Rating Prediction on Google Play Store Apps

Yiğit Can Akçay
Dokuz Eylül University
Computer Engineering
İzmir, Türkiye
yigitcan.akcay@ogr.deu.edu.tr

Harun Adem Temur
Dokuz Eylül University
Computer Engineering
İzmir, Türkiye
harunadem.temur@ogr.deu.edu.tr

***Abstract*—A data mining application that helps mobile app developers develop strategies to achieve high app scores, focusing on the challenges of attracting user interest.**

Keywords—rating, data mining, google, model

I. Introduction

App developers often struggle to attract users and maintain their interest over time. However, by understanding the factors associated with high app scores, developers can develop strategies to increase user engagement and create a positive user experience, ultimately contributing to higher scores and increased app usage. The data mining application that will be described in this article uses certain features of apps in the Google App Store to help developers predict how many out of 5 ratings their app will receive.

II. Literature Review

This paper [1] is about similar research on Blackberry App Store, about understanding the relations between the apps' price, their ratings, popularity, etc. While their goal is to examine the effects of those items specifically and focused on Blackberry App Store, we're doing our research and predictions in Google App Store, which is more comprehensive. We also use a more general approach and use other attributes of the apps for our research and prediction model.

Another study [2] that is done on the Google Play Store involves applying sentiment analysis on the reviews to try to predict the numeric ratings of the apps. We believe that using other numeric and categorical features and the data existing on Play Store could help us predict the ratings as well.

III. Dataset

The dataset used to build the model has 10841 rows and 13 columns. Some of these columns contain null values. Also, some columns are not suitable for use. The solution of such situations will be explained in the following sections. The first column of the data set is the 'App' column. It contains unique values. The data is of discrete type. The data in the 'Category', 'Content Rating', 'Genres', 'Current Ver' and 'Android Ver' columns are nominal. Although the 'Reviews', 'Size', 'Installs' and 'Price' columns contain categorical values, they actually contain continuous values. This transformation will be seen in the data cleaning operations in the next steps. 'Last Updated' column contains interval type data. The 'Size' column contains continuous data type data. In addition, the 'Rating' column was selected as the target feature while creating the model.

```
Data columns (total 13 columns):
 #   Column          Non-Null Count
---  ------          --------------
 0   App             10841 non-null
 1   Category        10841 non-null
 2   Rating          9367 non-null
 3   Reviews         10841 non-null
 4   Size            10841 non-null
 5   Installs        10841 non-null
 6   Type            10840 non-null
 7   Price           10841 non-null
 8   Content Rating  10840 non-null
 9   Genres          10841 non-null
 10  Last Updated    10841 non-null
 11  Current Ver     10833 non-null
 12  Android Ver     10838 non-null
dtypes: float64(1), object(12)
```

Fig. 1. Dataset information table

IV. Data Preparation

In the quest for a well-organized and cleaned dataset, a systematic approach is needed to address various data anomalies and missing values. The following steps outline a comprehensive strategy to improve the quality and consistency of the dataset:

A. Addressing Missing Values with the Median

Using the median instead of the mean to fill in missing values is often a better approach. This approach helps to reduce the impact of outliers and skewed distributions on the imputation process.

B. Standardizing Version Formats

To simplify data representation and facilitate uniform analysis, all versions in the dataset should be converted to numeric format. This not only simplifies the data set but also ensures consistency in the display of version information.

C. *Cleaning Non-numeric and Unicode Characters*

A critical step in data preparation is the removal of non-numeric values and Unicode characters. This process is vital for maintaining the numerical integrity of the data set and supporting a standard format for subsequent analyses.

D. *Analyzing Unique Values in the Category Column*

Understanding the variation in the 'category' column is very important. Counting the number of unique values contributes to a more nuanced analysis by providing information on the breadth and distribution of application categories.

E. *Handling of Irrational Values*

Rigorous examination of records containing seemingly illogical values, for example the number 19 as the average rating of an app, suggests an incomplete data pattern. In such cases where the first value (Application name) is missing, it is wise to drop the entire record. Propagating values backwards from "Category" to "Current Data" risks introducing inaccuracies and jeopardizes the reliability of the dataset.

F. *Strategic Row Dropping*

In cases where certain rows contain data that deviates significantly from the norm of the dataset, it becomes a logical choice to drop the entire row. This not only eliminates irrelevant or erroneous information, but also facilitates subsequent data processing.

G. *Focused NaN Removal*

To further refine the dataset, a targeted approach involves removing NaN values, especially from the 'Last Update' and 'Content Rating' columns. This step contributes to the overall data cleaning and provides a more accurate representation of application information.

