Google Play Store Dataset Analysis

by

Sagar Pandey

18BCE1002

A project report submitted to

Dr. PATTABIRAMAN V

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

in partial fulfilment of the requirements for the course of

CSE3020 –DATA VISUALISATION

In

B.Tech. (Computer Science Engineering)

VIT UNIVERSITY, CHENNAI

Vandalur – Kelambakkam Road

Chennai – 600127

April -2020

CERTIFICATE

Certified that this project report entitled "**Google Play Store Analysis"** is a bonafide work of Sagar Pandey (18BC1002) who carried out the "J"-Project work under my supervision and guidance for CSE3020-Data Visualisation.

Dr. Pattabiraman V

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT University, Chennai

Chennai – 600 127.

ABSTRACT

This project evaluates the performance and predictive power of 3 models that have been trained and tested on data collected from web scrapping data of 10k Play Store apps for analysing the Android market. A model trained on this data that is seen as a good fit could then be used to make certain predictions about an app that how much rating can it fetch on play store.

This model would prove to be invaluable for some developer to aim at high rating and installs for his app.

To check the level of error of a model, we can Mean Squared Error, root mean square error and mean absolute error. It is one of the procedures to measures the average of the squares of error. Basically, it will check the difference between actual value and the predicted value. For the full theory, you can always search it online. To use it, we use the mean squared error function of scikit-learn by running this snippet of code.

ACKNOWLEDGEMENT

I wish to express our sincere thanks and deep sense of gratitude to our project faculty, **Dr. Pattabiraman V,** for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

Finally, I would like to thank our deemed university, VIT Chennai, for providing us with the opportunity and facilities which ensured this project's completion.

TABLE OF CONTENTS

S.No.	Topic	Page No.
1.	Introduction	**6-7**
2.	Dataset used	**6-7**
3.	Supervised Learning	**8**
4.	Algorithms Used	**8-10**
5.	Implementation Code	**11-22**
6.	Conclusion	**22-23**

1. INTRODUCTION

It has been observed that the significant growth of the mobile application market has a great impact on digital technology. Having said that, with the ever-growing mobile app market there is also a notable rise of mobile app developers; eventually resulting in sky-high revenue by the global mobile app industry. With immense competition from all over the globe, it is imperative for a developer to know that he is proceeding in the correct direction. To retain this revenue and their place in the market the app developers might have to find a way to stick into their current position. The Google Play Store is found to be the largest app market in the world. It has been observed that although it generates more than double the downloads than the Apple App Store but makes only half the money compared to the App Store. So, we scraped data from the Play Store to conduct our research on it.

2. DATASET USED

The Internet is a true gold mine of data. E-commerce and review sites are brimming with a lot of untapped data with a prominent potential to convert into meaningful insights that can help with robust decision making. Here, we explore using data science and machine learning techniques on data retrieved from one such avenue on the internet, the **Google Play Store**. The Play Store apps data has enormous potential to drive app-making businesses to success. Actionable insights can be drawn for developers to work on and capture the Android market. The dataset is chosen from Kaggle. It is the web scraped data of 10k Play Store apps for analysing the Android market. It consists of in total of 10841 rows and 13 columns.

This picture shows the top 5 data of the dataset and its description.

```
In [2]: df.head()
```

Out[2]:

	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

In [22]:
```
df.describe()
```

Out[22]:

	App	Rating	Size	Type	Content Rating	Genres	Last Updated	Current Ver	cat_ART_AND_DESIGN	cat_AUTO_AN
count	10840.000000	10840.000000	10840.000000	10840.000000	10840.000000	10840.000000	1.084000e+04	1.084000e+04	10840.000000	
mean	4046.368358	3.537685	18.151783	0.776845	1.222878	53.560978	1.272845e+09	7.566425e+25	0.005720	
std	3091.820892	1.591560	22.170849	0.416381	1.053488	37.950691	5.489807e+08	7.877808e+27	0.075415	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000	
25%	1073.750000	3.700000	2.600000	1.000000	1.000000	19.000000	1.464719e+09	1.000000e+00	0.000000	
50%	3881.500000	4.300000	9.200000	1.000000	1.000000	60.000000	1.518979e+09	1.400000e+00	0.000000	
75%	6755.250000	4.400000	26.000000	1.000000	1.000000	82.000000	1.530988e+09	3.102164e+00	0.000000	
max	9658.000000	5.000000	100.000000	1.000000	5.000000	118.000000	1.533667e+09	8.202005e+29	1.000000	

8 rows × 41 columns

3.Supervised Learning

What is Supervised Learning?

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

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

• In classification, the goal is to assign a class (or label) from a finite set of classes to an observation. That is, responses are categorical variables. Applications include spam filters, advertisement recommendation systems, and image and speech recognition. Predicting whether a patient will have a heart attack within a year is a classification problem, and the possible classes may be true and false.

