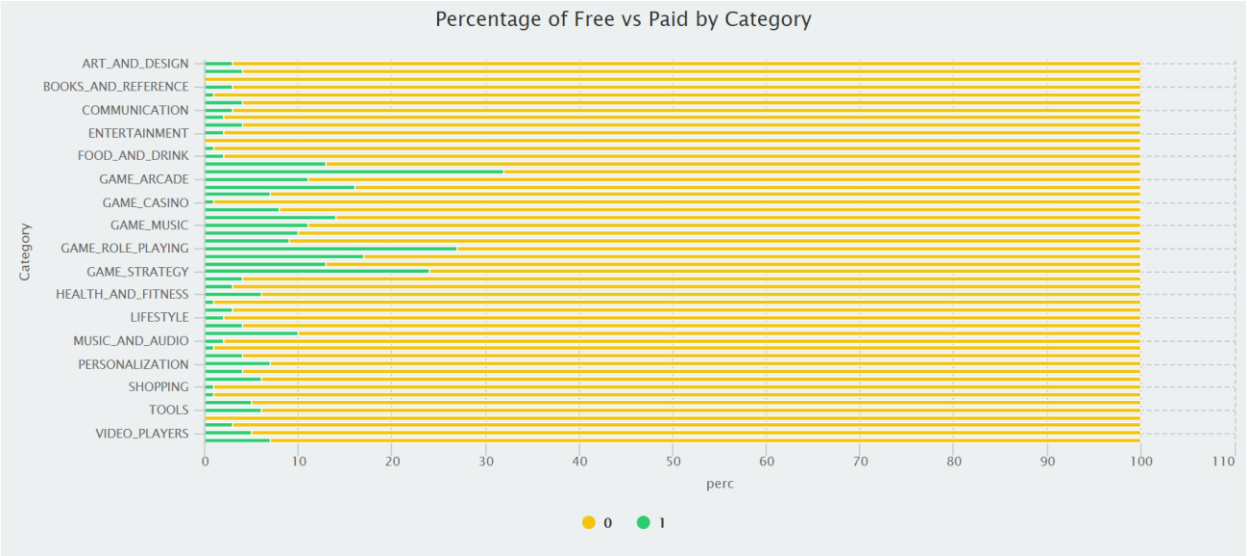

Fig.4.12 Percentage of free vs paid apps

It is observed that categories such as “Game_arcade”, “Game_rolePlaying” and “Game_strategy” have a too many paid apps then free (>20% of the apps are paid).

Figure 4.13 shows the median pricing. We avoid using mean because outliers could skew the data a lot. So, using median gives the distribution shown in figure 4.13.