H. *Outlier Analysis*

Reliable Outlier analysis is used to detect and correct outliers in numerical columns. It follows the following steps for each numeric column:

Outliers in the column are identified using the IQR (Interquartile Range) method. This method calculates the IQR using Q1 and Q3 values, which are the quartiles of the data. Then, lower, and upper bound values are determined.

After the outliers are detected, ‘clamp transformation’ is applied to bring these values within the bounds. Using the np.where function, outliers are equalized to the lower bound if they are less than the lower bound, and to the upper bound if they are greater than the upper bound.

Finally, the relevant column is updated in the original data frame. This process is applied separately for each numeric column.

Outlier analysis is a process that needs to be carefully considered depending on the characteristics of the data set and the purpose of the analysis. Before using this code, it is important to carry out a thorough examination and testing of the nature of the data set and the way outliers are handled.

By carefully following these steps, the dataset undergoes a thorough cleaning process, providing a more robust and reliable basis for subsequent analyses and modelling.

V. Categorical Data Encoding

Categorical features (strings) need to be converted into numeric features (numbers).

Many machine learning algorithms can support categorical values without further manipulation, but there are many more that do not. We needed to make all the data ready for the model, so we converted categorical variables (variables stored as text values) into numeric variables. We used the label encoding method to accomplish this goal. To implement the method in code, the preprocessing module of the sklearn library helped us by using it as shown in code (1).

```
LE = preprocessing.LabelEncoder()
data['App']=LE.fit_transform(data['App'])
```
(1)

VI. Visualization

Visualization in data mining plays a critical role in understanding complex data sets, improving customer relationships, and discovering meaningful patterns. Graphs, tables, and interactive visual tools provide an effective means of making information from large data sets more understandable. Furthermore, visualizations provide a valuable perspective for analysts to identify relationships between variables, analyze correlations and accelerate decision support processes. These visual tools also provide an effective way to share and understand data mining results and are a powerful communication tool to convey information to decision makers and stakeholders. The meaning and interactivity provided by visualization supports the successful implementation of data mining applications. Bar plot, heatmap, kdelpot, pie chart, wordcloud were used to visually analyze the existing data set.

