# APP SUCCESS PREDICTION ON PLAY STORE

## MACHINE LEARNING PROJECT

**SUBMITTED BY-**

BHAVYA - 101603076

CHIRAG MAHAWAR - 101603078

COE-6

**To-**

**Dr. Parteek Bhatia**

**Asst. Professor,CSED**

**Thapar Institute of Engg & Technology**

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

**Date: 28-11-2018**

# TABLE OF CONTENTS

# ABSTRACT

This project aims to analyse data about various applications available on Google Play Store. Based on data like application category, size, price, number of installs, content rating, review count , reviews, prediction of  how successful an android application will be on the Google Play Store is made. This is achieved by predicting the likely application rating on google play store. For doing this, Linear Regression model, SVM model and Random Forest regression model are used for predicting the rating. Also models are  evaluated by comparing the predicted results against the actual results by the use of mean squared error & mean absolute error.

# INTRODUCTION

## NEED OF THE SYSTEM

For developing a good android application, it is better to be aware of the characteristics that makes an application successful on that platform. This system helps one know how well their application will work on Google Play Store based on features of the application and what improvements can be made to make that application a hit on Playstore platform. It will also help developers in improving existing applications to achieve higher customer satisfaction levels and better reviews and ratings on Play Store.

## APPLICATIONS OF PROPOSED SYSTEM

- It can be used to predict rating of an application available on Google Play Store, based on current ratings of other applications.
- It can be used to predict the success of a new application on Google Play Store. One can simply add this new application's details in the testing set and get the results.

## CHALLENGES IN DEVELOPMENT

- The columns 'category' and 'genre' store almost the same data. If two explanatory variables in a model are highly linearly related, it poses a problem called multicollinearity. Together, these columns have nearly the same effect on the final result. So considering them both can affect the result. Therefore, we dropped 'genre' column from the dataset.
- The dataset contained columns like 'Last Updated', 'Current version', 'Android Version', which do not play any part in the app ratings on Play Store. So we dropped these columns by using dataset.drop (labels=[]) function.
- In data preprocessing stage, an error was incurred because of the rows which had NULL values. Thus, we applied 'dataset.dropna()' function to remove the rows which had NULL values in them.
- We encountered an error of approximately 65% while using SVM model. It was so because we were initially performing feature scaling in SVM model. We overcame this error by removing feature scaling and re-applying the model. Without feature scaling, an error of approximately 20% was there.

# WORKING OF PROPOSED SYSTEM

Proposed system uses Machine learning algorithms to predict the rating of the application of google play store based on their features. Branch of Machine learning used here is supervised Learning which needs a human to "supervise" and tell the computer what it should be trained to predict for, or give it the right answer. We feed the computer with training data containing various features, and we also tell it the right answer. Supervised learning can solve two problems- Classification & Regression. For the said problem, regression is used so as to predict application rating. Machine learning problem in supervised learning can be solved in three stages which are - Data Preparation, Training & Testing, & evaluation of used models.

For the regression problem, most commonly used regression models are used which are - Linear Regression, SVM Model & Random forest regression model.

Linear regression is the simplest of regression model which is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). Support vector machines (SVMs, also support vector networks[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. When used alone, decision trees are prone to overfitting. However, random forests help by correcting the possible overfitting that could occur. Random forests work by using multiple decision trees — using a multitude of different decision trees with different predictions, a random forest combines the results of those individual trees to give the final outcomes.