• In regression, the goal is to predict a continuous measurement for an observation. that is, the responses variables are real numbers. Applications include forecasting crypto-currency prices, energy consumption, or disease incidence and many other.

Basic work-flow of Supervised Learning:

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

4. Algorithms

Random Forest

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

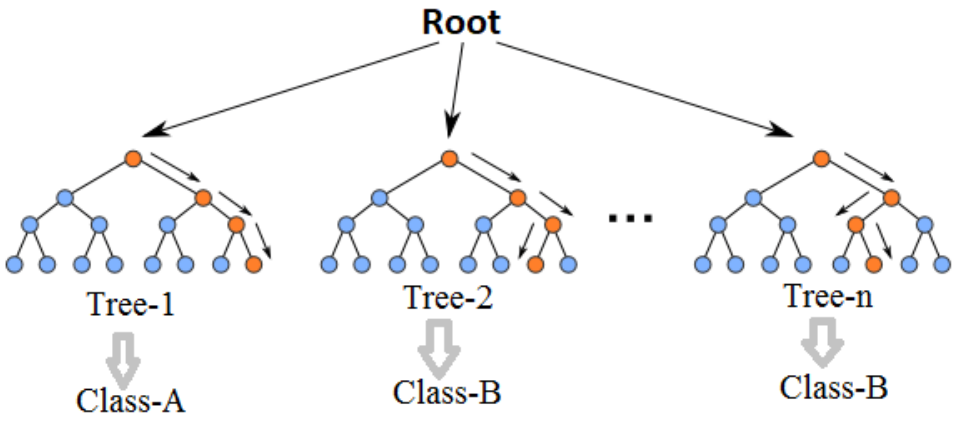
Definition: A random forest is a classifier consisting of a collection of trees structured classifiers $h(x,\Theta_k),k = 1, \ldots$ where the Θ_k are independent identically distributed (i.i.d) random vectors and each tree casts a unit vote for the most popular class at input.

Random Forest Algorithm:

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

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

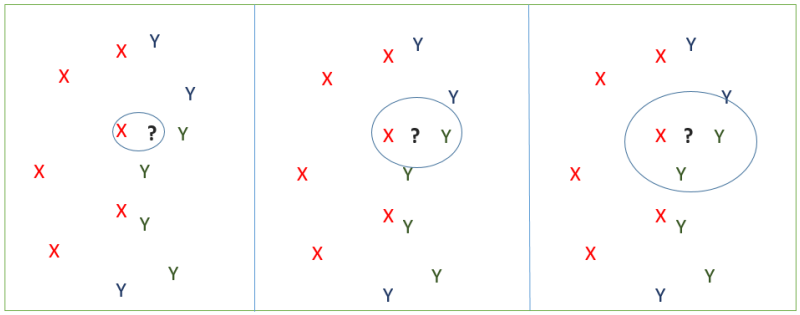
What ensemble does is take the mode (maximum occurring class) of the output produced by n different trees to create a better model. To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.



K-Nearest Neighbours

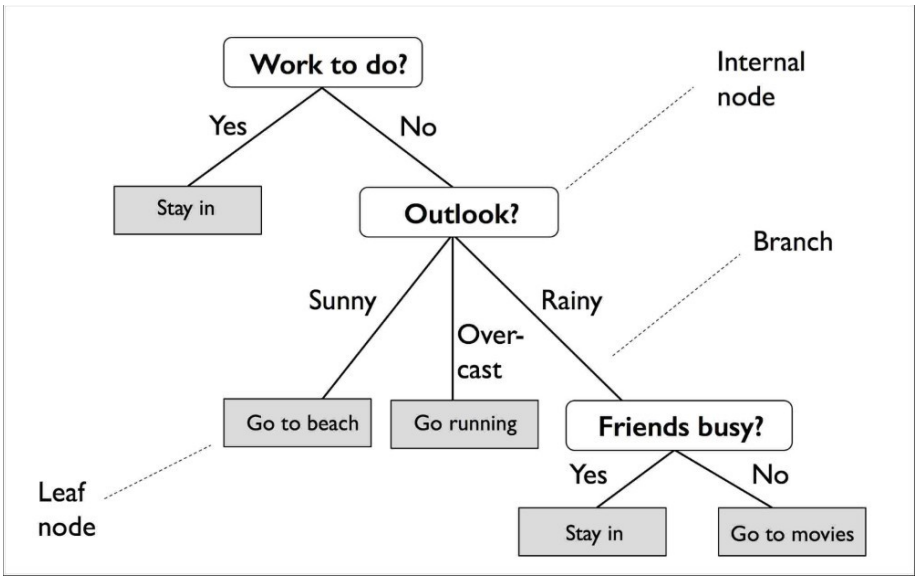
The k-nearest neighbour (k-NN) algorithm is one of the data mining techniques considered to
be among the top 10 techniques for data mining. The k-NN method uses the well-known principle of Cicero pares cum paribus facillime congregant (birds of a feather flock together or literally equals with equals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

sample. Therefore, the unknown sample may be classified based on the classification of this nearest neighbour. The figure shows the k-NN decision rule for k = 1, k = 2 and k = 3 for a set of samples



DECISION TREE REGRESSOR

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.