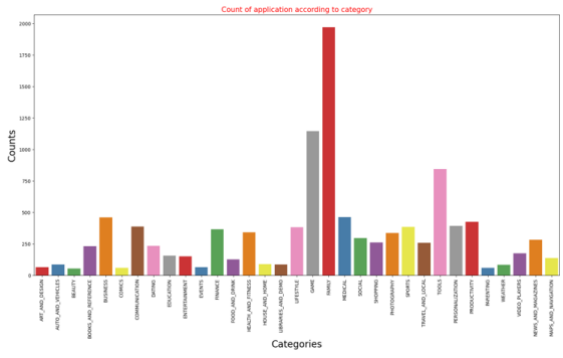

Fig. 2. Count of application according to category bar plot

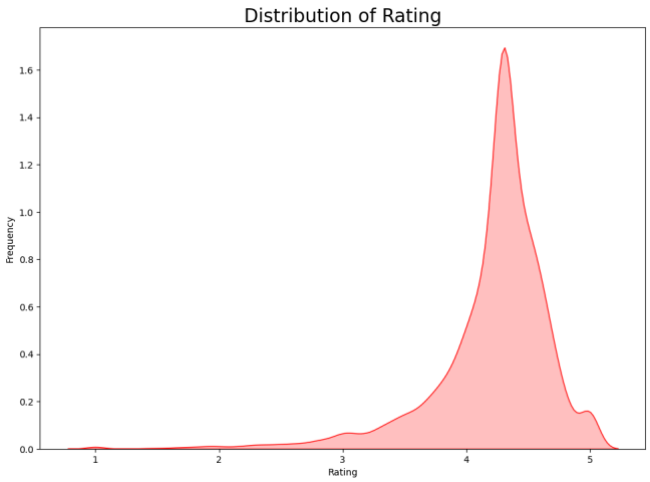

Fig. 3. Distributing of rating kdeplot

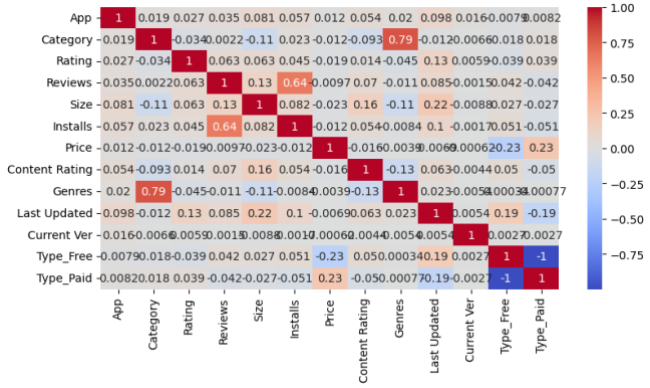

Fig. 4. Heatmap of dataset correlation

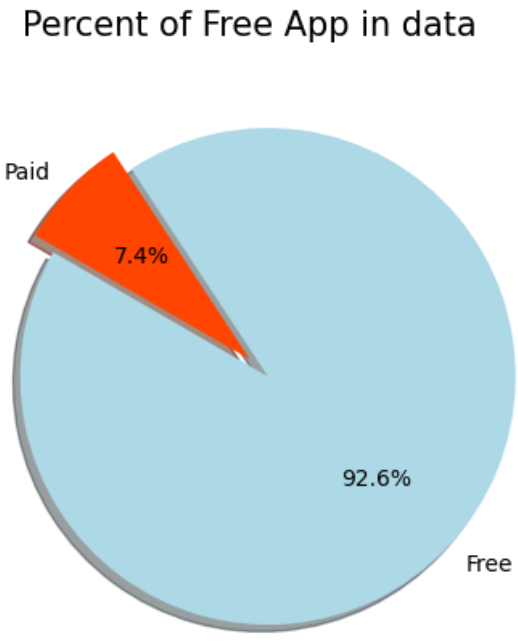

Fig. 5. Percent of free app in data pie chart

Fig. 6. Dataset categories with highest number of active apps as a wordcloud

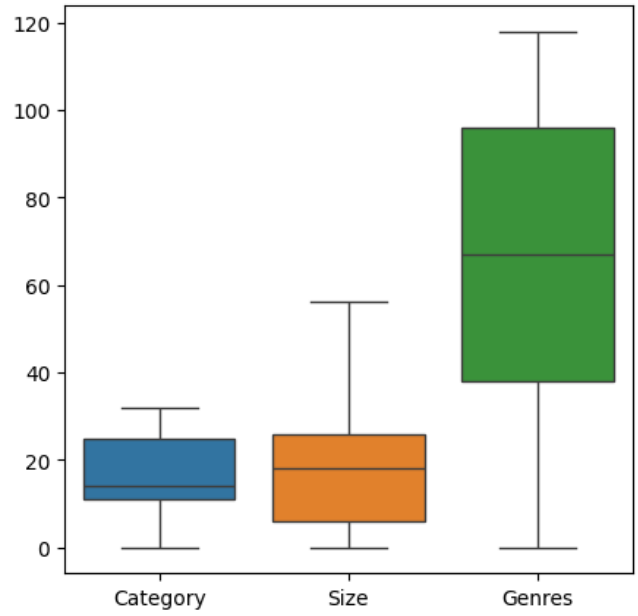

Fig. 7. Boxplots of 'Category', 'Size' and 'Genres' columns

VII. Applied Machine Learning Methods

It has decided to use regression rather than classification on the data. The reason for this is that it is desired to predict application ratings in a pinpoint manner. The following algorithms have used for regression problems:

- *Support Vector Regression*
- *Gradient Boosting Regressor*
- *XGBoost Regressor*
- *K Neighbors Regressor*

In addition, the following methods have used to evaluate the models more accurately:

- **n-fold Cross Validation (n=10):** Evaluation of the overall performance of the models.
- **Train-Test Split (90% Train, 10% Test):** Testing how the models perform on real-world data.

A. Feature Selection

Feature selection is the process of selecting or ranking features given as input to a machine learning model. This is done to filter out redundant or low-information features or to identify the most important features to improve model performance. Before applying this process for the models, 3 columns have added to the dataset. These are: **Review_to_Rating_Ratio, Size_to_Rating_Ratio and Installs_to_Rating_Ratio.**

SequentialFeatureSelector (SFS), used for feature selection, is a method used to automate the feature selection process. This method aims to select or eliminate features from the dataset in order to optimize model performance.

After using SFS, the features selected for each model and their line graphs (according to R2 scores) are as follows:

Support Vector Regression: ('Size', 'Size_to_Rating_Ratio')

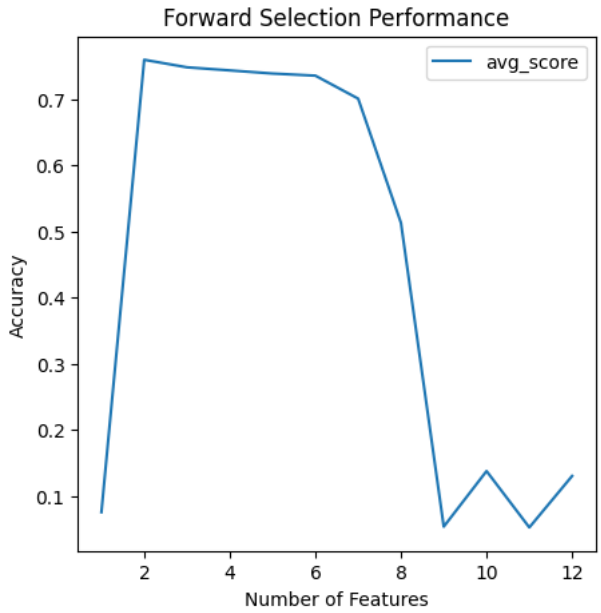

Fig. 8. Forward Selection Performance for SVR

Gradient Boosting Regressor: ('Installs', 'Price', 'Content Rating', 'Type_Free', 'Type_Paid', 'Installs_to_Rating_Ratio')