# DATA COLLECTION AND DATA PREPARATION

Dataset for the said problem was collected from Kaggle & contains columns as shown-

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Ra | Genres | Last Updat | Current Ve | Android Ver | |
| 2 | Photo Edit | ART_AND_ | 4.1 | 159 | 19M | 10,000+ | Free | 0 | Everyone | Art & Desi | January 7, | 1.0.0 | 4.0.3 and up | |
| 3 | Coloring b | ART_AND_ | 3.9 | 967 | 14M | 500,000+ | Free | 0 | Everyone | Art & Desi | January 15 | 2.0.0 | 4.0.3 and up | |
| 4 | U Launche | ART_AND_ | 4.7 | 87510 | 8.7M | 5,000,000+ | Free | 0 | Everyone | Art & Desi | August 1, 2 | 1.2.4 | 4.0.3 and up | |
| 5 | Sketch - Dr | ART_AND_ | 4.5 | 215644 | 25M | 50,000,000 | Free | 0 | Teen | Art & Desi | June 8, 201 | Varies with | 4.2 and up | |
| 6 | Pixel Draw | ART_AND_ | 4.3 | 967 | 2.8M | 100,000+ | Free | 0 | Everyone | Art & Desi | June 20, 20 | 1.1 | 4.4 and up | |
| 7 | Paper flow | ART_AND_ | 4.4 | 167 | 5.6M | 50,000+ | Free | 0 | Everyone | Art & Desi | March 26, | 1 | 2.3 and up | |
| 8 | Smoke Eff | ART_AND_ | 3.8 | 178 | 19M | 50,000+ | Free | 0 | Everyone | Art & Desi | April 26, 20 | 1.1 | 4.0.3 and up | |
| 9 | Infinite Pai | ART_AND_ | 4.1 | 36815 | 29M | 1,000,000+ | Free | 0 | Everyone | Art & Desi | June 14, 20 | 6.1.61.1 | 4.2 and up | |
| 10 | Garden Co | ART_AND_ | 4.4 | 13791 | 33M | 1,000,000+ | Free | 0 | Everyone | Art & Desi | September | 2.9.2 | 3.0 and up | |
| 11 | Kids Paint | ART_AND_ | 4.7 | 121 | 3.1M | 10,000+ | Free | 0 | Everyone | Art & Desi | July 3, 201 | 2.8 | 4.0.3 and up | |
| 12 | Text on Ph | ART_AND_ | 4.4 | 13880 | 28M | 1,000,000+ | Free | 0 | Everyone | Art & Desi | October 2 | 1.0.4 | 4.1 and up | |
| 13 | Name Art | ART_AND_ | 4.4 | 8788 | 12M | 1,000,000+ | Free | 0 | Everyone | Art & Desi | July 31, 20 | 1.0.15 | 4.0 and up | |
| 14 | Tattoo Na | ART_AND_ | 4.2 | 44829 | 20M | 10,000,000 | Free | 0 | Teen | Art & Desi | April 2, 201 | 3.8 | 4.1 and up | |
| 15 | Mandala C | ART_AND_ | 4.6 | 4326 | 21M | 100,000+ | Free | 0 | Everyone | Art & Desi | June 26, 20 | 1.0.4 | 4.4 and up | |
| 16 | 3D Color P | ART_AND_ | 4.4 | 1518 | 37M | 100,000+ | Free | 0 | Everyone | Art & Desi | August 3, 2 | 1.2.3 | 2.3 and up | |
| 17 | Learn To D | ART_AND_ | 3.2 | 55 | 2.7M | 5,000+ | Free | 0 | Everyone | Art & Desi | June 6, 201 | NaN | 4.2 and up | |
| 18 | Photo Des | ART_AND_ | 4.7 | 3632 | 5.5M | 500,000+ | Free | 0 | Everyone | Art & Desi | July 31, 20 | 3.1 | 4.1 and up | |
| 19 | 350 Diy Ro | ART_AND_ | 4.5 | 27 | 17M | 10,000+ | Free | 0 | Everyone | Art & Desi | November | 1 | 2.3 and up | |
| 20 | FlipaClip - | ART_AND_ | 4.3 | 194216 | 39M | 5,000,000+ | Free | 0 | Everyone | Art & Desi | August 3, 2 | 2.2.5 | 4.0.3 and up | |
| 21 | ibis Paint X | ART_AND_ | 4.6 | 224399 | 31M | 10,000,000 | Free | 0 | Everyone | Art & Desi | July 30, 20 | 5.5.4 | 4.1 and up | |
| 22 | Logo Make | ART_AND_ | 4 | 450 | 14M | 100,000+ | Free | 0 | Everyone | Art & Desi | April 20, 20 | 4 | 4.1 and up | |
| 23 | Boys Phot | ART_AND_ | 4.1 | 654 | 12M | 100,000+ | Free | 0 | Everyone | Art & Desi | March 20, | 1.1 | 4.0.3 and up | |
| 24 | Superhero | ART_AND_ | 4.7 | 7699 | 4.2M | 500,000+ | Free | 0 | Everyone 1 | Art & Desi | July 12, 20 | 2.2.6.2 | 4.0.3 and up | |