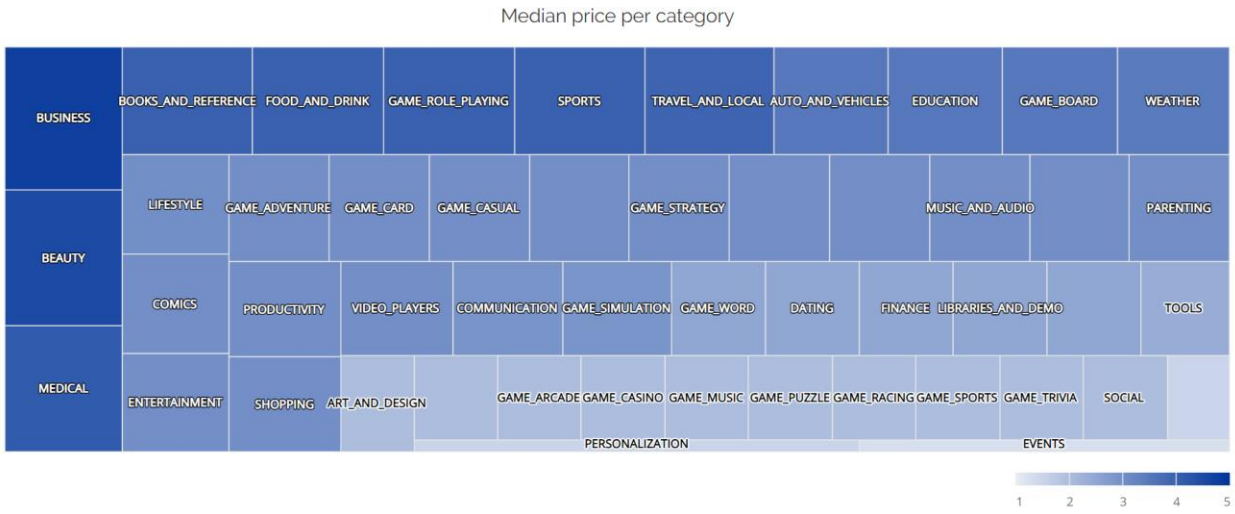

Fig.4.13 Median pricing per category

It is observed that, “Business” is the category with most expensive apps, a median of almost 4.98 USD followed by ‘Beauty’ and ‘Medical’ category.

Figure 4.14 shows the total amount spent by category which will give us the maximum revenue generated by each category.

Fig.4.14 Total revenue per category

Game_Arcade is the category that generated the highest revenue which is because it has the highest popularity plus highest number of paid games.

Figure 4.15 shows the no of installs targeted audience wise.

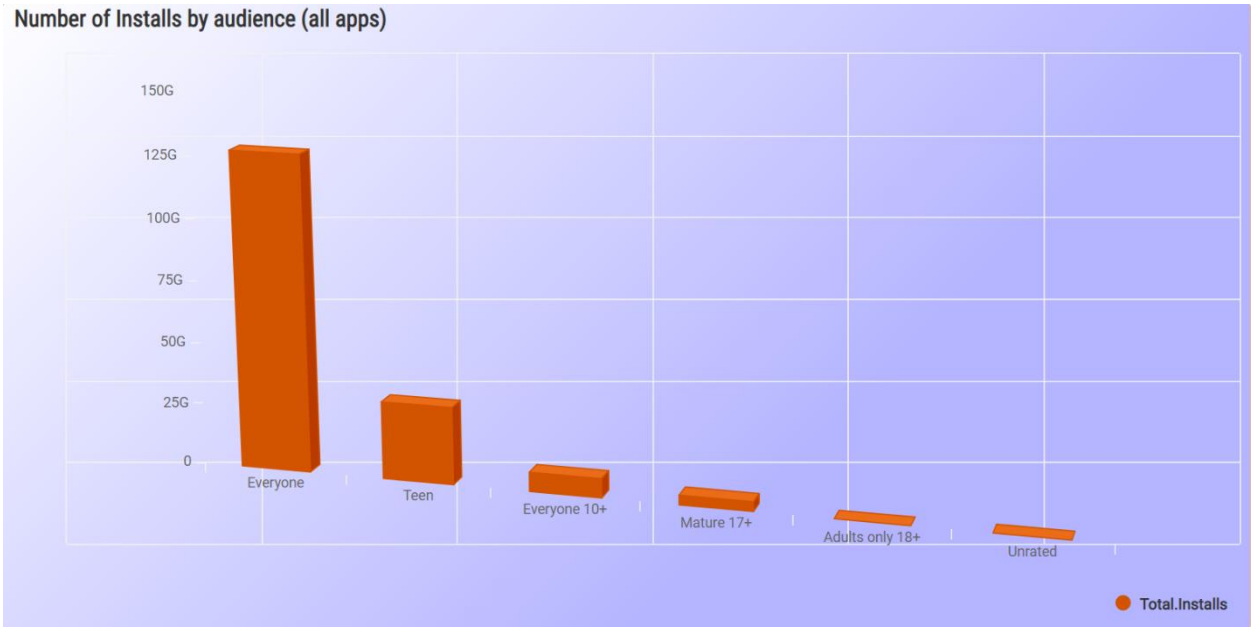

Fig.4.15 No of installs vs audience

This states that application targeted for everyone will get the maximum no of installs.

Figure 4.16 shows the time series chart according to last updated date.

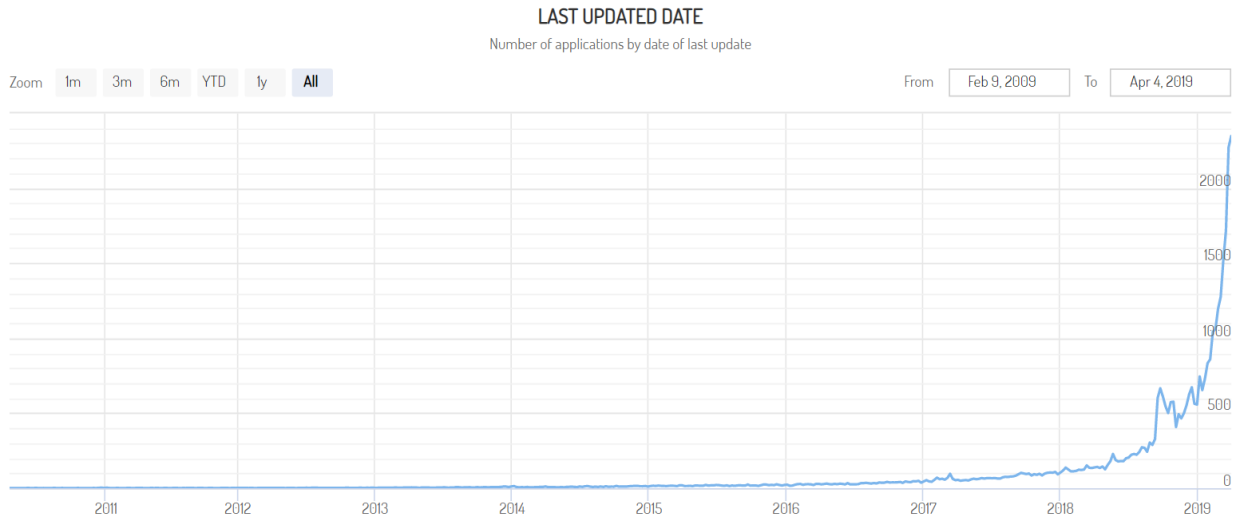

Fig.4.16 Time series for last updated date

It is observed that most applications have been updated within the last 6 months before April 2019, but there are applications that haven't not seen an update for the last 5 years.

The column "Android version" relates to the minimum Android version the application supports. Figure 4.17 shows the minimum android version required by no of installs.

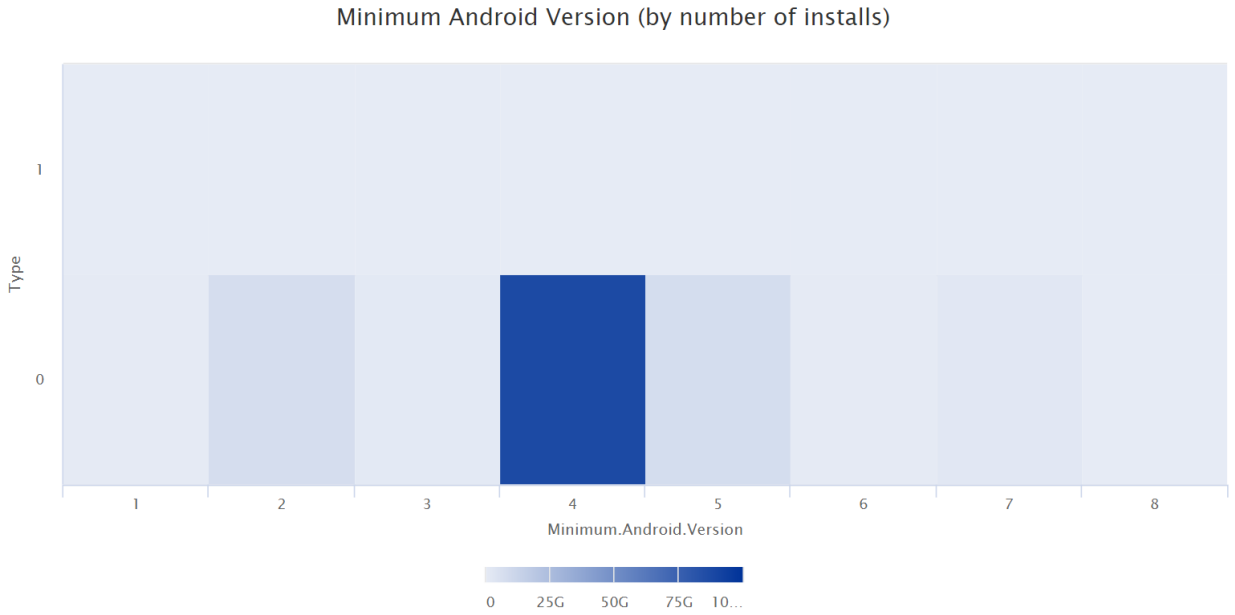

Fig.4.17 Minimum android version required for most hits

It is observed that applications rely mostly that users have Android version 4.1 and up which is useful for legacy devices.

Discussions and Conclusions

Principal Findings

In this study, we defined a new success metric for every android application. We employed clustering algorithm to get this new metrics. We extracted 3 features out of 11 for the clustering analysis. We tried out clustering with different values of k to find out the best cluster to fit all the data points. The issue of class imbalance was addressed by trying out different values of k during clustering. Once the class label was defined by clustering, we generated a classification model to predict the success metric of the testing dataset. A total of 3 classification model were used to develop the classification model i.e. Random forests, k-Nearest neighbours and Naïve Bayes algorithm. Naïve Bayes was trained with 10-fold cross validation which performed better than random sampling of data without replacement. Feature importance was considered on training subset by using mean Gini decrease before building the model. We found that the Random forest classification model performed the best and accuracy level was achieved to 95% and clustering with k=7 gave us the best set of clusters with a good distribution of datapoints.

As the android market is growing up day by day, our study has a significance use for software developers planning to launch a new application or maintaining existing application. The data analysis on various features can help any new developer to plan the type of audience to target for, defining size of the app, define the category of the application so that maximum revenue is generated. For developers who have already launched their applications, our study will help them to predict the success level of their application which will help them in keeping the application UpToDate for long run. The top applications according to our data analysis are also useful for developers for advertisement of upcoming/new applications in android market.

Limitations

The limitation that we observed in this study is in the training process of random forest model. The model leads to overfitting issue and performs in biased way. The generated clusters with different values of k could not be validated by determining their respective silhouette width due to less computational power and resource constraints.

Also, due to library limitations, categorical feature value with more than 53 value had to be dropped during implementation of random forest model.

Future Work

In future, we shall focus on selecting features based on a feature selection algorithm. Also, we shall try building classification models for the different number of clusters. This will help us to train us the classification model more accurately. This will lead us to overcome the overfitting issue that we observed in random forest model.

References

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

[2] 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.

[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] 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.

[5] 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.

[6] 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.

[7] Gautham Prakash, Jithin Koshy. (2019, Version 1). https://www.kaggle.com/gauthamp10/googleplaystore-apps/version/1#Google-Playstore-Full.csv

Appendix

This is a brief description about how to run a code to see above expounded results.

We have used R language and RStudio (version 1.2.5001) as a programming IDE (Integrated Development Environment) platform. RStudio can be installed by following the instructions on https://rstudio.com/products/rstudio/. Once the RStudio IDE is installed, when you try to execute code within the R notebook, the results appear beneath the code.