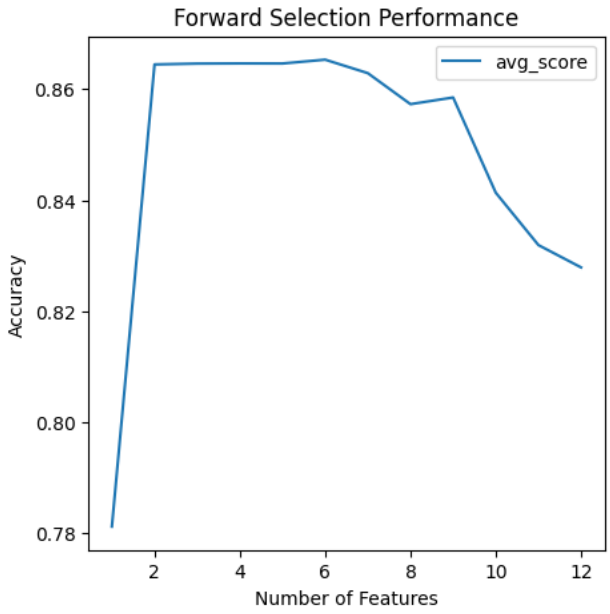

Fig. 9. Forward Selection Performance for Gradient Boosting

XGBoost Regressor: ('Reviews', 'Installs', 'Price', 'Content Rating', 'Type_Free', 'Type_Paid', 'Review_to_Rating_Ratio', 'Installs_to_Rating_Ratio')

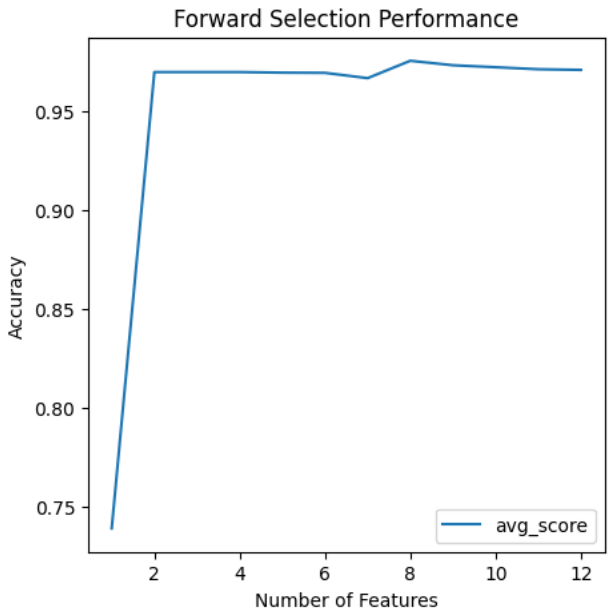

Fig. 10. Forward Selection Performance for XGBoost

K Neighbors Regressor: ('Installs', 'Installs_to_Rating_Ratio')

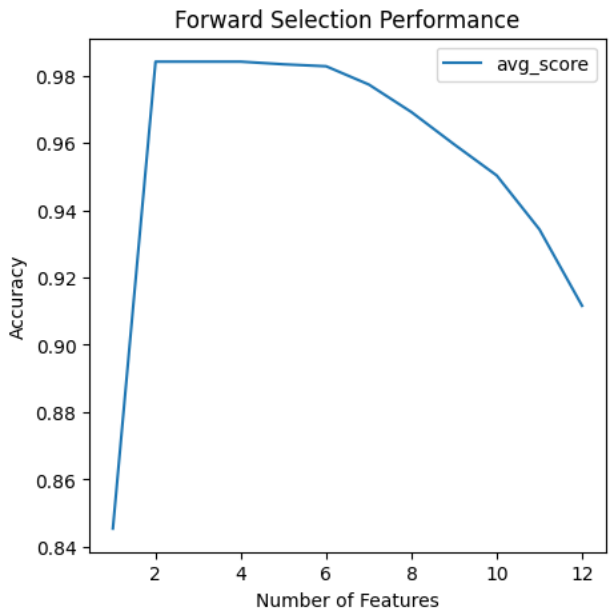

Fig. 11. Forward Selection Performance for K Neighbors Regressor

B. *Hyperparameter Tuning*

Hyperparameter tuning is a method used to optimize performance in machine learning models. This process involves systematically adjusting the hyperparameter values used to improve the model's success metrics.