Columns in the dataset are explained-

- App
  Application name
- Category
  Category the app belongs to
- Rating
  Overall user rating of the app
- Reviews
  Number of user reviews for the app
- Size
  Size of the app
- Installs
  Number of user downloads/installs for the app
- Type
  Paid or Free
- Price
  Price of the app
- Content Rating
  Age group the app is targeted at - Children / Mature 21+ / Adult

- Genres
  An app can belong to multiple genres
- Last Updated
  Date when the app was last updated on Play Store
- Current Ver
  Current version of the app available on Play Store
- Android Ver
  Min required Android version

Dataset contained many null values so our first step was to remove those rows with null values in it. Also the dataset contained many information that are irrelevant in predicting the rating of app. Thus, second step is to remove those unnecessary & unrelated column.

```
#remove missing values from dataset
dataset.dropna(inplace = True)

#dropping of unrelated and unnecessary columns from the dataset
dataset.drop(labels = ['Last Updated','Current Ver','Android Ver','App','Genres'], axis = 1, inplace = True)

```

Categories column contains more than one value & also it was in the string format. For any machine learning model to be applied, dataset must be in the format of integers. So the next step is to clean the Category column.

```

# Cleaning Categories into integers
CategoryString = dataset["Category"]
categoryVal = dataset["Category"].unique()
categoryValCount = len(categoryVal)
category_dict = {}
for i in range(0,categoryValCount):
    category_dict[categoryVal[i]] = i
dataset["Category_c"] = dataset["Category"].map(category_dict).astype(int)

#cleaning size of installation
```

Next step is to clean the size of installation column as it contains the values in KB & MB, so converting the all the values in bytes will be much easier for the dataset to be processed under any machine learning model.

```
#cleaning size of installation
def change_size(size):
    if 'M' in size:
        x = size[:-1]
        x = float(x)*1000000
        return(x)
    elif 'k' == size[-1:]:
        x = size[:-1]
        x = float(x)*1000
        return(x)
    else:
        return None

dataset["Size"] = dataset["Size"].map(change_size)
#filling Size which had NA
dataset.Size.fillna(method = 'ffill', inplace = True)

```

Number of installs column contains the value in the form of x+ which denotes no of installs of said app is greater than x. Thus cleaning of installs column as-

```
#Cleaning no of installs column
dataset['Installs'] = [int(i[:-1].replace(',','')) for i in dataset['Installs']]

```

Type of app denotes whether the app is free or paid which is binary values so, converting those values in 0 &1.

```
#Converting Type column into binary column
def type_cat(types):
    if types == 'Free':
        return 0
    else:
        return 1

dataset['Type'] = dataset['Type'].map(type_cat)
```

Also, Price column depicts the price of application on the play store, whose value is either 0 for free app & some amount in dollars for paid apps. Converting those amounts in float values as-

```
#Cleaning prices
def price_clean(price):
    if price == '0':
        return 0
    else:
        price = price[1:]
        price = float(price)
        return price

dataset['Price'] = dataset['Price'].map(price_clean).astype(float)
```

Now, the first step for any machine learning algorithm completes. Dataset is preprocessed & is ready to be applied on regression model.