5.IMPLEMENTATION CODE

Importing Libraries

In [1]:

```
import re
import sys

import time
import datetime

import numpy as np
import pandas as pd
import statistics

import seaborn as sns
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn import tree
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
%matplotlib inline

# Loading the data
df = pd.read_csv('googleplaystorecleaned.csv')
```

In [2]:

```
df.head()
```

Out[2]:

In [3]:

```
df.describe()
```

Out[3]:

Data Cleaning

In [4]:

```
plt.figure(figsize=(5, 3))
sns.heatmap(df.isnull(), cmap='viridis')
df.isnull().any()
```

Out[4]:

```
App                False
Category           False
Rating              True
Reviews            False
Size               False
Installs           False
Type                True
Price              False
Content Rating      True
Genres             False
Last Updated       False
Current Ver         True
Android Ver         True
dtype: bool
```

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
App                   0
Category              0
Rating             1474
Reviews               0
Size                  0
Installs              0
Type                  1
Price                 0
Content Rating        1
Genres                0
Last Updated          0
Current Ver           8
Android Ver           3
dtype: int64
```

In [6]:

```
# Filling missing values using the median instead of mean because the data
is right skewed
df['Rating'] = df['Rating'].fillna(df['Rating'].median())

# Cleaning all non numerical values & unicode charachters
replaces = [u'\u00AE', u'\u2013', u'\u00C3', u'\u00E3', u'\u00B3', '[', ']'
, "'"]
for i in replaces:
    df['Current Ver'] = df['Current Ver'].astype(str).apply(lambda x : x.re
place(i, ''))

regex = [r'[-+|/:/;(_)@]', r'\s+', r'[A-Za-z]+']
for j in regex:
    df['Current Ver'] = df['Current Ver'].astype(str).apply(lambda x : re.s
ub(j, '0', x))
```

```
df['Current Ver'] = df['Current Ver'].astype(str).apply(lambda x : x.replac
e('.', ',',1).replace('.', '').replace(',', '.',1)).astype(float)
df['Current Ver'] = df['Current Ver'].fillna(df['Current Ver'].median())
```

In [7]:

```
# Counting the number of unique values in category column
df['Category'].unique()
```

Out[7]:

```
array(['ART_AND_DESIGN', 'AUTO_AND_VEHICLES', 'BEAUTY',
       'BOOKS_AND_REFERENCE', 'BUSINESS', 'COMICS', 'COMMUNICATION',
       'DATING', 'EDUCATION', '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', 'PERSONALIZATION', 'PRODUCTIVITY', 'PARENTING', 'WEATHER',
       'VIDEO_PLAYERS', 'NEWS_AND_MAGAZINES', 'MAPS_AND_NAVIGATION',
       '1.9'], dtype=object)
```

In [8]:

```
# Check the record  of unreasonable value which is 1.9
i = df[df['Category'] == '1.9'].index
df.loc[i]
```

Out[8]:

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content Rating	Genres	Last Updated	Current Ver	Android Ver
10472	Life Made WI-Fi Touchscreen Photo Frame	1.9	19.0	3.0M	1,000+	Free	0	Everyone	NaN	February 11, 2018	1.0.19	4.0	NaN

In [9]:

```
# Drop this bad column
df = df.drop(i)
```

In [10]:

```
# Removing NaN values
df = df[pd.notnull(df['Last Updated'])]
df = df[pd.notnull(df['Content Rating'])]
```

Label Encoding

In [11]:

```
# App values encoding
le = preprocessing.LabelEncoder()
df['App'] = le.fit_transform(df['App'])
```
In [12]:

```
# Category features encoding
category_list = df['Category'].unique().tolist()
category_list = ['cat_' + word for word in category_list]
df = pd.concat([df, pd.get_dummies(df['Category'], prefix='cat')], axis=1)
```
In [13]:

```
# Genres features encoding
le = preprocessing.LabelEncoder()
df['Genres'] = le.fit_transform(df['Genres'])
```
In [14]:

```
# Encode Content Rating features
le = preprocessing.LabelEncoder()
df['Content Rating'] = le.fit_transform(df['Content Rating'])
```
In [15]:

```
# Price cealning by removing dollar
df['Price'] = df['Price'].apply(lambda x : x.strip('$'))
```
In [16]:

```
# Installs cealning
df['Installs'] = df['Installs'].apply(lambda x : x.strip('+').replace(',',
''))
```
In [17]:

```
# Type encoding
df['Type'] = pd.get_dummies(df['Type'])
```
In [18]:

```
# Last Updated encoding
df['Last Updated'] = df['Last Updated'].apply(lambda x : time.mktime(dateti
me.datetime.strptime(x, '%B %d, %Y').timetuple()))
```
In [19]:

```
# Convert kbytes to Mbytes
k_indices = df['Size'].loc[df['Size'].str.contains('k')].index.tolist()
converter = pd.DataFrame(df.loc[k_indices, 'Size'].apply(lambda x: x.strip(
'k')).astype(float).apply(lambda x: x / 1024).apply(lambda x: round(x, 3)).
astype(str))
df.loc[k_indices,'Size'] = converter
```
In [20]:

```
# Size cleaning
df['Size'] = df['Size'].apply(lambda x: x.strip('M'))
df[df['Size'] == 'Varies with device'] = 0
df['Size'] = df['Size'].astype(float)
```
In [21]:

```
df.head()
```
Out[21]:

In [22]:

```
df.describe()
```

Out[22]:

In [23]:

```
sns.distplot(df.Rating)
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b62272d278>
```

Splitting into train and test

In [24]:

```
# Split data into training and testing sets
features = ['App', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content
 Rating', 'Genres', 'Last Updated', 'Current Ver']
features.extend(category_list)
X = df[features]
y = df['Rating']
```

In [25]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
 random_state = 10)
```

KNearest Neighbors Regression

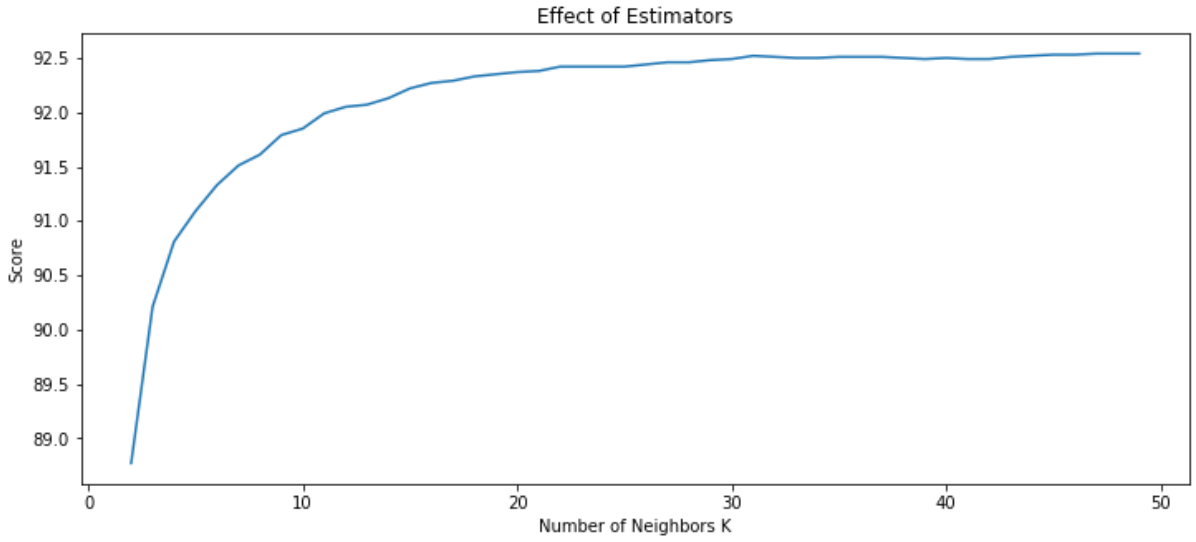
In [26]:

```
# Look at the 2:100 closest neighbors
x=[] #To store predictions scores
j=[] #To store the value of neighbors

for i in range(2,50):
    model = KNeighborsRegressor(n_neighbors=i)
    model.fit(X_train, y_train)
    accuracy = model.score(X_test,y_test)
    x.append(np.round(accuracy*100, 2))
    j.append(i)

print("Maximum accuracy : ",max(x),"% for",j[x.index(max(x))],"neighbors")
plt.figure(figsize=(12, 5))
plt.plot(j,x)
plt.title("Effect of Estimators")
plt.xlabel("Number of Neighbors K")
plt.ylabel("Score")
plt.show()
```

Maximum accuracy : 92.54 % for 47 neighbors



Random Forest Regressor

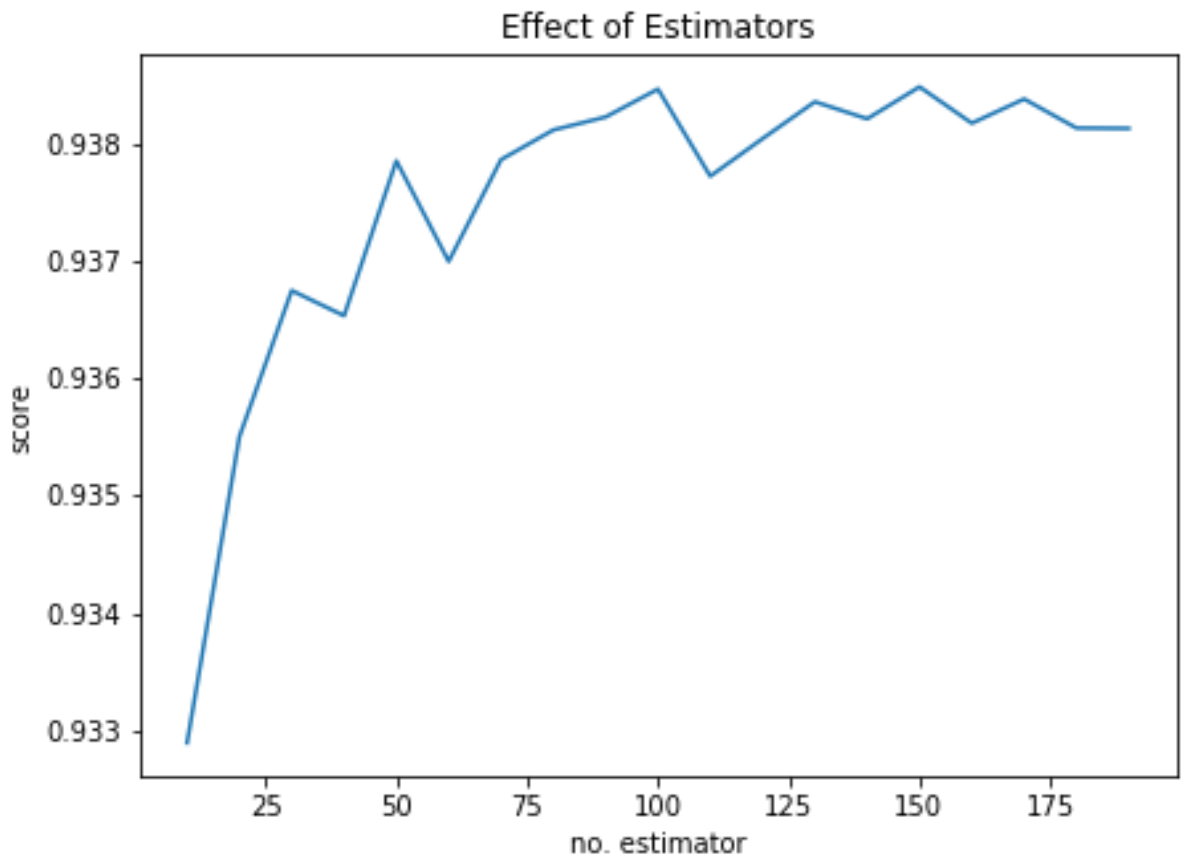
In [27]:

```
model = RandomForestRegressor(n_jobs=-1)
# Trying different numbers of n_estimators
estimators = np.arange(10, 200, 10)
scores = [] # to store scores of predictions
trees=[] # to store the number of trees

for n in estimators:
    model.set_params(n_estimators=n)
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))
    trees.append(n)

print("Maximum Accuracy :",np.round(max(scores)*100,2),"% for",trees[scores
.index(max(scores))],"trees")
plt.figure(figsize=(7, 5))
plt.title("Effect of Estimators")
plt.xlabel("no. estimator")
plt.ylabel("score")
plt.plot(estimators, scores)
plt.show()
```

Maximum Accuracy : 93.85 % for 150 trees



In [28]:

```
predictions = model.predict(X_test)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictio
ns))
```

```
Mean Absolute Error: 0.24158613322975353
```

In [29]:

```
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions
))
```

```
Mean Squared Error: 0.16144781500751293
```

In [30]:

```
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test
, predictions)))

#HERE RMS VALUE IS GREATER THAN EXPECTED BECAUSE ROOT OF NUMBER LYING BETWE
EN 0 AND 1 IS GREATER THAN THE NUMBER ITSELF!!!
```

```
Root Mean Squared Error: 0.40180569309992725
```

DECISION TREE REGRESSOR

In [42]:

```
clf = tree.DecisionTreeRegressor(criterion='mae', max_depth=5, min_samples_
leaf=5, random_state=42)
clf.fit(X_train, y_train)
accuracy = clf.score(X_test, y_test)
```

```
print("Accuracy:"+str(np.round(accuracy*100))+"%")
```

```
Accuracy:93.0%
```

In [46]:

```
objects = ("KNeighbors Classifier","Random Forest Regressor","Decision Tree
 Regressor")
y_pos = np.arange(len(objects))
performance = [92.54,93.85,93.0]

plt.barh(y_pos, performance, align='center', alpha=0.5, color="purple")
plt.yticks(y_pos, objects)
plt.xlabel("Accuracy")
plt.title("Algorithm Name")

plt.show()
```