BayesSearchCV, which has been decided to be used as the algorithm, is a hyperparameter optimization algorithm used to optimize the hyperparameter search process. This algorithm explores the hyperparameter space and tries to find the best hyperparameters through a limited number of trials.

Experiments with different values for different parameters for each algorithm are as follows:

1. Parameter value ranges for **Support Vector Regression**:

```
'C': [1, 25, 50, 100],
'kernel': ['linear', 'rbf',
'sigmoid'],
'gamma': ['scale', 'auto']
```

2. Parameter value ranges for **Gradient Boosting Regressor**:

```
'loss': ['squared_error',
'absolute_error', 'huber', 'quantile'],
'learning_rate': 0.001-0.3,
'n_estimators': 10-1000,
'criterion': ['friedman_mse',
'squared_error'],
'min_samples_leaf': 1-8,
'max_depth': 1-8,
'max_features': ['sqrt', 'log2']
```

3. Parameter value ranges for **XGBoost Regressor**:

```
'booster': ['gbtree', 'dart'],
'eta': [0.2, 0.3, 0.4],
'max_depth': 3-9,
'min_child_weight': 0-3,
'tree_method': ['auto', 'exact',
'approx', 'hist'],
```

4. Parameter value ranges for **K Neighbors Regressor**:

```
'n_neighbors': 1-8,
'weights': ['uniform', 'distance'],
'algorithm': ['auto', 'ball_tree', 'kd_tree',
'brute'],
'leaf_size': [1, 2, 3, 5, 10, 20, 30, 50],
'metric': ['euclidean', 'manhattan',
'chebyshev', 'minkowski'],
'p': [3, 4]
```

As seen in *"TABLE I"*, the K Neighbors Regressor model gave the best result after Bayesian Search has applied.

TABLE I. ACCURACY TABLE FOR BAYESIAN SEARCH

	Accuracy (R2)		
		After Model Fit	
Target Model	***Best***	***Train***	***Test***
Support Vector Regression	.9357	.9668	.9605
Gradient Boosting Regressor	.9911	.9999	.9556
XGBoost Regressor	.9918	.9994	.9852
K Neighbors Regressor	.9949	1.0	.9970

C. *N-fold Cross Validation*

N-fold cross-validation is a frequently used method to evaluate and validate the performance of a machine learning model. In this method, the data set is randomly divided into subgroups and each subgroup is used as a test set while the rest are used as training sets. This process continues until a specified number (n) of folds. The value of n has chosen as 10 for this method.

The error values, accuracy values and graphs for each model after n-fold cross validation are as follows:

TABLE II. K-FOLD RESULT TABLE

	Mean Scores			
Model	***MSE***	***RMSE***	***MAE***	***Test (R2)***
Support Vector Regression	.0144	.1192	.0778	.9378
Gradient Boosting Regressor	.0022	.0380	.0033	.9913
XGBoost Regressor	.0020	.0423	.0133	.9917
K Neighbors Regressor	**.0011**	**.0233**	**.0014**	**.9953**