# TRAINING & TESTING OF MODEL

For training of our selected models- Linear Regression,SVM, Random Forest Regression models.

## LINEAR REGRESSION

```python

"""Step 2 - Training & Testing of the model
Model 1 : Linear Regression"""

#splitting the dataset into training & test set

X = dataset.iloc[:, 1:].values
y = dataset.iloc[:, 0].values

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size = 0.3 , random_state = 0)

#fitting simple linear regression into training set

from sklearn.linear_model import LinearRegression
linear_regressor = LinearRegression()
linear_regressor.fit(X_train , y_train)

#predicting the test set results
y_pred = linear_regressor.predict(X_test)
```

## SVM REGRESSION (With Feature Scaling)

Firstly, Feature scaling is applied on the dataset in order to scale the data in all columns. Some machine learning algorithms uses Euclidean Distance between the features for training purpose. With wide range in the values of column, sometimes result gets purely dependent on the column with larger values.

```python

"""Step 2 - Training & Testing of the model
Model 2 : SVM Regression(with feature scaling)"""

X = dataset.iloc[:, 1:].values
y = dataset.iloc[:, 0].values

#feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = y.reshape(-1, 1)
y = sc_y.fit_transform(y)

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size = 0.3 , random_state = 0)

#fitting the SVR model to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')   #chhosing the gaussian kernel..ie rbf kernel
regressor.fit(X_train , y_train)

y_pred_svm = regressor.predict(X_test)

```

## SVM REGRESSION (Without Feature Scaling)

```
"""Step 2 - Training & Testing of the model
Model 2 : SVM Regression(without feature scaling)"""

X = dataset.iloc[:, 1:].values
y = dataset.iloc[:, 0].values

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size = 0.3 , random_state = 0)

#fitting the SVR model to the dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')   #chhosing the gaussian kernel..ie rbf kernel
regressor.fit(X_train , y_train)

y_pred_svm = regressor.predict(X_test)
```

## RANDOM FOREST REGRESSION

```
"""Step 2 - Training & Testing of the model
Model 2 : Random forest Regression"""

X = dataset.iloc[:, 1:].values
y = dataset.iloc[:, 0].values

from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size = 0.3 , random_state = 0)

#fitting the Random forest regression model to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 100 , random_state = 0)   #n_estimators is no of trees in forest
regressor.fit(X_train , y_train)

#predicting the new result with polynomial regression
y_pred = regressor.predict(X_test)

#random forest regression with 300 trees
from sklearn.ensemble import RandomForestRegressor
regressor_2 = RandomForestRegressor(n_estimators = 300 , random_state = 0)   #n_estimators is no of trees in fore
regressor_2.fit(X_train , y_train)

#predicting the new result with polynomial regression
y_pred_2 = regressor_2.predict(X_test)
```

# RESULTS AND DISCUSSIONS

## LINEAR REGRESSION

```
def Evaluationmatrix(y_test, y_pred):
    print ('Mean Squared Error: '+ str(metrics.mean_squared_error(y_test,y_pred)))
    print ('Mean absolute Error: '+ str(metrics.mean_absolute_error(y_test,y_pred)))

Evaluationmatrix(y_test,y_pred)
```

IPython console

Preprocessing/A | Linear_R/A | SVM_R1/A | SVM_R2/

```
   ...: linear_regressor.fit(X_train , y_train)
Out[3]: LinearRegression(copy_X=True, fit_intercept=

In [4]: y_pred = linear_regressor.predict(X_test)

In [5]: def Evaluationmatrix(y_test, y_pred):
   ...:     print ('Mean Squared Error: '+
str(metrics.mean_squared_error(y_test,y_pred)))
   ...:     print ('Mean absolute Error: '+
str(metrics.mean_absolute_error(y_test,y_pred)))

In [6]: Evaluationmatrix(y_test,y_pred)
Mean Squared Error: 0.26077708107509806
Mean absolute Error: 0.3528789627513037
```