Visualisations

In [31]:

```
c=[]
for x in df.Category:
    if type(x)==str:
        c.append(x)
plt.subplots(figsize=(10,10))
text = " ".join(cat for cat in c)
print ("There are {} words in the combination of all review.".format(len(te
xt)))
stopwords = set(STOPWORDS)
wordcloud = WordCloud(stopwords=stopwords, background_color="white", colloc
ations=False).generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

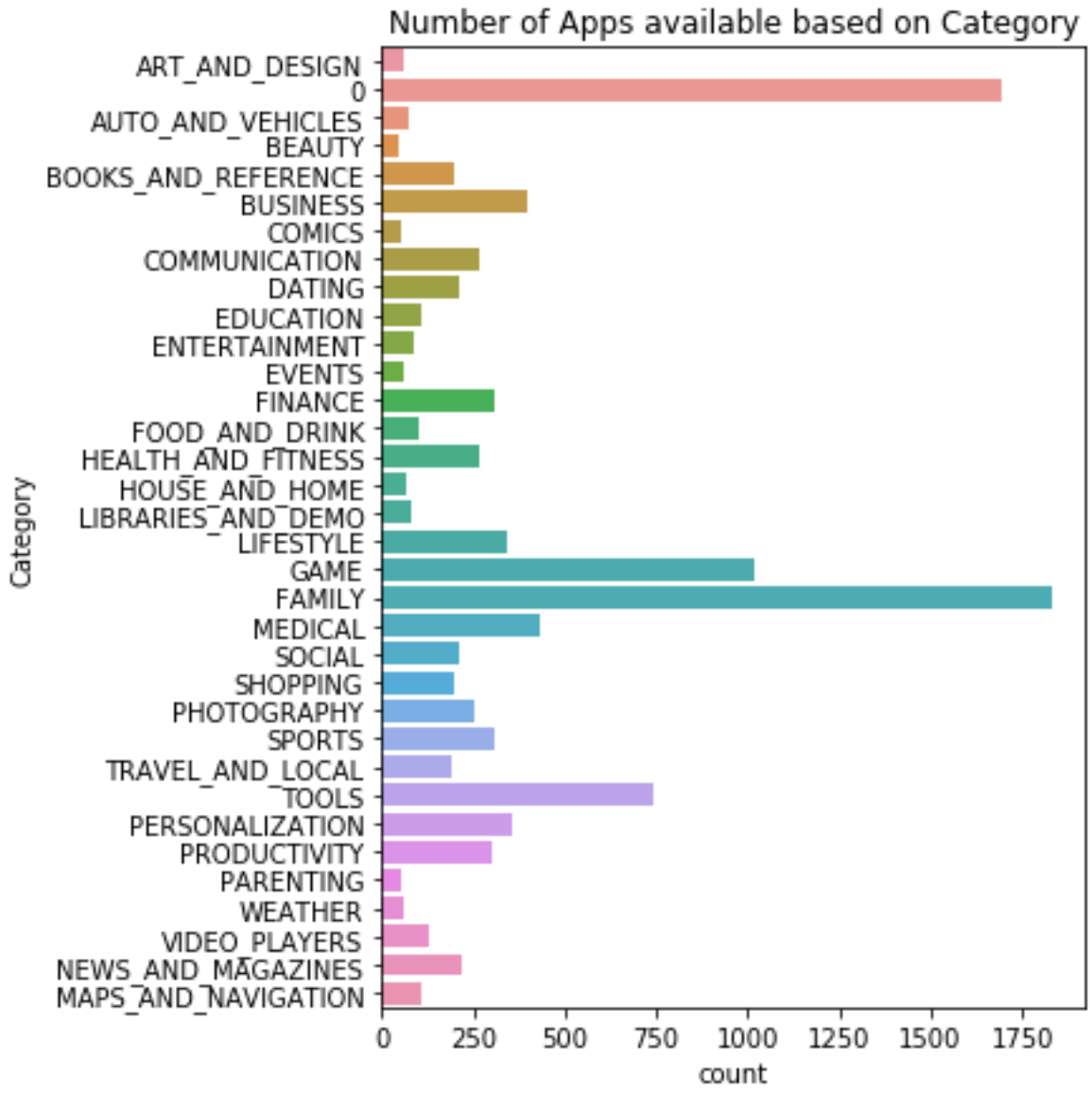
```
There are 89754 words in the combination of all review.
```

In [32]:

```
#Checking the number of Apps available on playstore based on category
plt.figure(figsize=(5,7))
sns.countplot(y='Category',data=df)
plt.title("Number of Apps available based on Category")
```

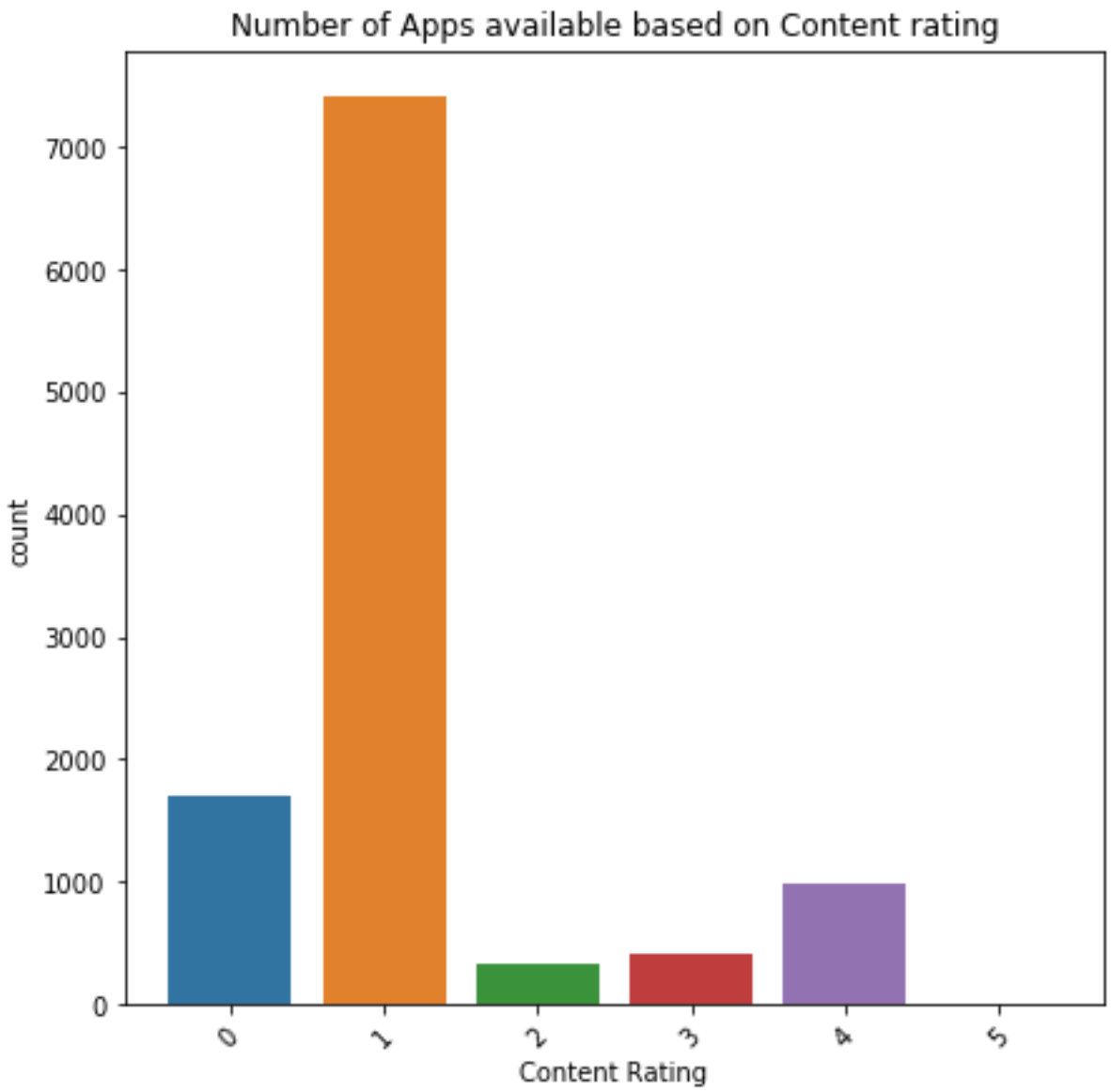
Out[32]:

```
Text(0.5, 1.0, 'Number of Apps available based on Category')
```



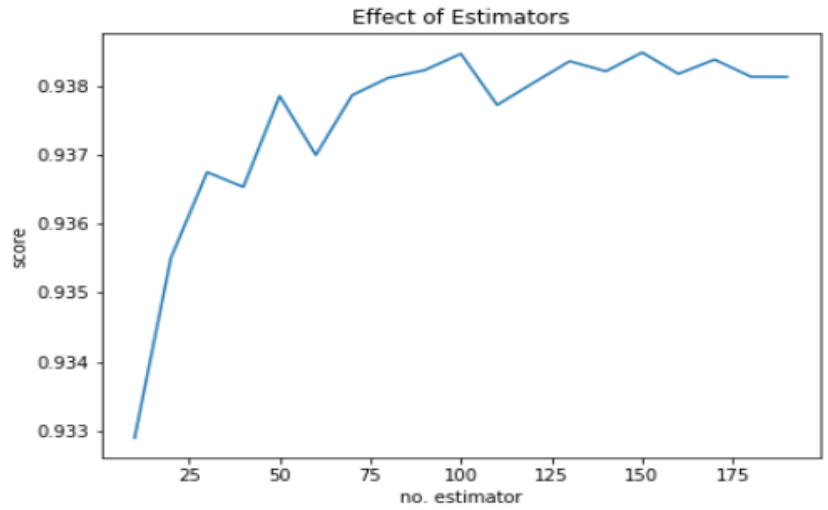
In [34]:

```
#Apps available based on Content rating
plt.figure(figsize=(7,7))
sns.countplot(x='Content Rating',data=df,)
plt.xticks(rotation=45)
plt.title("Number of Apps available based on Content rating")
plt.plot()
plt.show()
```

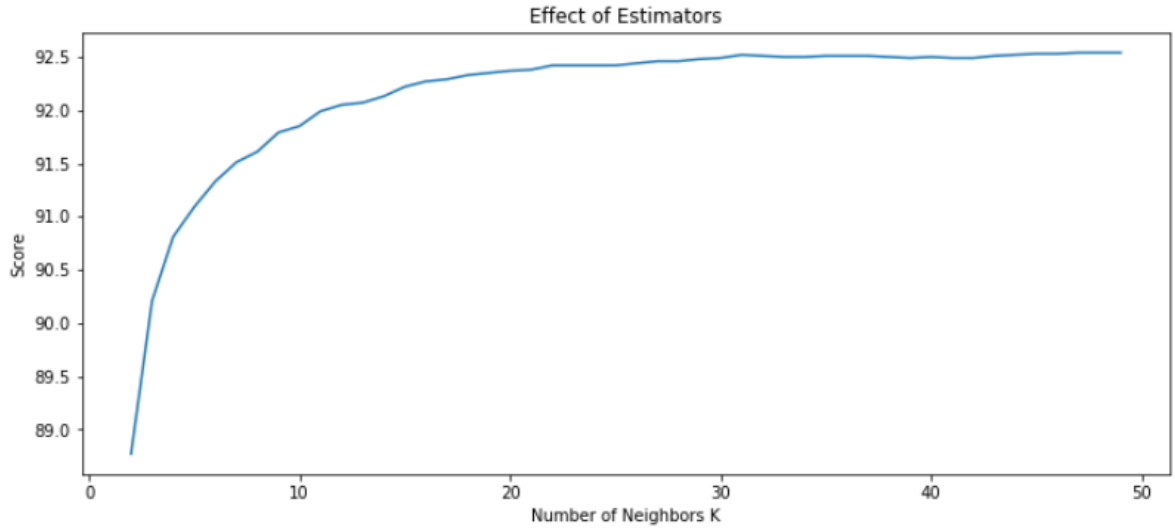


Effect of K estimators for Random Tree Regressor:

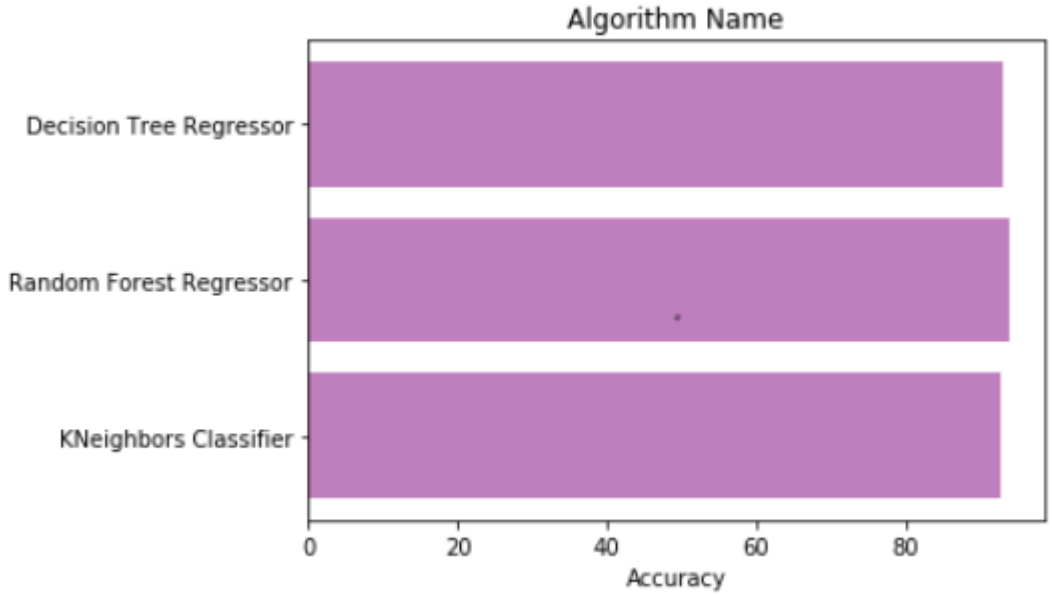
```
Maximum Accuracy : 93.85 % for 150 trees
```



```
Maximum accuracy :  92.54 % for 47 neighbors
```



6. CONCLUSION



The algorithms namely KNeighbor, Random Forest Classifier and Decision Tree Regressor were used to fit the dataset and predict the results. The **Random Forest Regressor** proved to be the best suited algorithm for this dataset with an accuracy of **93.85%** followed by **Decision Tree Regressor** and **KNeighbor** with accuracies of **93.00%** and **92.59%** respectively.

The Random Forest regressor gives the best accuracy but lags behind in training time since around 150 trees were made in this process to achieve this accuracy. On the other hand, the Decision Tree Classifier proves to be the fastest with quite acceptable accuracy of 93.0%