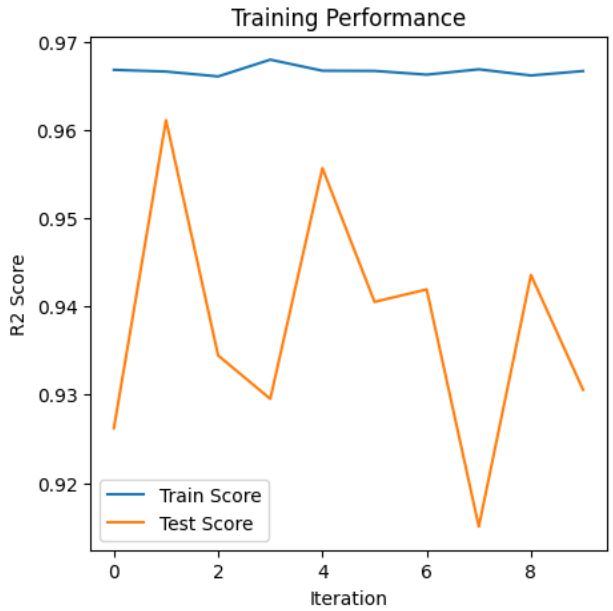

Fig. 12. N-fold training performance for SVR

Fig. 13. N-fold training performance for Gradient Boosting

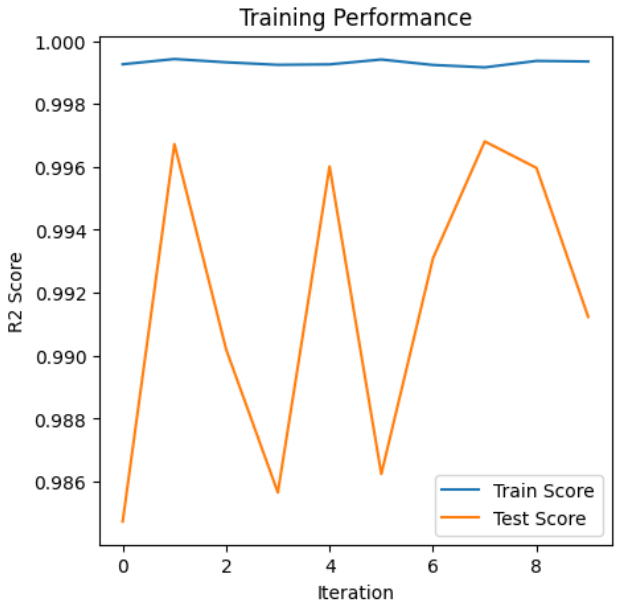

Fig. 14. N-fold training performance for XGBoost

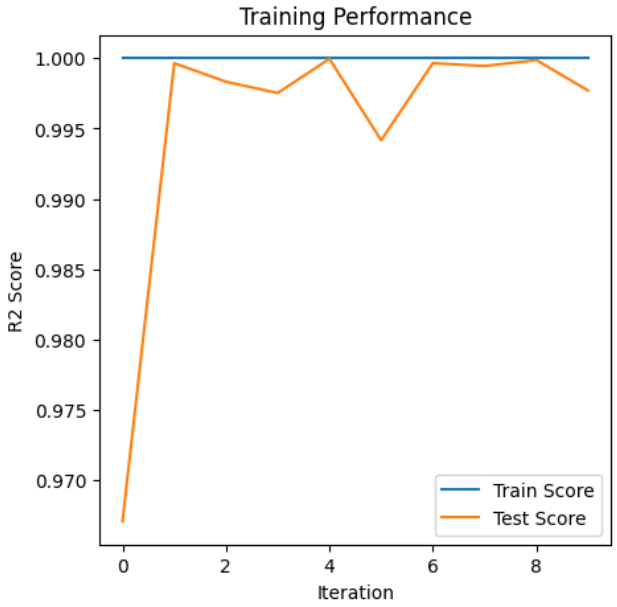

Fig. 15. N-fold training performance for K Neighbors Regressor

These figures show that during n-fold cross validation the training scores were always higher than the test scores. This may indicate that the models are not overfitted. Looking at the results in *"TABLE II"*, we see that the best values are in the K Neighbors Regressor model.

VIII. Results

Focusing on the parameter settings for each algorithm to achieve the best performance, it has been observed that the K Neighbors Regressor gave the best results with a specific parameter set against the other algorithms used (SVR, XGBoost, Gradient Boosting). This suggests that the model becomes optimal with a specific parameter configuration and that this parameter set should be chosen carefully. With these results, it is recommended to use the K Neighbors Regressor model to predict the rating of apps in the Google Play Store.