After the Linear regression model is trained, it is then tested on test set. Predicted results & actual results are compared using the evaluation parameter for regression model such as mean squared error & mean absolute error. As shown in the picture,

Mean squared error - 0.2607

Mean absolute error - 0.3528

## SVM REGRESSION (With Feature Scaling)

```python

def Evaluationmatrix(y_test, y_pred):
    print ('Mean Squared Error: '+ str(metrics.mean_squared_error(y_test,y_pred)))
    print ('Mean absolute Error: '+ str(metrics.mean_absolute_error(y_test,y_pred)))

Evaluationmatrix(y_test,y_pred_svm)


```

IPython console

Preprocessing/A | Linear_R/A | SVM_R1/A | SVM_R2/A

```
In [6]: y_pred_svm = regressor.predict(X_test)

In [7]: def Evaluationmatrix(y_test, y_pred):
   ...:     print ('Mean Squared Error: '+
str(metrics.mean_squared_error(y_test,y_pred)))
   ...:     print ('Mean absolute Error: '+
str(metrics.mean_absolute_error(y_test,y_pred)))
   ...:
   ...:
   ...: Evaluationmatrix(y_test,y_pred_svm)
Mean Squared Error: 1.0190424013531676
Mean absolute Error: 0.654663949977556

In [8]:
```

After the SVM model is trained, it is then tested on test set. Predicted results & actual results are compared using the evaluation parameter for regression model such as mean squared error & mean absolute error. In SVM model here, firstly feature scaling is used for scaling the all the values in columns. As shown in the picture,

Mean squared error - 1.0190

Mean absolute error - 0.6546

## SVM REGRESSION (Without Feature Scaling)

```python
def Evaluationmatrix(y_test, y_pred):
    print ('Mean Squared Error: '+ str(metrics.mean_squared_error(y_test,y_pred)))
    print ('Mean absolute Error: '+ str(metrics.mean_absolute_error(y_test,y_pred)))

Evaluationmatrix(y_test,y_pred_svm)
```

```
IPython console
Preprocessing/A   Linear_R/A   SVM_R1/A   SVM_R2/A

In [5]: y_pred_svm = regressor.predict(X_test)

In [6]: def Evaluationmatrix(y_test, y_pred):
   ...:     print ('Mean Squared Error: '+
str(metrics.mean_squared_error(y_test,y_pred)))
   ...:     print ('Mean absolute Error: '+
str(metrics.mean_absolute_error(y_test,y_pred)))
   ...:
   ...:
   ...: Evaluationmatrix(y_test,y_pred_svm)
Mean Squared Error: 0.26232867994406256
Mean absolute Error: 0.34716344856088266
```

After the SVM model is trained, it is then tested on test set. Predicted results & actual results are compared using the evaluation parameter for regression model such as mean squared error & mean absolute error. In SVM model here, feature scaling is not used. As shown in the picture,