Try executing the program chunks by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

To run all the code chunks, press *Ctrl+Alt+R.*

The attached code takes around half an hour to execute. We have already knitted the results to a html file. Kindly open 'bigdataresults.html' file to see the results in detail. This code cannot be run be with a random sample, as our results will totally vary because of the clustering solution.

Below guidelines can be used to follow up on the chunk by chunk results.

1. Execute chunk 'Load packages' for successful installation of all the required packages (resolving all the dependencies of the libraries), libraries will be imported with this chunk itself.
2. Load the data and see the classes of all the features by executing 'Load data' chunk.
3. Execute 'Data preprocessing' chunk to clean the data and build a graph for columns with *NA* values.
4. To assign the class labels to dataset through clustering solution, execute 'Clustering with k as 3' chunk. This chunk will show the cluster table and instances count per cluster.
5. Execute 'Cluster plot' to see the cluster plotting for k=3.
6. To calculate MeanDecreaseGini value for most of the features, execute 'Calculate feature importance' chunk. Gini index value has been calculated by using random forest model. The higher the value of MeanDecreaseGini, the more the importance of feature.

7. Execute ‘Naïve bayes without CV and FS’ chunk to get the confusion matrix on plain data by using Naïve Bayes algorithm.
8. Next chunk ‘Naïve bayes with CV’ has been written by taking some of the features into consideration(Last.updated and Latest.version features has been removed main data while performing cross validation method, as it has factor level more than 53 categories, and cannot be used to perform cross validation, it’s a library limitation). Execute ‘Naïve bayes with CV’ chunk to see results of Naïve Bayes algorithm with cross validation method. This result will show the numerical possibility for each value of all the features and confusion matrix at last.
9. To see the results of Naïve Bayes with cross validation and feature selection, execute chunk named ‘Naive bayes with CV and FS’.
10. Results of random forest can be seen from chunk ‘Random forest without CV and FS’.
11. In chunk ‘Clustering with k as 7’, you can see the clustering with 7 clusters, which gives you the 7 different classes.
12. Execute chunk ‘Plot for k as 7’ to see the clustering for k=7.
13. We have applied Naïve Bayes classification on data with 7 class labels, for that execute chunk ‘Naives bayes with 7clusters’.
14. Execute ‘Naïve bayes with CV for k as 7’ chunk for seeing the results of Naive Bayes classification method with cross validation after formation of 7 clusters.
15. ‘Random forest with k as 7’ chunk execution gives the accuracy for random forest algorithm using 7 clusters over the given dataset.
16. For 7 clusters, K-Nearest Neighbor algorithm results can be seen by executing ‘KNN’ chunk.
17. Exploratory data analysis can be observed from ‘Popular Category wrt NumOfInstalls’ chunk and below. This graph shows the popular categories with respect to number of installs.
18. Execute chunk ‘Categories wrt mean rating’ to see the graph for all the categories with respect to their mean rating.
19. ‘Dist of app size’ chunk shows the distribution of application size that is total number of applications for each size.
20. Pie chart for each installation group can be seen by executing ‘NumOfApp for each InstallationCat’. This pie chart shows the number of applications per installation grouping.
21. For 100 applications, how many free and how many paid applications are there, can be observed by executing ‘Paid per free apps’ chunk.
22. Chunk ‘Per of paid vs free by cat’ gives the graph for percentage of paid vs free applications for each category. Green proportion is for ‘paid’ whereas, yellow proportion is for ‘free’ applications.
23. Median price per category is shown by ‘Price per category’ chunk.
24. Execute ‘Revenue generated’ chunk to see the graph for which category has generated the highest revenue.
25. Execute chunk ‘Number of installs by age’ to see the bar graph for number of installations for each audience group.
26. To see number applications by date of last update, execute ‘Last updated date’ chunk.

27. Execute ‘Min android ver’ chunk to see the heatmap for popular minimum android version by number of installs by application type whether free or paid.
28. Execute ‘Top 10app’ chunk to get top 10 applications in android market.