IX. Additional Figures

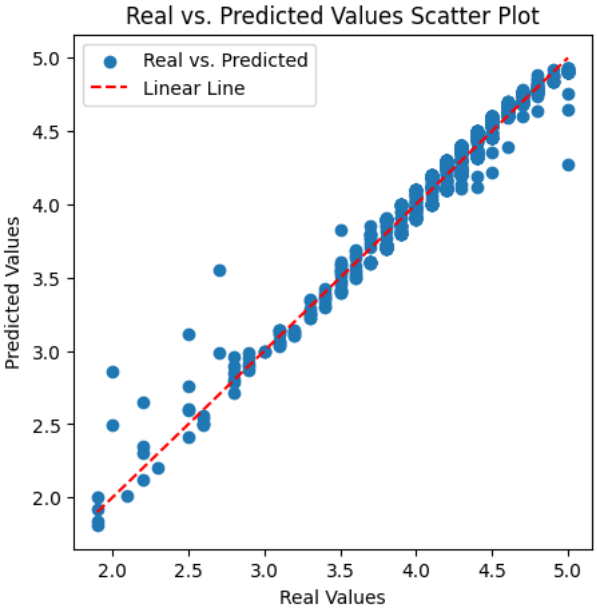

Fig. 16. Scatter plot for SVM predictions

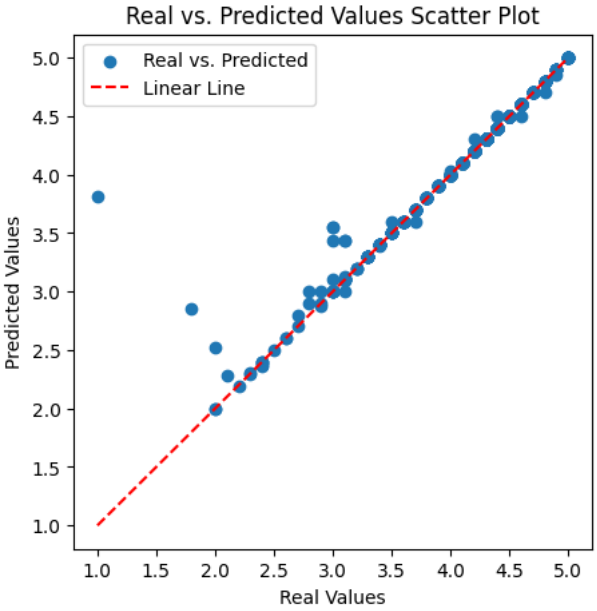

Fig. 17. Scatter plot for Gradient Boosting predictions

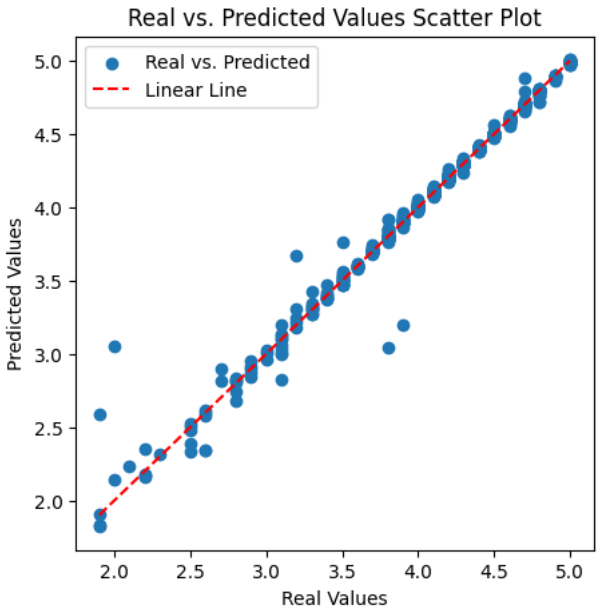

Fig. 18. Scatter plot for XGBoost predictions

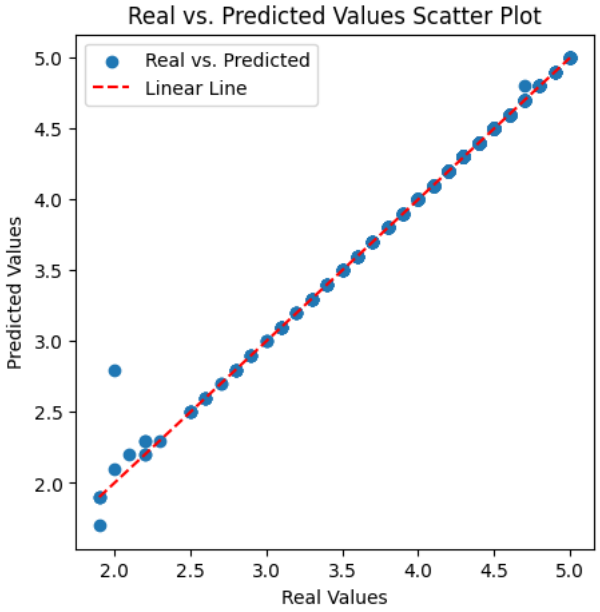

Fig. 19. Scatter plot for K Neighbors Regressor predictions

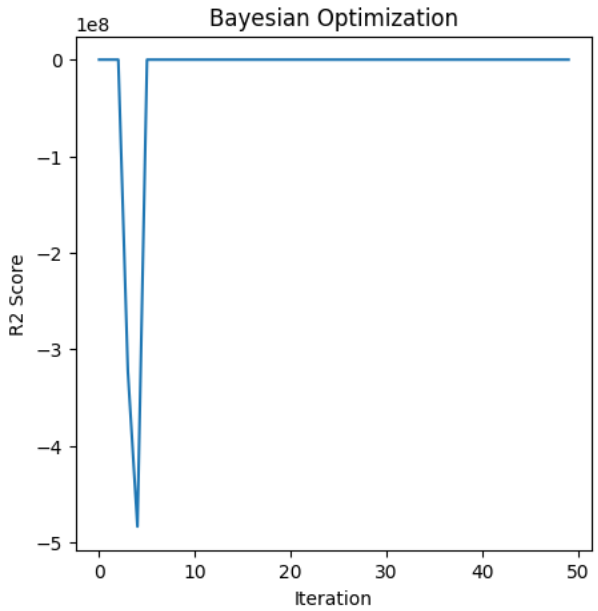

Fig. 20. SVM Bayesian Optimization score for each iteration

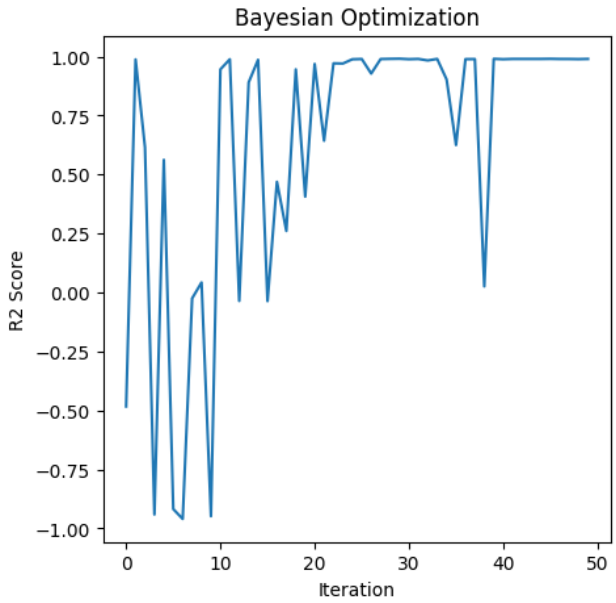

Fig. 21. Gradient Boosting Bayesian Optimization score for each iteration

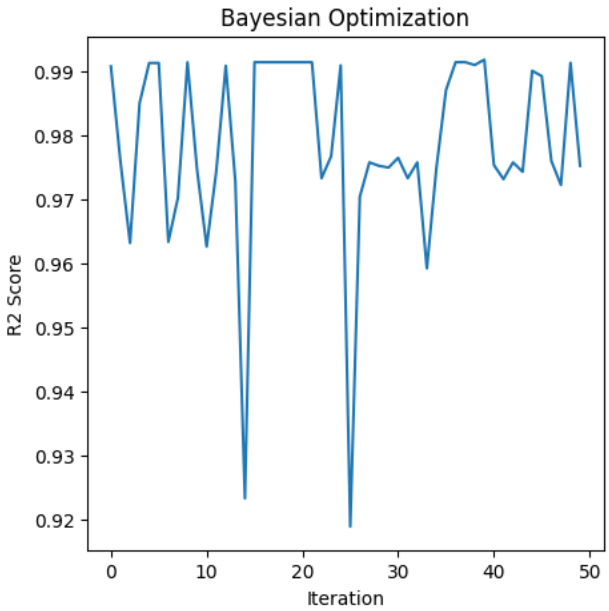