Mean squared error - 0.2623

Mean absolute error - 0.3471

## RANDOM FOREST REGRESSION

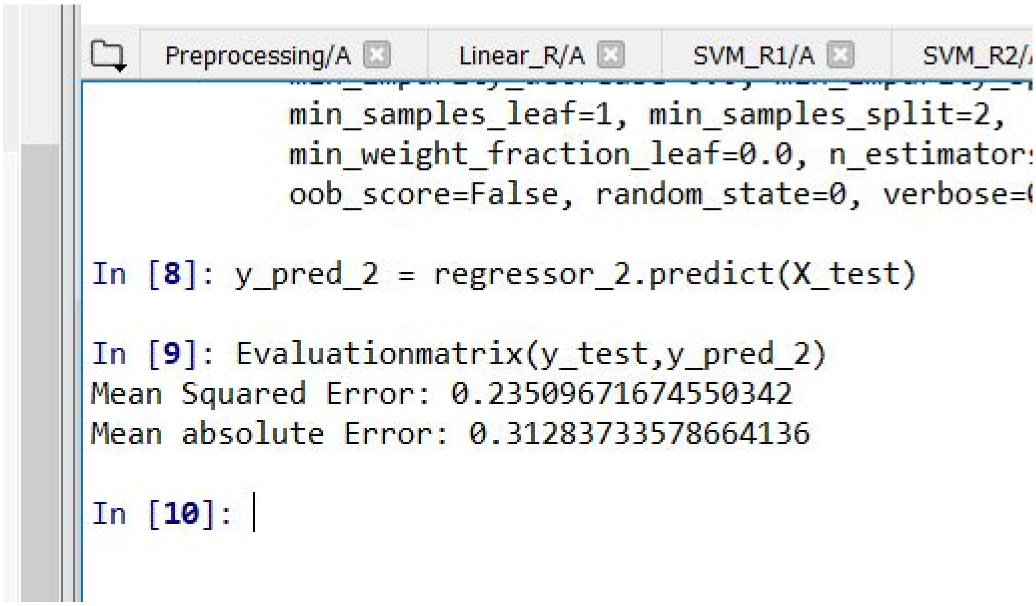
```python
def Evaluationmatrix(y_test, y_pred):
    print ('Mean Squared Error: '+ str(metrics.mean_squared_error(y_test,y_pred)))
    print ('Mean absolute Error: '+ str(metrics.mean_absolute_error(y_test,y_pred)))

Evaluationmatrix(y_test,y_pred)
Evaluationmatrix(y_test,y_pred_2)
```

```
IPython console

Preprocessing/A   Linear_R/A   SVM_R1/A   SVM_R2/A

In [6]: def Evaluationmatrix(y_test, y_pred):
   ...:     print ('Mean Squared Error: '+
str(metrics.mean_squared_error(y_test,y_pred)))
   ...:     print ('Mean absolute Error: '+
str(metrics.mean_absolute_error(y_test,y_pred)))
   ...:
   ...:
   ...: Evaluationmatrix(y_test,y_pred)
Mean Squared Error: 0.23598689141030588
Mean absolute Error: 0.3129456255935423
```

After the Random forest model is trained, it is then tested on test set. Predicted results & actual results are compared using the evaluation parameter for regression model such as mean squared error & mean absolute error. Regression model shown here is trained with 100 trees. As shown in the picture,

Mean squared error - 0.23598

Mean absolute error - 0.31294




After the Random forest model is trained, it is then tested on test set. Predicted results & actual results are compared using the evaluation parameter for regression model such as mean squared error & mean absolute error. Regression model shown here is trained with 300 trees. As shown in the picture,

Mean squared error - 0.23509

Mean absolute error - 0.3128

# CONCLUSIONS

Results produced by all the models are-

### LINEAR REGRESSION

Mean squared error - 0.2607
Mean absolute error - 0.3528

### SVM REGRESSION (With Feature Scaling)

Mean squared error - 1.0190
Mean absolute error - 0.6546

### SVM REGRESSION (Without Feature Scaling)

Mean squared error - 0.2623
Mean absolute error - 0.3471

### RANDOM FOREST REGRESSION

### Trees = 100

Mean squared error - 0.23598
Mean absolute error - 0.31294

### Trees = 300

Mean squared error - 0.23509
Mean absolute error - 0.3128

As observed from the results, Random forest regression model produces better results in comparison of other models. By increasing the no of trees(n_estimators parameter) in random forest regression, increases the performance and makes the predictions more stable. One of the big problems in machine learning is overfitting, but most of the time this won't happen that easy to a random forest model. That's because if there are enough trees in the forest, the regressor won't overfit the model. The main limitation of Random Forest is that a large number of trees can make the algorithm to slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.

# REFERENCES


1. https://www.kaggle.com/lava18/google-play-store-apps
2. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
5. https://scikit-learn.org/stable/modules/preprocessing.html
6. https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd