Mobile application success prediction report

Team members : -

180330018 - Sravya

180330040 - Peddy Shiva Sai

180330041 - Srividya

180330053 - Vineetha

Introduction :

In this world of smartphones, everything is now being ported to the mobile in the form of applications. A successful application can benefit developers, investors, advertisers and users. The rapid rise of the use of mobile applications has put a doubt in the minds of the developers that whether their app will be a success or not.In Order to assist the investors to make a decision about funding, we used **decision tree, logistic regression, neural network, randomForest, bayes classifier** and **K-Nearest neighbor** algorithms.Since the dataset collected is labeled,unsupervised learning algorithms are more suitable.The analysis of attributes of many successful applications is done and the resultant attributes are used to compare with the new apps to find out the success possibility.

Literature Survey :

1. Naive bayes classifier

   Author : TUCKERMAN, C.J.

   Published year : December 12, 2014.

   Naive bayes classifier and a linear regression are used to predict the average rating of the application. The performance of the models is not sufficient to justify their use in driving investments in new applications.

   Success metrics : The success metrics used are the number of installations.The top five percent of applications are selected as successful to find the region of success containing two clusters; one with high numbers of downloads and one close to one lakh downloads.

   success x = 1 {xscore >= 4.5} · 1 xinstalls >= 5 × 104

   They considered average user rating as a success metric to see how a relatively implementation of linear regression performs.

Prediction :

Naive Bayes: A naive bayes text classifier was chosen, and the implementation was written in Java for use in a MapReduce pipeline over the text. Cross-validation was performed so that the performance of the model can be measured. The run-time performance of the algorithm was extremely good.

p (x“download” = 1|success x = 1) = .0001

Linear Regression : They used the continuous features to predict the average rating of the application.This shows the performance of the model via RMSE.

They started to build a picture of the genres of applications in which users are interested by using this model.Building classifiers to find these types of low utility, potentially dangerous applications could help protect consumers from accidentally installing bad applications. Exploring methods for detecting these non-sentence segments of the description would be a natural next step to remedying the problem.

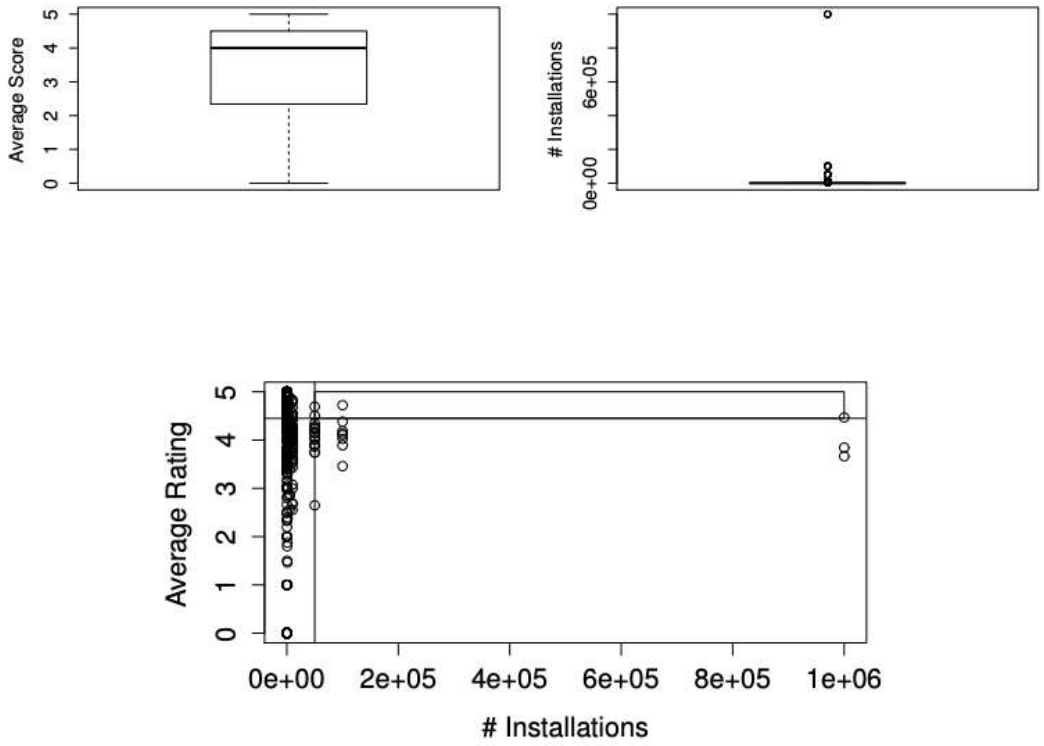


FIGURE 1. The distribution of average scores and numbers of installations.

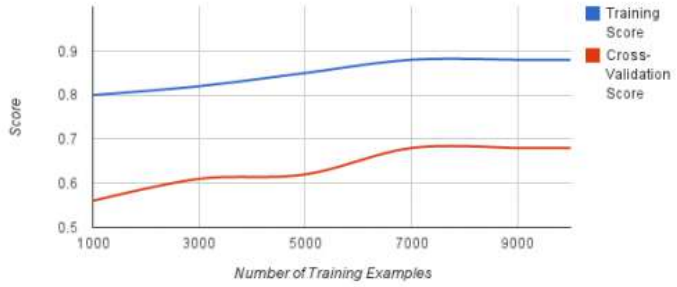


FIGURE 2. Training curve for the naïve bayes model.

2. Random Forest, Linear regression, Neural network

   Authors : Kevin Daimi and Noha Hazzazi

   Published year : 2019

   Linear regression : Linear regression finds the best-fitting straight line through the given points.The table provides results for prediction during testing.

TABLE II. ASSOCIATED STATISTICS

Correlation coefficient	0.7719
Mean absolute error	0.6350
Root mean squared error	0.9576
Relative absolute error	56.311%
Root relative squared error	63.5983%

Only 7187 instances are used because 10 instances were saved for the unseen samples (dataset).

TABLE III. USER RATING PREDICTION - UNSEEN INSTANCES

Unseen Instance	Predicted User Rating
1	4.069656
2	3.573414
3	2.191008
4	4.138177
5	3.951197
6	3.292516
7	3.995180
8	4.914003
9	4.014383
10	4.523481

Neural network : Neural networks are able to learn from examples and capture hidden and non-linear dependencies even with presence of major noise in the training set.

TABLE IV. ASSOCIATED STATISTICS

Correlation coefficient	0.7724
Mean absolute error	0.5986
Root mean squared error	0.9747
Relative absolute error	53.083%
Root relative squared error	64.7353%

TABLE V. USER RATING PREDICTION - UNSEEN INSTANCES

Unseen Instance	Predicted User Rating
1	4.658
2	4.660
3	3.022
4	4.284
5	4.378
6	3.931
7	4.616
8	4.718
9	1.277
10	3.500

Random Forest : The random forest algorithm is adaptable to use ML algorithms that generate excellent results, even without hyper-tuning parameters.

3. Multilayer perceptron classifier using PCA

Authors : Mehrdad Razavi Dehkordi, Habib Seifzadeh, Ghassan Beydoun Mohammad H, Nadimi-Shahraki

Published year : 03 June 2020

Table 7 Best accuracy for each algorithm

Algorithm	Accuracy	
	With applying PCA	Without applying PCA
MLP with **trainlm** Learning function	99.99%	89.47%
MLP with **trainbr** Learning function	99.99%	75.68%
LVQ with **learnlv1** Learning function	76.5%	77.5%
LVQ with **learnlv2** Learning function	76.5%	77%
NPR	87%	95.5%
kNN	76%	76%
SVM	85%	88.5%
Random forest	74.6%	76.85%
Decision tree	85.5%	86%

Table 8 Average runtime for each algorithm

Algorithm	Runtime	
	With applying PCA	Without applying PCA
MLP with **trainlm** Learning Function	5.3 s	1.798 s
MLP with **trainbr** Learning Function	2.392 s	3.281 s
LVQ with **learnlv1** Learning Function	14.39 s	10.995 s
LVQ with **learnlv2** Learning Function	18.696 s	18.654 s
NPR	1.572 s	1.572 s
kNN	9.168 s	10.07 s
SVM	2.01 s	1.258 s
Random forest	1.986 s	2.138 s
Decision tree	19.081 s	17.766 s

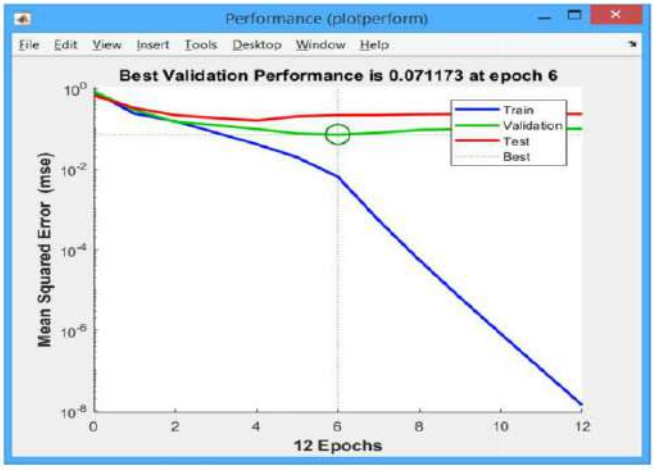


Fig. 10 Mean-squared error for train, test, and validation in MLP before applying PCA algorithm

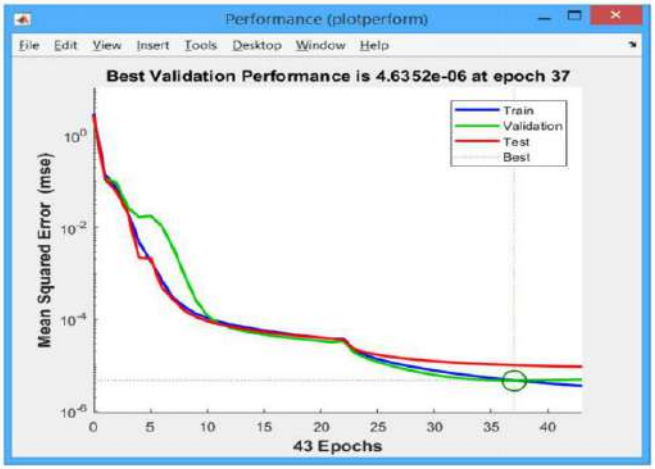


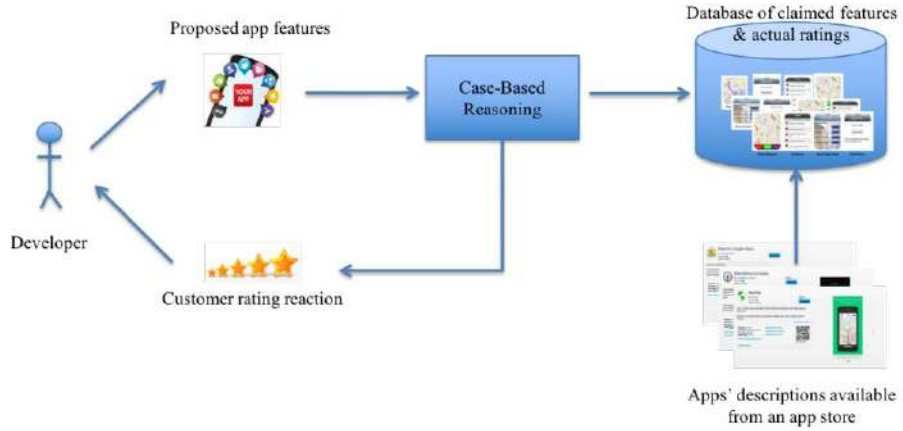
Fig. 11 Mean-squared error for train, test, and validation in MLP after applying PCA algorithm

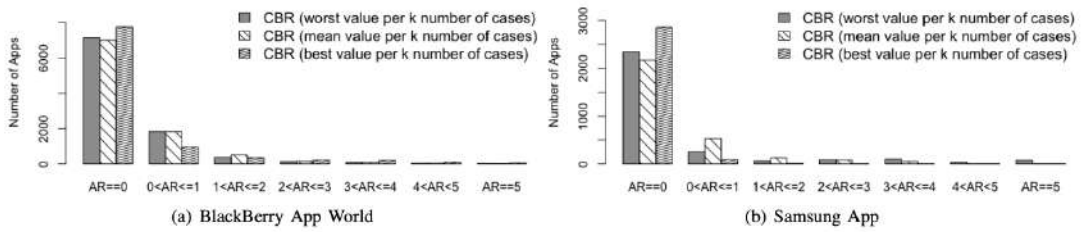
4 . Customer Rating Reactions Predicted Purely Using App Features

Authors : Federica Sarro, Mark Harman, Yue Jia, Yuanyuan Zhang

Published year : 2019

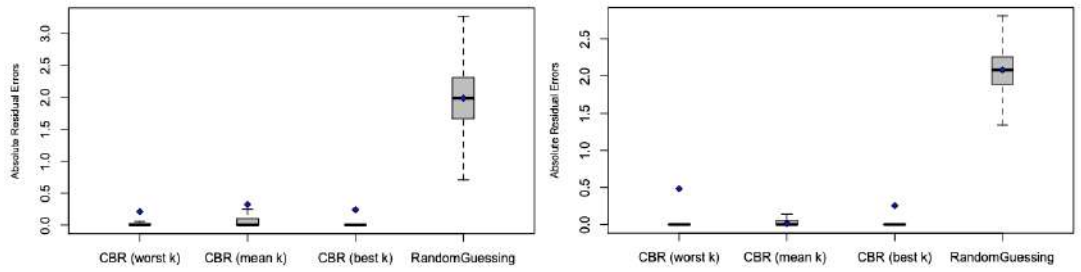
They estimated the app's rating from claimed features. They focused on answering the research questions, methods and statistical tests. They observed ratings predictability variability across app store’s categories.They measured the accuracy of a prediction system as the Mean of Absolute Residual errors, where the Absolute Residual error is defined as the absolute value of the difference between the observed value of the dependent variable and the predicted value.





(a) BlackBerry App World (b) Samsung App

Fig. 2. **RQ1:** Number of apps with Absolute Residual (AR) errors ranging from 0 to 5 obtained by CBR (worst, mean and best *k*) for the BlackBerry App World (a) and Samsung App (b) stores.



(a) Prediction Error for BlackBerry App World (b) Prediction Error for Samsung App

5 . Random forest, KNN and SVM

Authors: Golam Md. Muradul Bashir, Md. Showrov Hossen, Dip Karmoker, Md. Junaeed Kamal

Published year : December 18, 2019

They implemented random forest, KNN and SVM algorithms.In KNN algorithm, if the classification of a sample is unknown, then it could be predicted by considering the classification of its nearest neighbor samples.

In case of linearly separable data in two dimensions, a typical machine learning algorithm tries to find a boundary that divides the data in such a way that the misclassification error can be reduced. In the SVM algorithm, it chooses the decision boundary that maximizes the distance from the nearest data points of all the classes. It finds the most optimal decision boundary. In case of nonlinearly separable data, the simple SVM algorithm cannot be used.

Pseudo codes for random forest and KNN :

- Randomly select "k" features from total "m" features. Where k << m
- Among the "k" features, calculate the node "d" using the best split point.
- Split the node into daughter nodes using the best split.
- Repeat 1st to 3rd steps until "l" number of nodes has been reached.
- Build forest by repeating steps 1st to 4th for "n" number times to create "n" number of trees.

- for i=1 to m do
- Compute distance d(Xi,x)
- end for
- Compute set I containing indices for the k smallest distances d(Xi,x)
- Return majority label for{Yi where i ∈ I}

6 . Naïve Bayes Classifier, Decision tree

Authors : Rahul Aralikatte, Giriprasad Sridhara, Neelamadhav Gantayat and Senthil Mani

Published year : 2018

The authors discussed the review-rating mismatch, through establishing multiple systems that can automatically detect the inconsistency between these two, to prove this mismatch they implemented Naïve Bayes Classifier, Decision tree.The outcome of the survey was both the Developer and end users of Android app agreed that rating of an app should match with its review and they automated system to detect the mismatch between rating and review. They calculated ratings based on the store rating.

Mir Riyanul Islam stated 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 users are dependent on others opinion while making any decision so according to the author people install the app based on the rating. 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.

Luiz et al proposed a framework for mobile application developers with which they will be able to bring in modification on features that are found negative based on the end users' evaluation of their application. Their 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 biases of the features that affect the overall rating of an application.

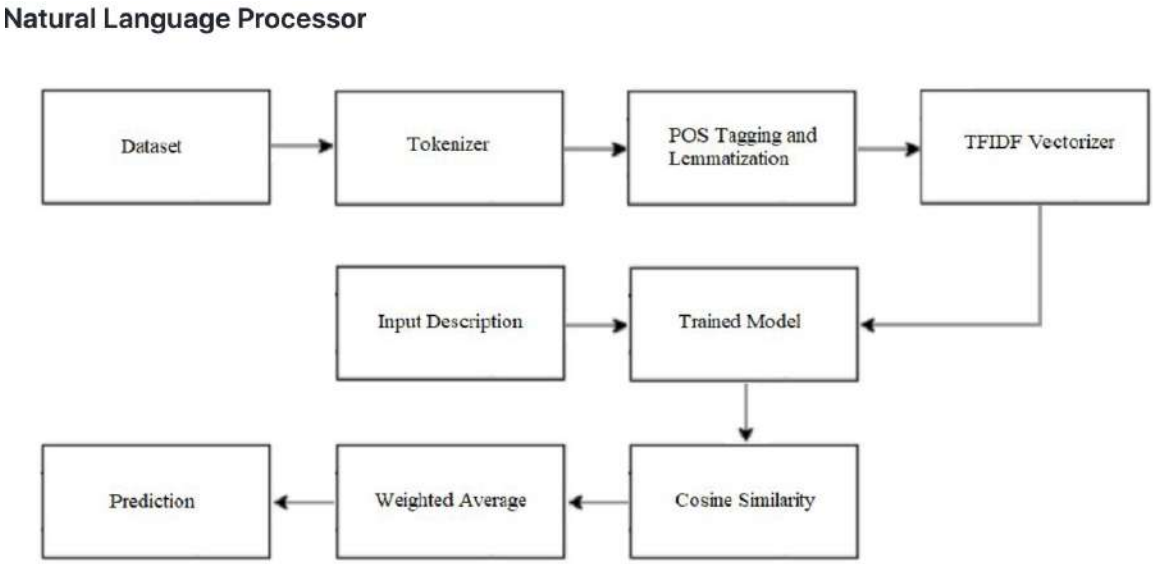
7. Mobile app success prediction system

Authors : Shreyam Kela, Sayalee Shankar Bhusari, Harsh Agarwal, Vaishali Koul

Published year : 2018

They applied Natural Language Processing on the App Store dataset to find the Nearest Neighbours of the new description entered, using TF IDF of the descriptions in the dataset and their Cosine Similarities with respect to the new description. They considered real world constraints such as age-group focus area, geographical focus area, app size.

Our system analyses the new description entered and determines the detailed analysis such as the potential app store rating, total number of installs, top similar apps, and so on.

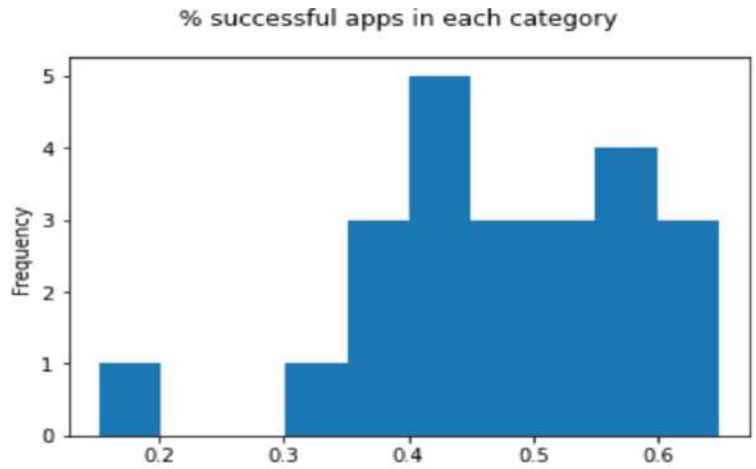


8. KNN

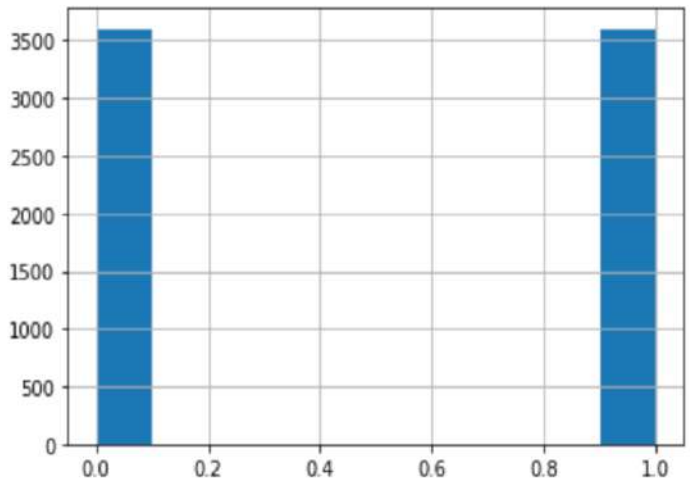
Authors : Jackbridger

Published year : 2020

Two types of predictions are made based on no.of ratings and average rating. Initial testing for categories is done with logistic regression. The models implemented are LinearSVC, gridSearchCV, randomforest which gave them 56%, 58% and 60% accuracies respectively.



Logistic Regression initial test with categories



9. Neural network model

Authors : Antony paulson

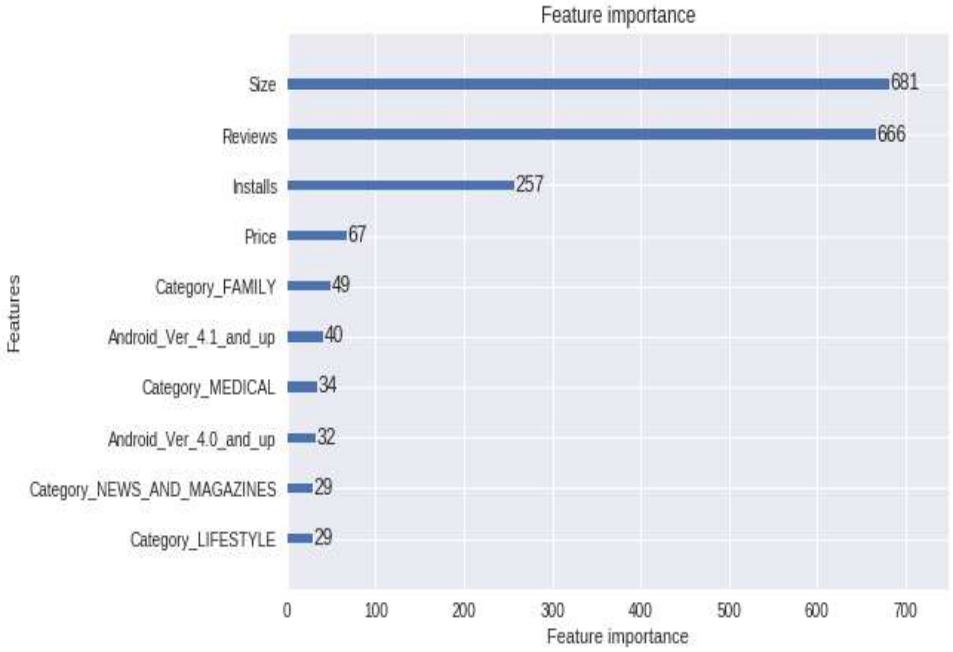
Published year : 2019

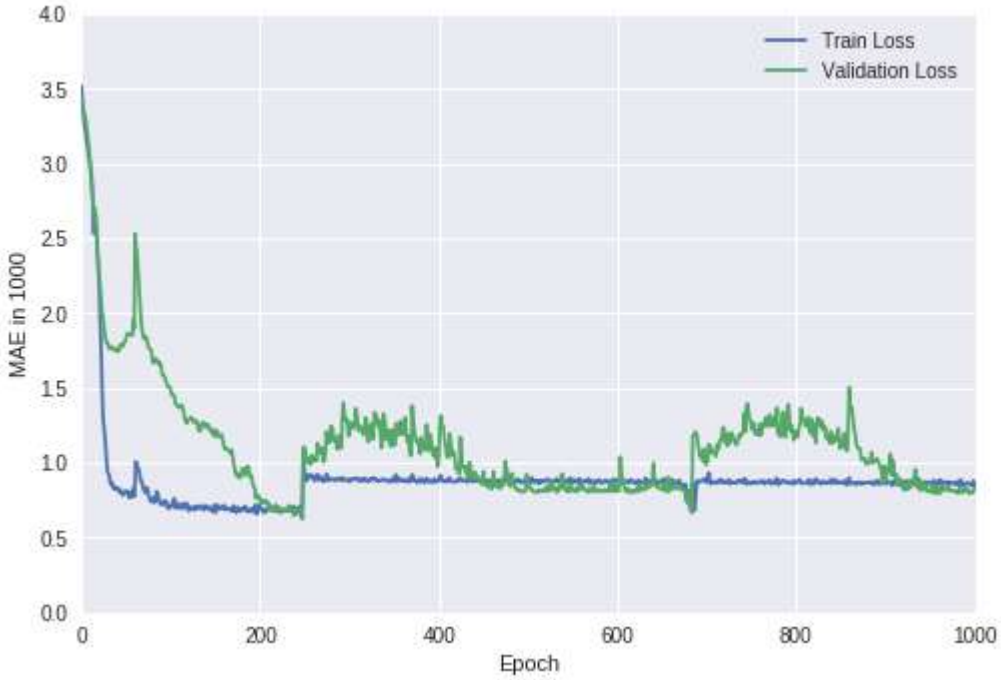
They used a tensorflow sequential model with 3 layers, a decision tree model, gradient boosted method.They plotted several graphs to correlate the preprocessed data.