Fig. 22. XGBoost Bayesian Optimization score for each iteration

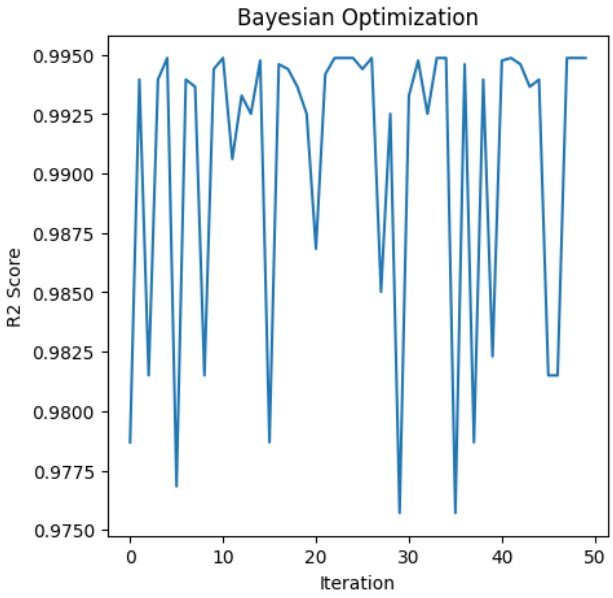

Fig. 23. K Neighbors Regressor Bayesian Optimization score for each iteration

REFERENCES

[1] G.*Anthony Finkelstein, Mark Harman, Yue Jia, William Martin, Federica Sarro, Yuanyuan Zhang,Investigating the relationship between price, rating, and popularity in the Blackberry World App Store, Information and Software Technology, Volume 87,2017*

[2] *M. R. Islam, "Numeric rating of Apps on Google Play Store by sentiment analysis on user reviews," 2014 International Conference on Electrical Engineering and Information & Communication Technology, Dhaka, Bangladesh, 2014, pp. 1-4, doi: 10.1109/ICEEICT.2014.6919058.*