	MAE	MSE	RMSE
Multiple Linear Regression	1.147578	2.425740	1.557479
Neural Networks	0.833658	2.224392	1.491440
Decision Tree Regression	0.767805	2.184993	1.478172
Light Gradient Boosted Model	0.623170	1.065379	1.032172

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 64)                4928
_________________________________________________________________
dense_17 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_v1_8 (Ba (None, 32)                128
_________________________________________________________________
dense_18 (Dense)             (None, 16)                528
_________________________________________________________________
batch_normalization_v1_9 (Ba (None, 16)                64
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 17
=================================================================
Total params: 7,745
Trainable params: 7,649
Non-trainable params: 96
```

The plot below helps us to visualise important predictors.They aimed to employ machine learning & visual analytics concepts to gain insights into how applications become successful and achieve high user ratings.The aim of the project was to first generally visualize the distribution of the dataset across categories, identify correlations among the parameters and to then find an accurate machine learning model which could fairly accurately predict user ratings on any app when similar data is available. Seaborn & Matplotlib libraries of python were used to perform visualizations on python. Subsequently, four different machine learning models were used and trained on this data.





10. Google play store app success prediction

Authors : Kishan lal

Published year : 2019

They used kinds of regression models like linear regression, support vector regressor, random forest regressor and tested the test dataset using linear regression, support vector regressor and random forest regressor.

Testing on Test Set

1. Linear Regression

[36]:
```
pipe = make_pipeline(column_trans,linreg)
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

[37]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
print('Mean Absolute Error: {}'.format(mean_absolute_error(y_pred,y_test)))
print('Mean Squared Error: {}'.format(mean_squared_error(y_pred,y_test)))
print('Root Mean Squared Error: {}'.format(np.sqrt(mean_absolute_error(y_pred,y_test))))
```

```
Mean Absolute Error: 0.3489443281752163
Mean Squared Error: 0.25837127713467606
Root Mean Squared Error: 0.5907150989903815
```

2. Support Vector Regressor

[38]:
```
pipe = make_pipeline(column_trans,svr)
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

[39]:
```
print('Mean Absolute Error: {}'.format(mean_absolute_error(y_pred,y_test)))
print('Mean Squared Error: {}'.format(mean_squared_error(y_pred,y_test)))
print('Root Mean Squared Error: {}'.format(np.sqrt(mean_absolute_error(y_pred,y_test))))
```

```
Mean Absolute Error: 0.33534403036396315
Mean Squared Error: 0.2711355594863636
Root Mean Squared Error: 0.579088965845459
```

3. Random Forest Regressor

[40]:
```
pipe = make_pipeline(column_trans,forest_model)
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

[41]:
```
print('Mean Absolute Error: {}'.format(mean_absolute_error(y_pred,y_test)))
print('Mean Squared Error: {}'.format(mean_squared_error(y_pred,y_test)))
print('Root Mean Squared Error: {}'.format(np.sqrt(mean_absolute_error(y_pred,y_test))))
```

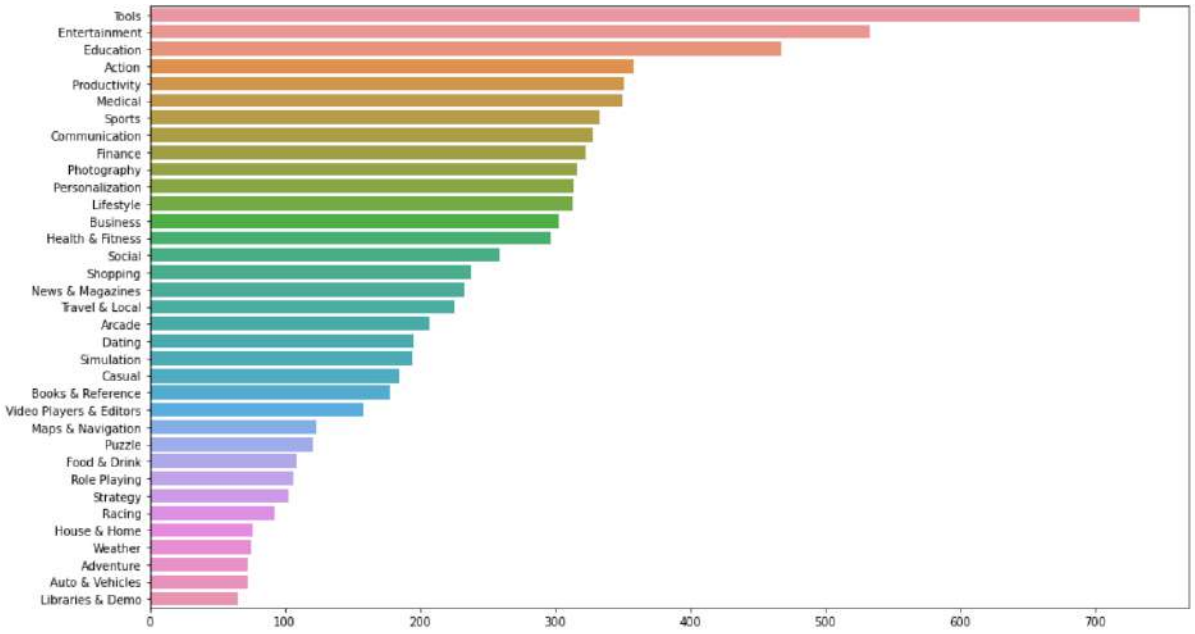
```
Mean Absolute Error: 0.3472564853666874
Mean Squared Error: 0.2566080142048638
Root Mean Squared Error: 0.589284723513759
```

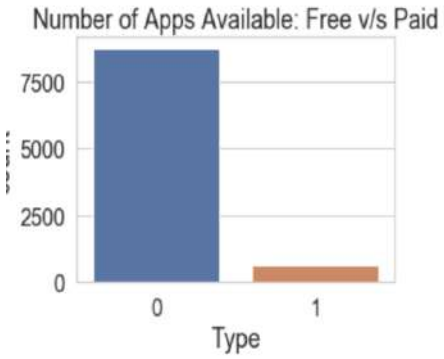
Data Collection : The google play store dataset is extracted from kaggle which contains 13 attributes (columns) and 10842 instances(rows).The instances with incorrectly labeled app categories and null rating are removed. The main attributes which are used to predict the success are : **Category, Number of Installs, Rating, Reviews, genre and type(**free or paid).

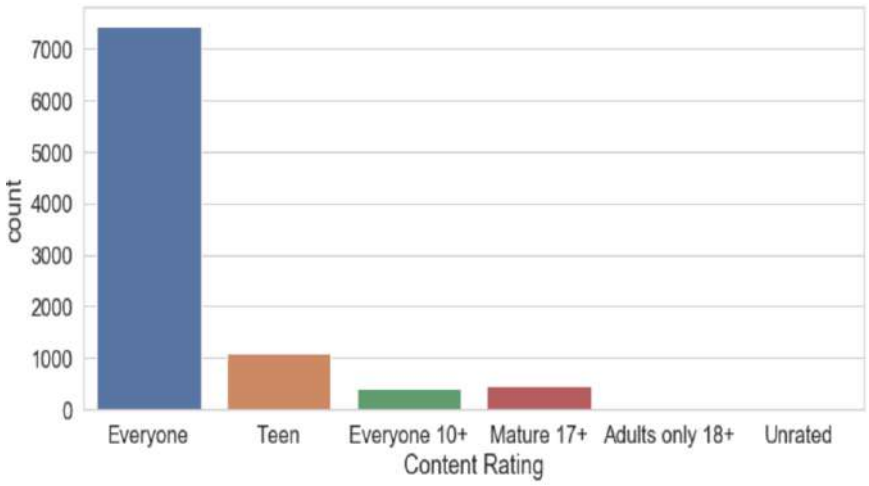
Training data : 80%, Testing data : 20%

Exploratory Data Analysis : Analysis from the dataset before predicting the results using the model.

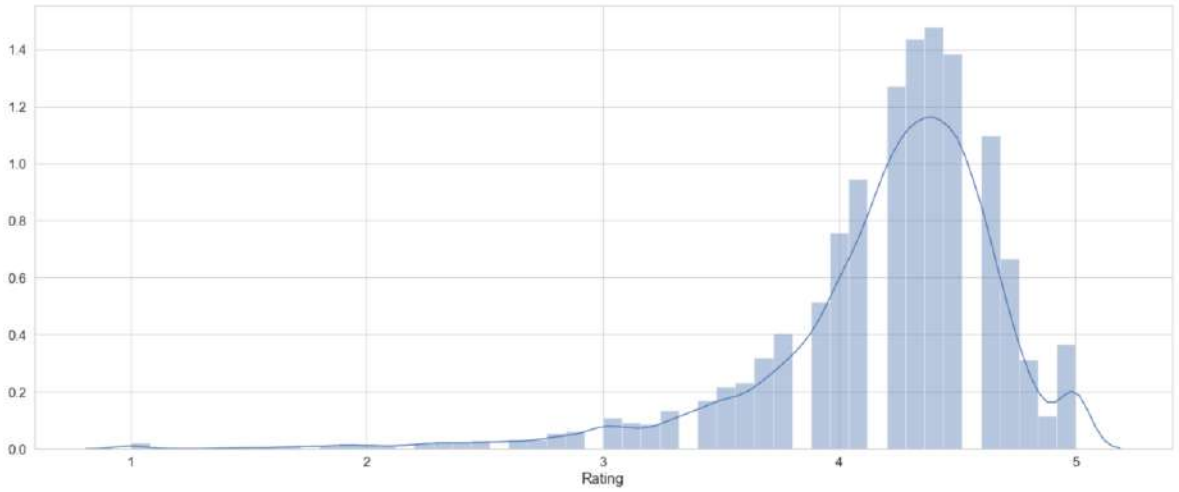
Top 35 app genres

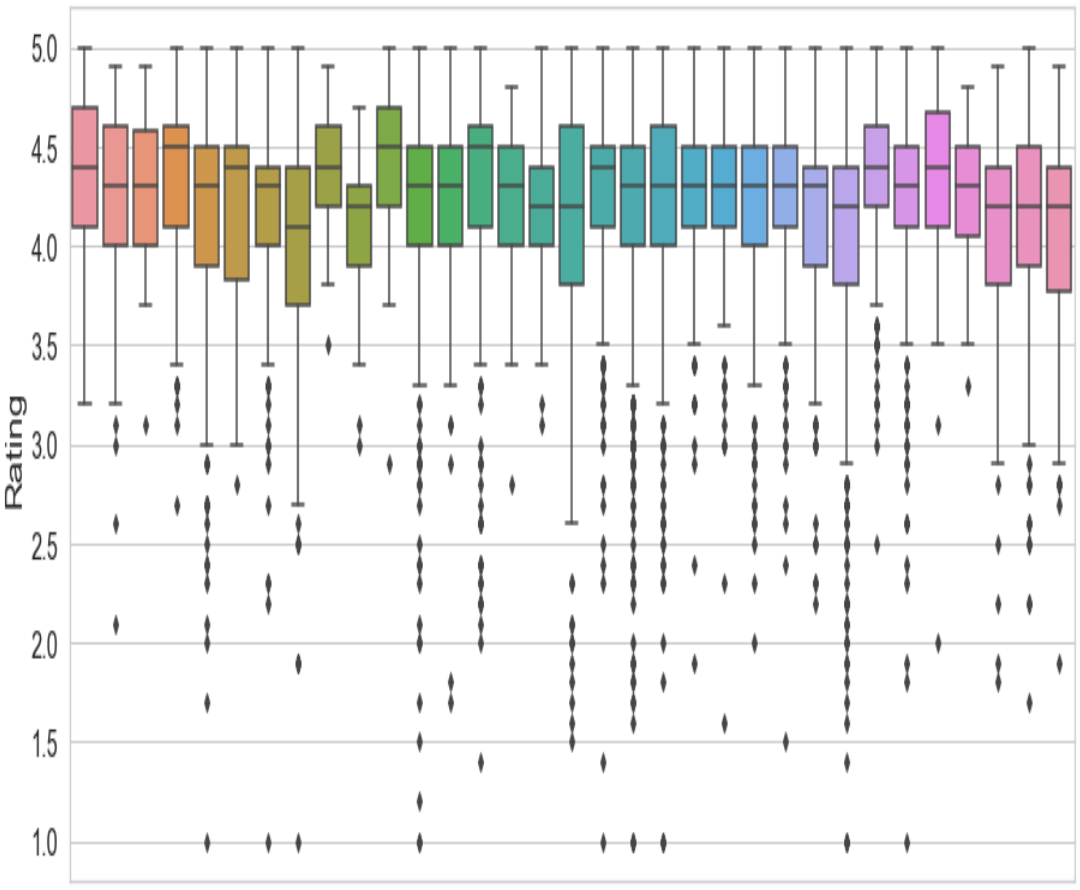




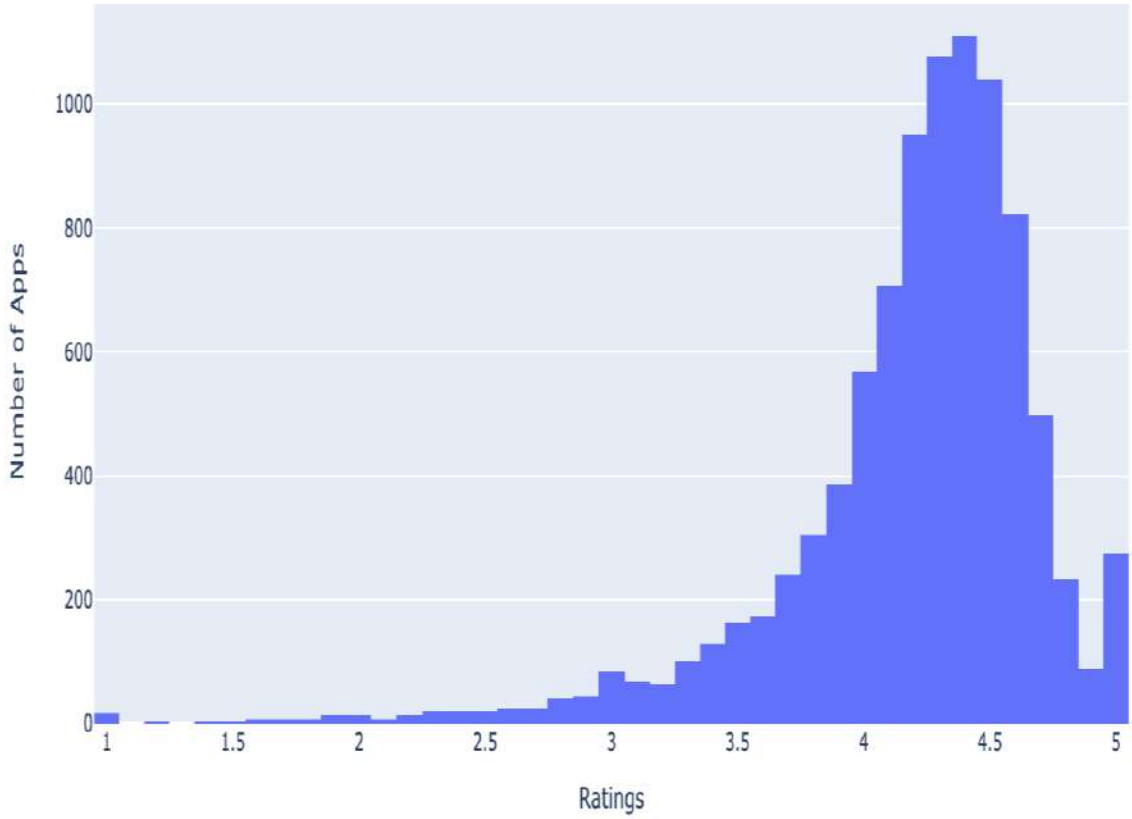


No. of Apps with full ratings: 274





Rating vs categories

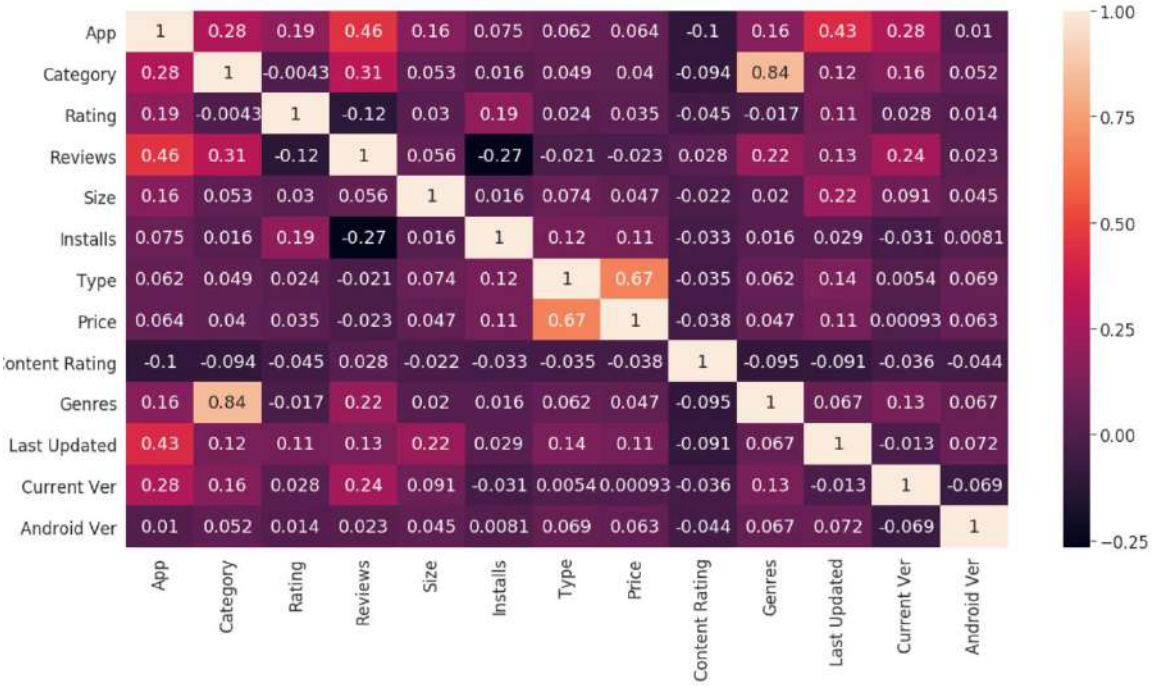


Key Features :

The key features that affect the success prediction are category, ratings, reviews, no.of installs, genres and types.

Implementation :

The parameter that would affect ad revenue the most is the number of installs an app has. More installs means more people are opening the app and viewing the embedded ads, hence, there is more money being made. A free application may lead to more installs. The correlation between installs and other parameters :



Installs and Reviews has the strongest inverse correlation because more reviews are conducted on apps that are the most popular.Since the Installs parameter is independent and not correlated to any other parameters, we must only use Installs to show the popularity of an app. Apps with larger amounts of installs would generate the most revenue.

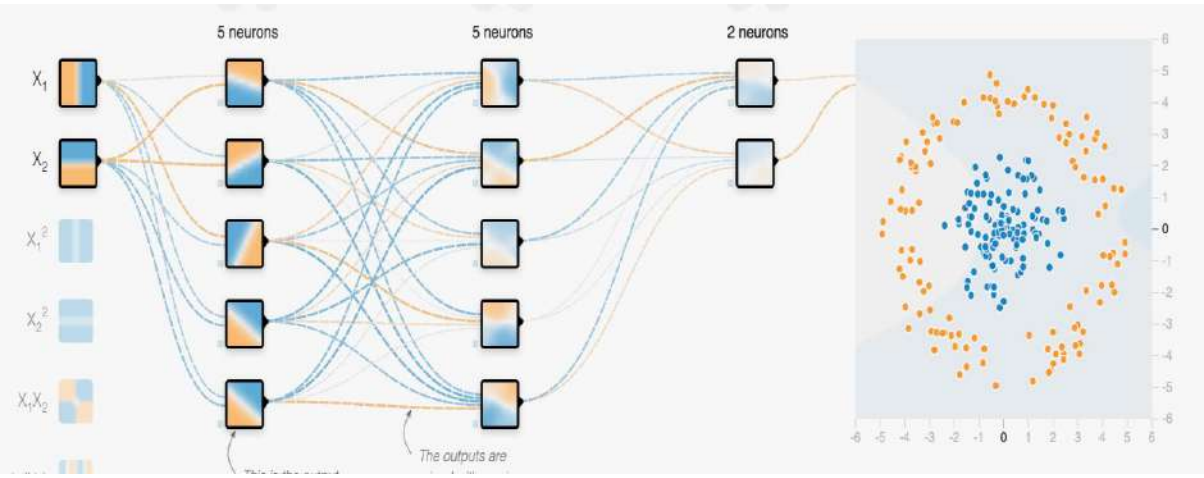
Categories with highest rating

Apps that produce huge revenue

Category	
ART_AND_DESIGN	10
AUTO_AND_VEHICLES	8
BEAUTY	10
BOOKS_AND_REFERENCE	11
BUSINESS	11
COMICS	10
COMMUNICATION	10
DATING	11
EDUCATION	10
ENTERTAINMENT	8
EVENTS	11
FAMILY	11
FINANCE	11
FOOD_AND_DRINK	10
GAME	11
HEALTH_AND_FITNESS	11
HOUSE_AND_HOME	9
LIBRARIES_AND_DEMO	9
LIFESTYLE	11
MAPS_AND_NAVIGATION	10
MEDICAL	11
NEWS_AND_MAGAZINES	11
PARENTING	11
PERSONALIZATION	11
PHOTOGRAPHY	11
PRODUCTIVITY	11
SHOPPING	10
SOCIAL	11
SPORTS	11
TOOLS	11
TRAVEL_AND_LOCAL	10
VIDEO_PLAYERS	10
WEATHER	9

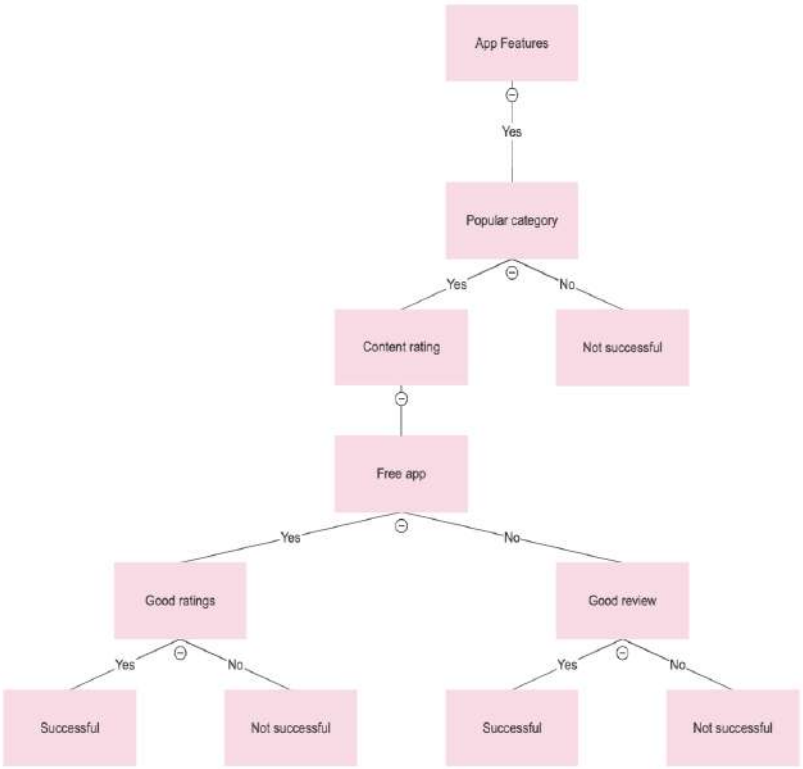
	App	Installs	Content Rating
0	Messenger – Text and Video Chat for Free	1000000000	Everyone
1	Google Drive	1000000000	Everyone
2	Instagram	1000000000	Teen
3	Google	1000000000	Everyone
4	Instagram	1000000000	Teen
5	Google+	1000000000	Teen
6	Subway Surfers	1000000000	Everyone 10+
7	Maps - Navigate & Explore	1000000000	Everyone
8	Google	1000000000	Everyone
9	Hangouts	1000000000	Everyone
10	Google+	1000000000	Teen
11	Google Drive	1000000000	Everyone
12	Google Play Movies & TV	1000000000	Teen
13	Google Photos	1000000000	Everyone
14	Google Street View	1000000000	Everyone
15	Subway Surfers	1000000000	Everyone 10+
16	Maps - Navigate & Explore	1000000000	Everyone
17	Subway Surfers	1000000000	Everyone 10+
18	Google Drive	1000000000	Everyone
19	Instagram	1000000000	Teen
20	Google Chrome: Fast & Secure	1000000000	Everyone

Multilayer perceptron architecture :



Neural network consists of 3 hidden layers, first layer with 5 neurons, second layer with 5 neurons and third layer with 2 neurons.

Decision tree :



We have improved the accuracy of our models compared to the models in the literature survey by increasing the training data and max leaf nodes and changing the attributes which predict the success.The way in which we will go about preprocessing the data is by binarizing the Installs column. Anything above 100,000 will be considered equal to 1, and everything below that threshold will be equal to 0. Also, we will encode the object labels of desired features.

Source code :

```
import pandas as pd

import numpy as np

import seaborn as sns

from plotly.offline import iplot

import plotly.graph_objs as go

import matplotlib.pyplot as plt

from sklearn import preprocessing

from sklearn.metrics import accuracy_score,precision_score,recall_score

from sklearn.tree import DecisionTreeClassifier

import sklearn

import graphviz
```

```
from sklearn import tree
```

```
df_apps = pd.read_csv("D:/TheAnaconda/googleplaystore.csv")
```

```
df_apps.head()
```

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
0	Photo Editor & Candy Camera & Grid & ScrapBook	ART_AND_DESIGN	4.1	159	19M	10,000+	Free	0	Everyone	Art & Design	January 7, 2018	1.0.0	4.0.3 and up
1	Coloring book moana	ART_AND_DESIGN	3.9	967	14M	500,000+	Free	0	Everyone	Art & Design;Pretend Play	January 15, 2018	2.0.0	4.0.3 and up
2	U Launcher Lite – FREE Live Cool Themes, Hide ...	ART_AND_DESIGN	4.7	87510	8.7M	5,000,000+	Free	0	Everyone	Art & Design	August 1, 2018	1.2.4	4.0.3 and up
3	Sketch - Draw & Paint	ART_AND_DESIGN	4.5	215644	25M	50,000,000+	Free	0	Teen	Art & Design	June 8, 2018	Varies with device	4.2 and up
4	Pixel Draw - Number Art Coloring Book	ART_AND_DESIGN	4.3	967	2.8M	100,000+	Free	0	Everyone	Art & Design;Creativity	June 20, 2018	1.1	4.4 and up

```
categories = list(df_apps["Category"].unique())
```

```
print("There are {0:.0f} categories!".format(len(categories)-1))
```

```
print(categories)
```

```
size = df_apps["Size"]
```

```
categories.remove('1.9')
```

```
There are 33 categories!
['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY', 'BOOKS_AND_REFERENCE', 'BUSINESS', 'COMICS', 'COMMUNICATION', 'DATING', 'EDUC
ATION', 'ENTERTAINMENT', 'EVENTS', 'FINANCE', 'FOOD_AND_DRINK', 'HEALTH_AND_FITNESS', 'HOUSE_AND_HOME', 'LIBRARIES_AND_DEMO',
'LIFESTYLE', 'GAME', 'FAMILY', 'MEDICAL', 'SOCIAL', 'SHOPPING', 'PHOTOGRAPHY', 'SPORTS', 'TRAVEL_AND_LOCAL', 'TOOLS', 'PERSONAL
IZATION', 'PRODUCTIVITY', 'PARENTING', 'WEATHER', 'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_AND_NAVIGATION', '1.9']
```

```
a = df_apps.loc[df_apps["Category"] == "1.9"]
```

```
print(a.head())
```

```
print("This mislabeled app category affects {} app at index {}.".format(len(a),int(a.index.values)))
```

```
df_apps = df_apps.drop(int(a.index.values),axis=0)
```

```
df_apps['Rating'].isnull().sum()
```

```
                                        App Category  Rating Reviews  \
10472  Life Made WI-Fi Touchscreen Photo Frame      1.9    19.0    3.0M

         Size Installs Type     Price Content Rating            Genres  \
10472  1,000+     Free    0  Everyone            NaN  February 11, 2018

      Last Updated Current Ver Android Ver
10472       1.0.19  4.0 and up         NaN
This mislabeled app category affects 1 app at index 10472.

1474
```

```
df_apps = df_apps.drop(df_apps[df_apps['Rating'].isnull()].index, axis=0)
```

```
df_apps.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9366 entries, 0 to 10840
Data columns (total 13 columns):
App                9366 non-null object
Category           9366 non-null object
Rating             9366 non-null float64
Reviews            9366 non-null object
Size               9366 non-null object
Installs           9366 non-null object
Type               9366 non-null object
Price              9366 non-null object
Content Rating     9366 non-null object
Genres             9366 non-null object
Last Updated       9366 non-null object
Current Ver        9362 non-null object
Android Ver        9364 non-null object
dtypes: float64(1), object(12)
memory usage: 1.0+ MB
```

df_apps["Rating"].describe()

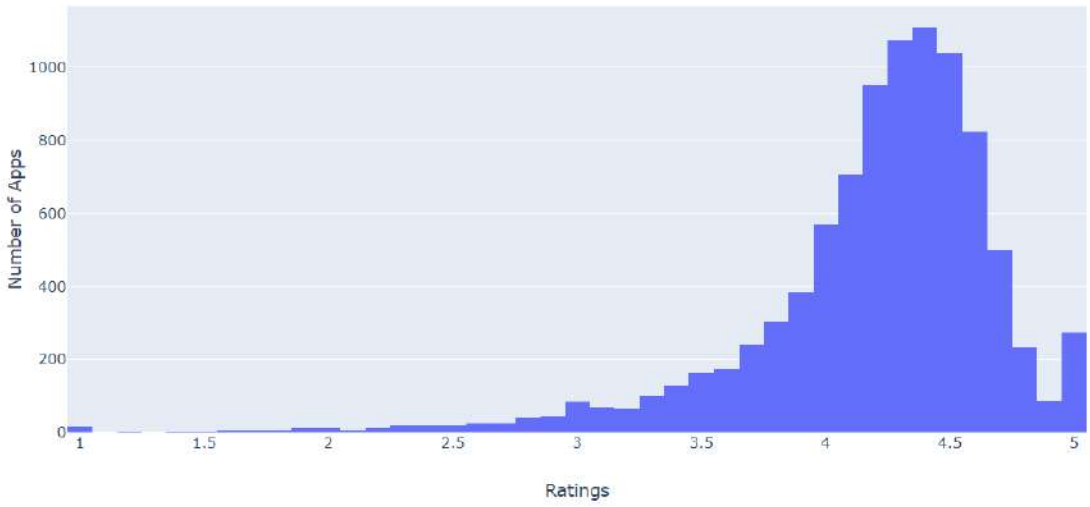
```
count    9366.000000
mean        4.191757
std         0.515219
min         1.000000
25%         4.000000
50%         4.300000
75%         4.500000
max         5.000000
Name: Rating, dtype: float64
```

layout = go.Layout(xaxis=dict(title='Ratings'),yaxis=dict(title='Number of Apps'))

data = [go.Histogram(x=df_apps["Rating"])]

fig = go.Figure(data=data, layout=layout)

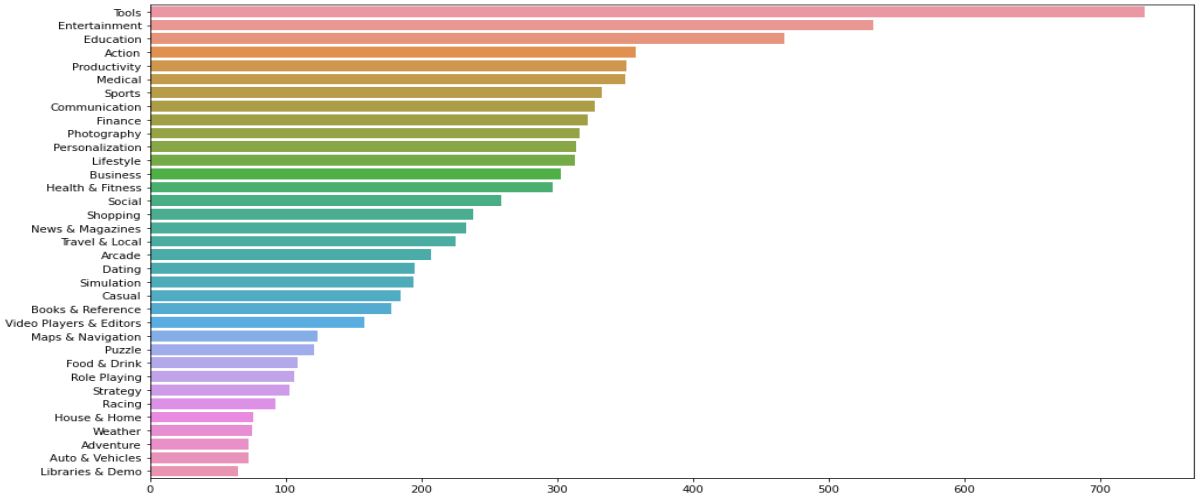
iplot(fig, filename='basic histogram')



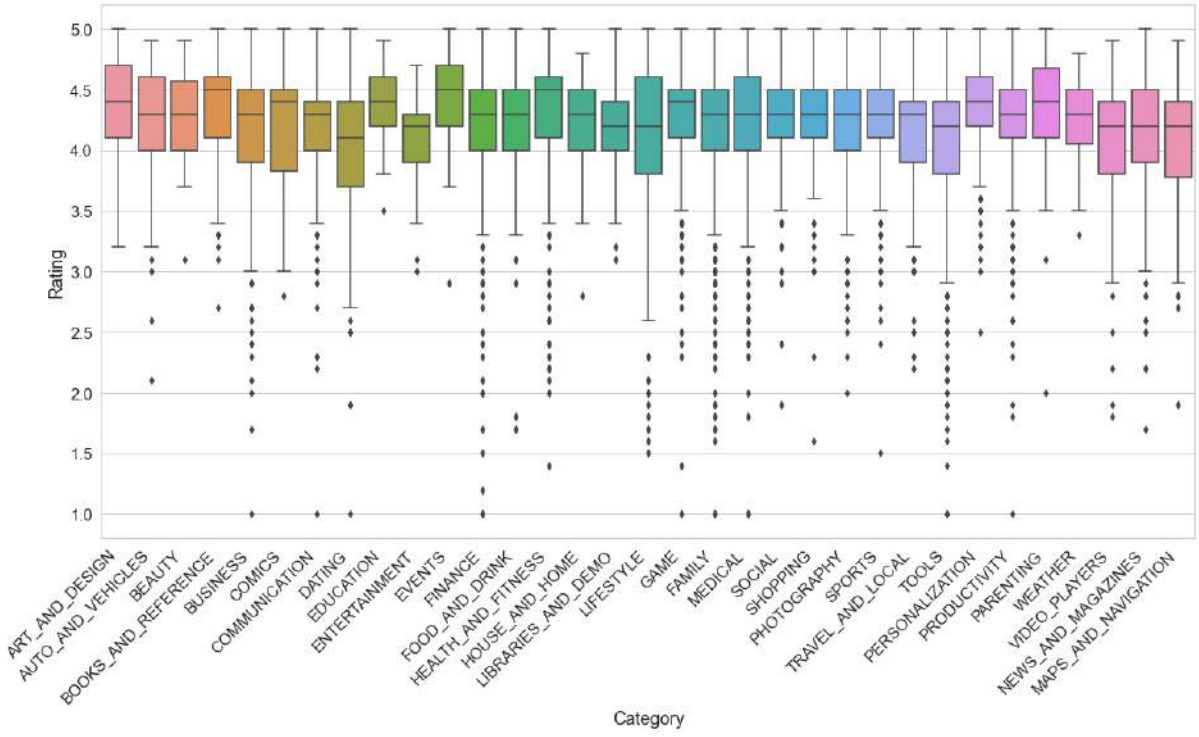
plt.figure(figsize=(16, 9.5))

genres = df_apps["Genres"].value_counts()[:35]

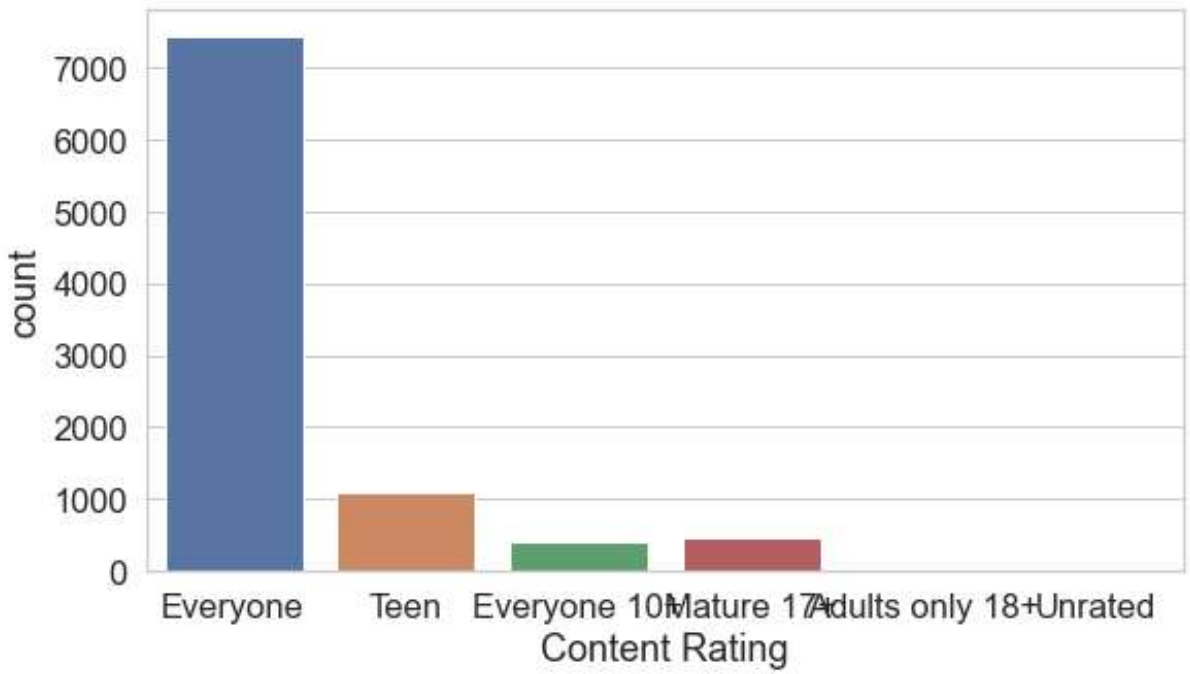
ax = sns.barplot(x=genres.values, y=genres.index)



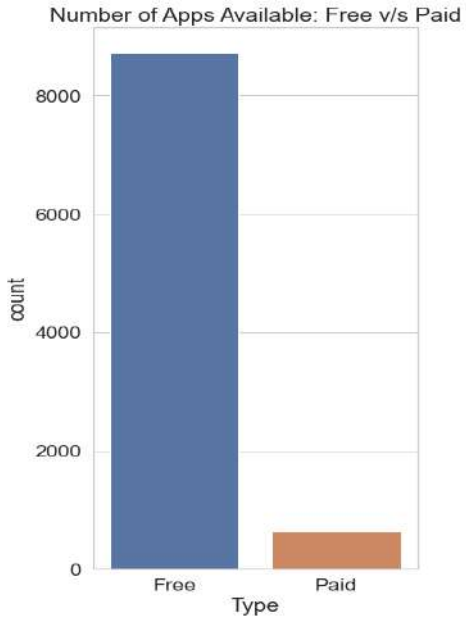

```
sns.set(rc={'figure.figsize':(20,10)}, font_scale=1.5, style='whitegrid')

ax = sns.boxplot(x="Category",y="Rating",data=df_apps)

labels = ax.set_xticklabels(ax.get_xticklabels(), rotation=45,ha='right')
```




```
plt.figure(figsize = (9,5))

sns.countplot(df_apps['Content Rating'])

plt.show()
```

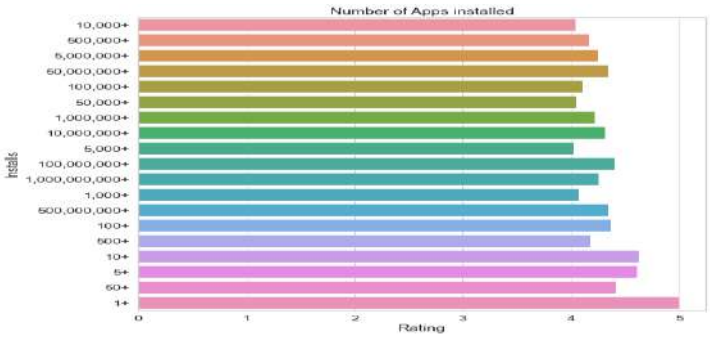



```
plt.figure(figsize=(10,10))

plt.subplot(1,2,1)

sns.countplot(x='Type',data=df_apps)

plt.title("Number of Apps Available: Free v/s Paid")
```




```
plt.subplot(1,2,2)

sns.barplot(x='Rating',y='Installs',data=df_apps,ci=None)

plt.title("Number of Apps installed")
```

plt.tight_layout()

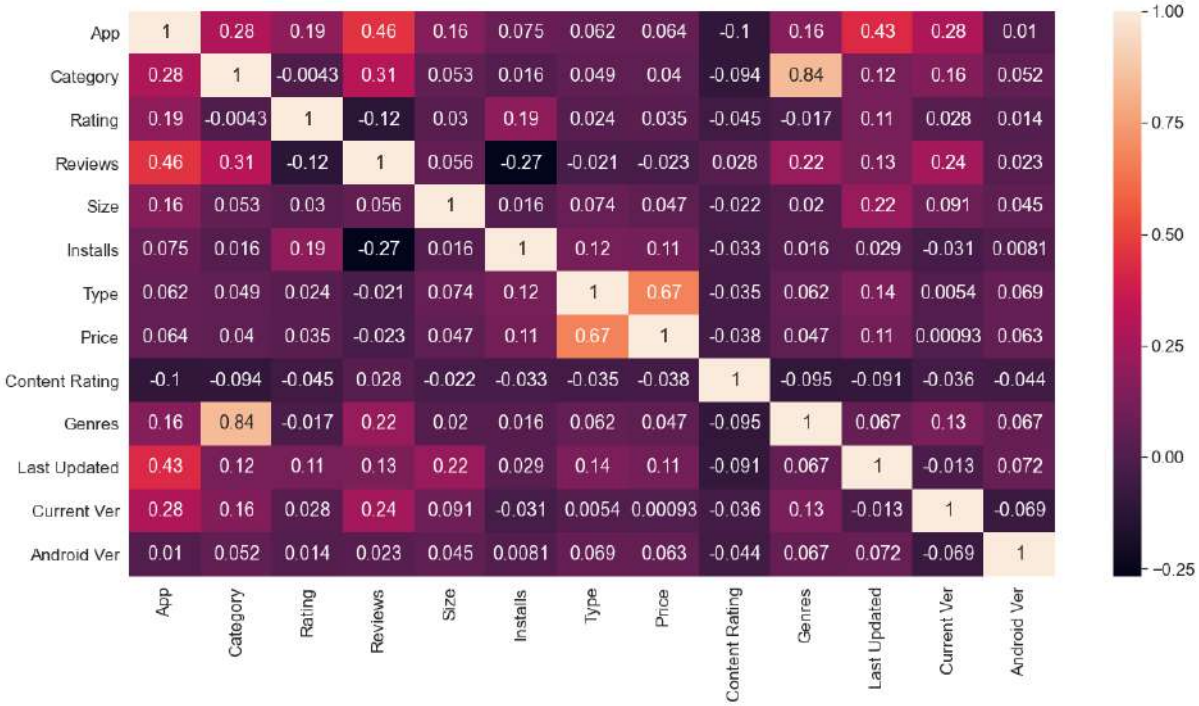


df_apps.dtypes

df_apps["Type"] = (df_apps["Type"] == "Paid").astype(int)

corr = df_apps.apply(lambda x: x.factorize()[0]).corr()

sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,annot=True)



highRating = df_apps.copy()

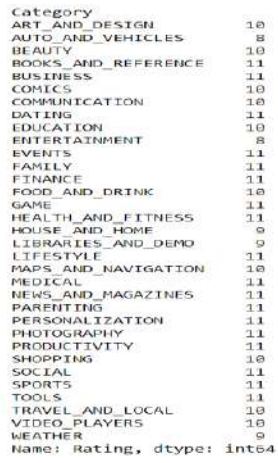
highRating = highRating.loc[highRating["Rating"] >= 4.0]

highRateNum = highRating.groupby('Category')['Rating'].nunique()

df_apps.dtypes

df_apps["Type"] = (df_apps["Type"] == "Paid").astype(int)

highRateNum



```
Category
ART_AND_DESIGN          10
AUTO_AND_VEHICLES        8
BEAUTY                  10
BOOKS_AND_REFERENCE     11
BUSINESS                11
COMICS                  10
COMMUNICATION           10
DATING                  11
EDUCATION               10
ENTERTAINMENT            8
EVENTS                  11
FAMILY                  11
FINANCE                 11
FOOD_AND_DRINK          10
GAME                    11
HEALTH_AND_FITNESS      11
HOUSE_AND_HOME           9
LIBRARIES_AND_DEMO       9
LIFESTYLE               11
MAPS_AND_NAVIGATION     10
MEDICAL                 11
NEWS_AND_MAGAZINES      11
PARENTING               11
PERSONALIZATION         11
PHOTOGRAPHY             11
PRODUCTIVITY            11
SHOPPING                10
SOCIAL                  11
SPORTS                  11
TOOLS                   11
TRAVEL_AND_LOCAL        10
VIDEO_PLAYERS           10
WEATHER                  9
Name: Rating, dtype: int64
```

popApps = df_apps.copy()

popApps = popApps.drop_duplicates()

popApps["Installs"] = popApps["Installs"].str.replace("+","")

popApps["Installs"] = popApps["Installs"].str.replace(",","")

popApps["Installs"] = popApps["Installs"].astype("int64")

popApps["Price"] = popApps["Price"].str.replace("$","")

popApps["Price"] = popApps["Price"].astype("float64")

popApps["Size"] = popApps["Size"].str.replace("Varies with device","0")

popApps["Size"] = (popApps["Size"].replace(r'[kM]+$', '', regex=True).astype(float) *\

popApps["Size"].str.extract(r'[\d\.]+([kM]+)', expand=False).fillna(1).replace(['k','M'], [10**3, 10**6]).astype(int))

popApps["Reviews"] = popApps["Reviews"].astype("int64")

popApps = popApps.sort_values(by="Installs",ascending=False)

popApps.reset_index(inplace=True)

popApps.drop(["index"],axis=1,inplace=True)

popApps.loc[:40,['App','Installs','Content Rating']]

	App	Installs	Content Rating
0	Messenger – Text and Video Chat for Free	1000000000	Everyone
1	Google Drive	1000000000	Everyone
2	Instagram	1000000000	Teen
3	Google	1000000000	Everyone
4	Instagram	1000000000	Teen
5	Google+	1000000000	Teen
6	Subway Surfers	1000000000	Everyone 10+
7	Maps - Navigate & Explore	1000000000	Everyone
8	Google	1000000000	Everyone
9	Hangouts	1000000000	Everyone
10	Google+	1000000000	Teen
11	Google Drive	1000000000	Everyone
12	Google Play Movies & TV	1000000000	Teen
13	Google Photos	1000000000	Everyone
14	Google Street View	1000000000	Everyone
15	Subway Surfers	1000000000	Everyone 10+
16	Maps - Navigate & Explore	1000000000	Everyone
17	Subway Surfers	1000000000	Everyone 10+
18	Google Drive	1000000000	Everyone
19	Instagram	1000000000	Teen
20	Google Chrome: Fast & Secure	1000000000	Everyone
21	Subway Surfers	1000000000	Everyone 10+
22	YouTube	1000000000	Teen
23	Google Play Books	1000000000	Teen
24	Google Photos	1000000000	Everyone
25	WhatsApp Messenger	1000000000	Everyone
26	Google Photos	1000000000	Everyone
27	Facebook	1000000000	Teen
28	Google Play Games	1000000000	Teen
29	YouTube	1000000000	Teen
30	Google Photos	1000000000	Everyone
31	Facebook	1000000000	Teen
32	Google Street View	1000000000	Everyone
33	Google News	1000000000	Teen
34	Subway Surfers	1000000000	Everyone 10+
35	Messenger – Text and Video Chat for Free	1000000000	Everyone
36	Gmail	1000000000	Everyone
37	Hangouts	1000000000	Everyone
38	Gmail	1000000000	Everyone
39	Hangouts	1000000000	Everyone
40	WhatsApp Messenger	1000000000	Everyone

```
popAppsCopy = popApps.copy()

label_encoder = preprocessing.LabelEncoder()


popAppsCopy['Category']= label_encoder.fit_transform(popAppsCopy['Category'])

popAppsCopy['Content Rating']= label_encoder.fit_transform(popAppsCopy['Content Rating'])

popAppsCopy['Genres']= label_encoder.fit_transform(popAppsCopy['Genres'])

popAppsCopy.dtypes


popAppsCopy = popAppsCopy.drop(["App","Last Updated","Current Ver","Android Ver"],axis=1)

print("There are {} total rows.".format(popAppsCopy.shape[0]))

countPop = popAppsCopy[popAppsCopy["Installs"] > 100000].count()

print("{} Apps are Popular!".format(countPop[0]))

print("{} Apps are Unpopular!\n".format((popAppsCopy.shape[0]-countPop)[0]))

print("For an 80-20 training/test split, we need about {} apps for testing\n".format(popAppsCopy.shape[0]*.20))

popAppsCopy["Installs"] = (popAppsCopy["Installs"] > 100000)*1 #Installs Binarized

print("Cut {} apps off Popular df for a total of 3558 Popular training apps.".format(int(4568*.22132)))
```

```
print("Cut {} apps off Unpopular df for a total of 3558 Unpopular training apps.\n".format(int(4324*.17738)))


testPop1 = popAppsCopy[popAppsCopy["Installs"] == 1].sample(1010,random_state=0)

popAppsCopy = popAppsCopy.drop(testPop1.index)

print("Values were not dropped from training dataframe.",testPop1.index[0] in popAppsCopy.index)


testPop0 = popAppsCopy[popAppsCopy["Installs"] == 0].sample(766,random_state=0)

popAppsCopy = popAppsCopy.drop(testPop0.index)

print("Values were not dropped from training dataframe.",testPop0.index[0] in popAppsCopy.index)


testDf = testPop1.append(testPop0)

trainDf = popAppsCopy

H_X_train=trainDf

testDf = testDf.sample(frac=1,random_state=0).reset_index(drop=True)

trainDf = trainDf.sample(frac=1,random_state=0).reset_index(drop=True)

y_train = trainDf.pop("Installs")

X_train = trainDf.copy()

y_test = testDf.pop("Installs")

X_test = testDf.copy()


X_train = X_train.drop(['Rating'], axis=1)

X_test = X_test.drop(['Rating'], axis=1)
```

```
There are 8892 total rows.
4568 Apps are Popular!
4324 Apps are Unpopular!

For an 80-20 training/test split, we need about 1778.4 apps for testing

Cut 1010 apps off Popular df for a total of 3558 Popular training apps.
Cut 766 apps off Unpopular df for a total of 3558 Unpopular training apps.

Values were not dropped from training dataframe. False
Values were not dropped from training dataframe. False
```

```
feature_cols=['Category','Reviews','Size','Type','Price','Content Rating','Genres']
```

print("{} Apps are used for Training.".format(y_train.count()))

print("{} Apps are used for Testing.".format(y_test.count()))

X_test.head(3)

```
7116 Apps are used for Training.
1776 Apps are used for Testing.
```

	Category	Reviews	Size	Type	Price	Content Rating	Genres
0	11	74744	60000000.0	0	0.0	1	101
1	14	725897	31000000.0	0	0.0	4	0
2	11	36	48000000.0	0	0.0	1	94

DECISION TREE

popularity_classifier = DecisionTreeClassifier(max_leaf_nodes=29)

popularity_classifier.fit(X_train, y_train)

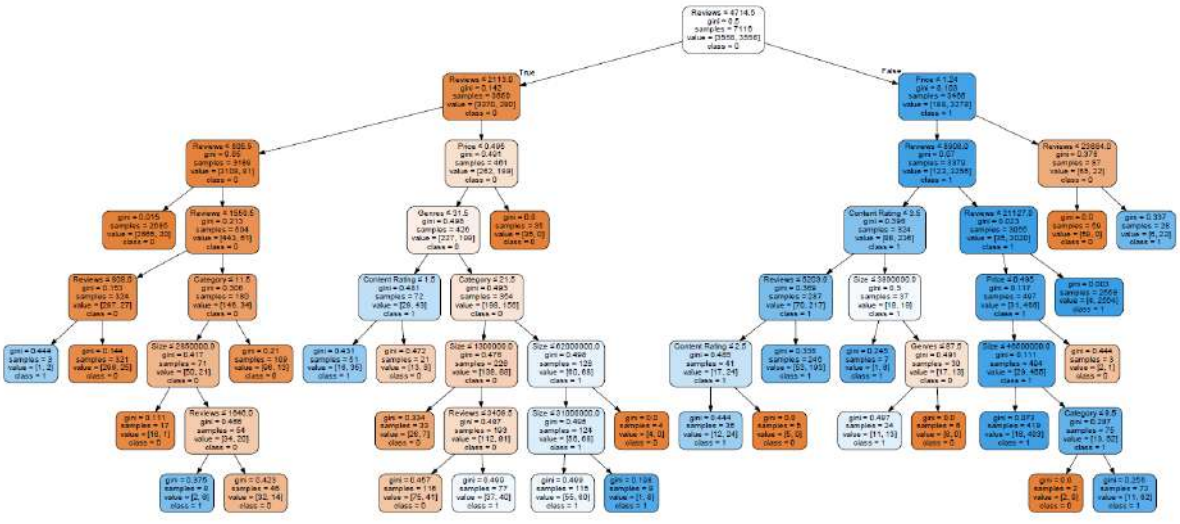
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=29,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

dot_data = tree.export_graphviz(popularity_classifier, out_file=None, filled=True, rounded=True,

special_characters=True,class_names=['0','1'],feature_names=feature_cols)

graph = graphviz.Source(dot_data)

graph



predictions = popularity_classifier.predict(X_test)

```
print("Predicted: ",predictions[:30])

print("Actual:    ",np.array(y_test[:30]))
```

```
Predicted:  [1 1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 1 0 0 1 1 1 1]
Actual:     [1 1 0 1 0 0 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 1 1 1]
```

```
print('Accuracy of decision tree: {0:.2f}'.format(accuracy_score(y_true = y_test, y_pred = predictions) * 100))

print('Precision of decision tree: {0:.2f}'.format(precision_score(y_true = y_test, y_pred = predictions) * 100))

print('Recall of decision tree: {0:.2f}'.format(recall_score(y_true = y_test, y_pred = predictions) * 100))

print('F-score of decision tree: {0:.2f}'.format(f1_score(y_true = y_test, y_pred = predictions) * 100))

cm1 = confusion_matrix(y_test, predictions)

print('Specificity of decision tree : {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))
```

```
Accuracy of decision tree: 95.33
Precision of decision tree: 96.49
Recall of decision tree: 95.25
F-score of decision tree: 95.86
Specificity of decision tree : 95.43
```

```
from sklearn.metrics import classification_report

print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.94      0.95      0.95       766
           1       0.96      0.95      0.96      1010

    accuracy                           0.95      1776
   macro avg       0.95      0.95      0.95      1776
weighted avg       0.95      0.95      0.95      1776
```

```
X_testCopy = X_test.copy()

X_testCopy["Popular?"] = y_test

X_testCopy[X_test["Size"] == 3600000].head(10)
```

	Category	Reviews	Size	Type	Price	Content Rating	Genres	Popular?
112	11	323	3600000.0	0	0.00	1	50	0
616	12	26	3600000.0	0	0.00	1	58	0
1297	19	16657	3600000.0	0	0.00	1	68	1
1310	31	11	3600000.0	0	0.00	4	110	0
1352	23	21	3600000.0	0	0.99	1	78	0

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split


app_success_model = LogisticRegression()

app_success_model.fit(X_train, y_train)

Logisticprediction=app_success_model.predict(X_test)

print('Accuracy of logistic regression: {0:.2f}'.format(accuracy_score(Logisticprediction,y_test) * 100))

print('Precision of logistic regression: {0:.2f}'.format(precision_score(Logisticprediction,y_test) * 100))

print('Recall of logistic regression: {0:.2f}'.format(recall_score(Logisticprediction,y_test) * 100))

print('F-score of logistic regression: {0:.2f}'.format(f1_score(Logisticprediction,y_test) * 100))

cm1 = confusion_matrix(y_test, predictions)

print('Specificity of logistic regression : {0:.2f}'.format( cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))
```

```
Accuracy of logistic regression: 89.86
Precision of logistic regression: 92.97
Recall of logistic regression: 89.60
F-score of logistic regression: 91.25
Specificity of logistic regression : 95.43
```

```
print(classification_report(y_test, predictions))
```

```
                  precision    recall  f1-score   support

               0       0.90      0.86      0.88       766
               1       0.90      0.93      0.91      1010

        accuracy                           0.90      1776
       macro avg       0.90      0.89      0.90      1776
    weighted avg       0.90      0.90      0.90      1776
```

NAIVE BAYES CLASSIFICATION

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)

from sklearn.naive_bayes import GaussianNB

nvclassifier = GaussianNB()

nvclassifier.fit(X_train, y_train)

y_pred = nvclassifier.predict(X_test)

print(y_pred)

y_compare = np.vstack((y_test,y_pred)).T
```

```
[1 1 0 ... 0 1 0]
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)


a = cm.shape

corrPred = 0

falsePred = 0


for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            corrPred +=cm[row,c]
        else:
            falsePred += cm[row,c]
print('Correct predictions: ', corrPred)

print('False predictions', falsePred)

print ('\nAccuracy of Naive Bayes Classification: {0:.2f} '.format(corrPred/(cm.sum()) * 100))

print('Precision of Naive Bayes Classification: {0:.2f}'.format(precision_score(y_pred,y_test) * 100))

print('Recall of Naive Bayes Classification: {0:.2f}'.format(recall_score(y_pred,y_test) * 100))

print('F-score of Naive Bayes Classification: {0:.2f}'.format(f1_score(y_pred,y_test) * 100))

cm1 = confusion_matrix(y_test, y_pred)
```

print('Specificity of Naive Bayes Classification : {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))

```
Correct predictions:  1621
False predictions 155

Accuracy of Naive Bayes Clasification: 91.27
Precision of Naive Bayes Clasification: 90.79
Recall of Naive Bayes Clasification: 93.67
F-score of Naive Bayes Clasification: 92.21
Specificity of Naive Bayes Clasification : 91.91
```

print(classification_report(y_test, y_pred))

	precision	recall	f1-score	support
0	0.88	0.92	0.90	766
1	0.94	0.91	0.92	1010
accuracy			0.91	1776
macro avg	0.91	0.91	0.91	1776
weighted avg	0.91	0.91	0.91	1776

RANDOM FOREST

from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier(random_state=42)

random_forest_model.fit(X_train, y_train)

rfprediction=random_forest_model.predict(X_test)

print('Accuracy of random forest: {0:.2f}'.format(accuracy_score(y_test,rfprediction) * 100))

print('Precision of random forest: {0:.2f}'.format(precision_score(y_test,rfprediction) * 100))

print('Recall of random forest: {0:.2f}'.format(recall_score(y_test,rfprediction) * 100))

print('F-score of random forest: {0:.2f}'.format(f1_score(y_test,rfprediction) * 100))

cm1 = confusion_matrix(y_test, rfprediction)

print('Specificity of random forest : {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))

```
Accuracy of random forest: 95.16
Precision of random forest: 96.76
Recall of random forest: 94.65
F-score of random forest: 95.70
Specificity of random forest : 95.82
```

print(classification_report(y_test, rfprediction))

	precision	recall	f1-score	support
0	0.93	0.96	0.94	766
1	0.97	0.95	0.96	1010
accuracy			0.95	1776
macro avg	0.95	0.95	0.95	1776
weighted avg	0.95	0.95	0.95	1776

MLP Classifier

```
import sklearn.neural_network
model = sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(10, ), activation='relu', solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5,
max_iter=1000,
shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9,
nesterovs_momentum=True,
early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10)
model.fit(X_train, y_train)

print('\n-- Training data --')
predictions = model.predict(X_train)
print('Accuracy: {0:.2f}'.format(accuracy_score(y_train, predictions) * 100.0))
print('Precision: {0:.2f}'.format(precision_score(y_train, predictions) * 100.0))
print('Recall: {0:.2f}'.format(recall_score(y_train, predictions) * 100.0))
print("F-score: ", f1_score(y_train, predictions) * 100)
cm1 = confusion_matrix(y_train, predictions)
print('Specificity : {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))
print('Classification Report:'.format(classification_report(y_train, predictions)))
print('Confusion Matrix:', confusion_matrix(y_train, predictions))
```

```
-- Training data --
Accuracy: 91.64
Precision: 98.91
Recall: 84.20
F-score:  90.96705632306058
Specificity : 99.07
Classification Report:
Confusion Matrix: [[3525   33]
 [ 562 2996]]
```

```
print('\n---- Test data ----')

predictions = model.predict(X_test)

print('Accuracy: {0:.2f}'.format(accuracy_score(y_test, predictions) * 100.0))

print('Precision: {0:.2f}'.format(precision_score(y_test, predictions) * 100.0))

print('Recall: {0:.2f}'.format(recall_score(y_test, predictions) * 100.0))

print("F-score: {0:.2f}".format(f1_score(y_test, predictions) * 100.0))

cm1 = confusion_matrix(y_test, predictions)

print('Specificity : {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))

print('Classification Report:', classification_report(y_test, predictions))

print('Confusion Matrix:', confusion_matrix(y_test, predictions))
```

```
---- Test data ----
Accuracy: 91.05
Precision: 99.42
Recall: 84.75
F-score: 91.50
Specificity : 99.35
Classification Report:               precision    recall  f1-score   support

           0       0.83      0.99      0.91       766
           1       0.99      0.85      0.92      1010

    accuracy                           0.91      1776
   macro avg       0.91      0.92      0.91      1776
weighted avg       0.92      0.91      0.91      1776

Confusion Matrix: [[761   5]
 [154 856]]
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
predictions = knn.predict(X_test)

print("Accuracy of K-NN : {0:.2f}".format(accuracy_score(y_test,predictions) * 100))

print("Precision of K-NN : {0:.2f}".format(precision_score(y_test,predictions) * 100))
```

```
print("Recall of K-NN : {0:.2f}".format(recall_score(y_test,predictions) * 100))

print("F-score of K-NN : {0:.2f}".format(f1_score(y_test,predictions) * 100))

cm1 = confusion_matrix(y_test, predictions)

print('Specificity of K-NN : {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))
```

```
Accuracy of K-NN : 73.70
Precision of K-NN : 79.41
Recall of K-NN : 72.57
F-score of K-NN : 75.84
Specificity of K-NN : 75.20
```

```
print('Classification Report:', classification_report(y_test, predictions))
```

```
Classification Report:              precision    recall  f1-score   support

           0       0.68      0.75      0.71       766
           1       0.79      0.73      0.76      1010

    accuracy                           0.74      1776
   macro avg       0.73      0.74      0.73      1776
weighted avg       0.74      0.74      0.74      1776
```

KMEANS CLUSTERING

```
from sklearn.cluster import KMeans

clustering = KMeans(n_clusters=2)

clustering.fit(X_train,y_train)

K_predictions = clustering.predict(X_test)

print('Accuracy of KMeans: {0:.2f}'.format(accuracy_score(y_test,K_predictions)*100))

print('Precision of KMeans: {0:.2f}'.format(precision_score(y_test,K_predictions)*100))

print('Recall of KMeans: {0:.2f}'.format(recall_score(y_test,K_predictions)*100))

print('F-score of KMeans: {0:.2f}'.format(f1_score(y_test,K_predictions)*100))

cm1 = confusion_matrix(y_test, K_predictions)

print('Specificity of KMeans: {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))
```

```
Accuracy of KMeans: 52.53
Precision of KMeans: 58.60
Recall of KMeans: 56.34
F-score of KMeans: 57.45
Specificity of KMeans: 47.52
```

print(classification_report(y_test,K_predictions))

	precision	recall	f1-score	support
0	0.45	0.48	0.46	766
1	0.59	0.56	0.57	1010
accuracy			0.53	1776
macro avg	0.52	0.52	0.52	1776
weighted avg	0.53	0.53	0.53	1776

HIERARCHICAL CLUSTERING

```
from sklearn.cluster import AgglomerativeClustering

H_clustering = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')

H_clustering.fit(H_X_train)

H_predictions = H_clustering.fit_predict(X_test)

print('Accuracy of Hierarchical Clustering: {0:.2f}'.format(accuracy_score(y_test,H_predictions)*100))

print('Precision of Hierarchical Clustering: {0:.2f}'.format(precision_score(y_test,H_predictions)*100))

print('Recall of Hierarchical Clustering: {0:.2f}'.format(recall_score(y_test,H_predictions)*100))

print('F-score of Hierarchical Clustering: {0:.2f}'.format(f1_score(y_test,H_predictions)*100))

cm1 = confusion_matrix(y_test, H_predictions)

print('Specificity of Hierarchical Clustering : {0:.2f}'.format(cm1[0,0]/(cm1[0,0]+cm1[0,1]) * 100))
```

```
Accuracy of Hierarcheal Clustering: 48.42
Precision of Hierarcheal Clustering: 59.71
Recall of Hierarcheal Clustering: 28.61
F-score of Hierarcheal Clustering: 38.69
Specificity of Heirarchial Clustering : 74.54
```

print(classification_report(y_test,H_predictions))

	precision	recall	f1-score	support
0	0.44	0.75	0.55	766
1	0.60	0.29	0.39	1010
accuracy			0.48	1776
macro avg	0.52	0.52	0.47	1776
weighted avg	0.53	0.48	0.46	1776

Results :

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE	SPECIFICITY
DECISION TREE	95.33%	96.49%	95.25%	95.86%	95.43%
LOGISTIC REGRESSION	89.86%	92.97%	89.60%	91.25%	95.43%
BAYES CLASSIFICATION	91.27%	90.79%	93.67%	92.21%	91.91%
RANDOM FOREST	95.16%	96.76%	95.65%	95.70%	95.82%
MLP CLASSIFIER	91.05%	99.42%	85.75%	91.50%	99.35%
K-NEAREST NEIGHBOUR	73.70%	79.41%	72.57%	75.84%	75.20%

Analysis :

Decision trees overfit the data and predictions are noisy.Random forests merges decision trees to get more accurate predictions. Logistic regression gives low variance results. Bayes classification needs less training data and performs well in multi class prediction.K-NN tries to classify unknown samples based on known classification of its neighbours.Multilayer perceptron classifies an unknown pattern with other known patterns that share the same features.This means noisy inputs will be classified because of their similarity with pure and complete inputs.

App success prediction :

```
Predicted:  [1 1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 1 0 0 1 1 1 1]
Actual:     [1 1 0 1 0 0 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 1 1 1]
```

Every 27 of 30 app’s success is predicted successfully.

Comparing with Literature papers:

Accuracy :

Decision tree:

We got 95.33% of accuracy

Academia – 70.49%

R&D Intern – 72%

Random forest :

We got 95.61% of accuracy

Academia – 75.59%

Dspace – 84.13%

KNN:

We got 73.7% of accuracy

Academia-77.26%

Dspace-82.44%

MLP:

We got 91.33% of accuracy

Complex and intelligent systems used both PCA and MLP so they achieved 99.99% of accuracy

Precision :

Decision tree:

We got 96.49% of precision

R&D Intern: 79%

Random Forest:

We got 96.7% of precision

Dspace: 84%

KNN:

We got 79.41% of precision

Dspace-82%

Recall:

Decision tree:

We got 95.25% of recall

R&D Intern-69%

Random Forest:

We got 95.45% of recall

Dspace-87%

KNN:
We got 72.57% of recall

Dspace-86%

F1-Score:

Random Forest:

We got 96.11% of F1-Score

Dspace-85%

KNN:
We got 75.84% of F1-Score

Dspace-84%

Conclusion :

To reveal various predictions, unfamiliar correlation and other valuable knowledge to make well-informed decisions.Knowing how users rate applications is useful for both users and developers.Users can decide what application to purchase and developers will invest time and money on on-demand apps.

Six methods were used to extract the user rating on apps.Decision tree technique scored the highest accuracy(95.33%), while K-Nearest Neighbor produced the least accuracy(73.70%).With respect to the dataset, the performance of K-Nearest Neighbor was poorer than other 5 algorithms.From all the statistics provided above, we observe that the decision tree algorithm performs well.The reason is that the data set contains key features which are used to determine the decisions of decision tree.This allows us to extract dynamic features from the data set and make more accurate predictions.

The preferable algorithm which has least error and high correlation coefficient is decision tree. Hence, it is the best algorithm for predicting mobile application success.

Limitations :

- When preprocessing the model, OneHotEncoding should have been used rather than the LabelEncoder. This is because a bias may exist between numbers in 0-33 rather than a matrix of 0 and 1 for category, genre, and content rating.
- Other models could have been used during the cross validation process to determine the best binary classifier model to use.
- As the training data is more, decision trees can overfit the data.
- A K-fold Cross Validation set should have been used to find and tune the model before using the test set.

References :

- https://ijarcce.com/wp-content/uploads/2015/12/IJARCCE-36.pdf
- http://cs229.stanford.edu/proj2014/Cameron%20Tuckerman,%20Predicting%20Mobile%20Application%20Success.pdf
- http://dspace.bracu.ac.bd/xmlui/bitstream/handle/10361/11407/15101108,15101020,15101109,15141002_CSE.pdf?isAllowed=y&sequence=1
- https://link.springer.com/article/10.1007/s40747-020-00154-3
- https://csce.ucmss.com/cr/books/2019/LFS/CSREA2019/ICD8050.pdf
- https://nikhil-jain.github.io/papers/prediction_SC13.pdf
- https://tdan.com/predictive-analytics-for-mobile-app-dev/25586
- https://www.tpptechnology.com/artificial-intelligence-and-machine-learning-use-cases-for-mobile-app-development
- https://search.proquest.com/openview/9700830bdb10a35139adf0ce143496d7/1?pq-origsite=gscholar&cbl=2044308
- https://www.academia.edu/43485762/Mobile_App_Success_Prediction