Table of Content

CAPSTONE PROJECT

INTRODUCTION TO DATA SCIENCE REPORT

1. Overview

The rise in E - commerce, has brought a significant rise in the importance of customer reviews. In addition, mobile apps have become so prevalent, so more and more people make their living as a mobile developer. In this project, we attempt to build a system that can make prediction about the average rating for an app based on its categories.

1.1 Introduction

Our goal was to find the overall rating of an app because so much of the users' trust in the app comes from that one statistic alone. Higher rated apps are more likely to be recommended and more likely to be trusted by users that find the app while browsing the app store.

1.2 Aim of the project

The vast majority of this project was about cleaning up, preprocessing the data and train a machine learning algorithm model to predict ratings of apps. Since all of the data was scraped directly from the Google Play Store, there were a lot of errors in transcription (NaN values representing nothing scraped, shifted data columns, etc.) and categorical values to translate or encode (more on that later). From there, we applied a multitude of regression models such as the Decision Tree Regression and Random Forest Regression.

1.3 Investigation

- The approaches and techniques used to conduct predictive analysis:
  - Regression techniques: focus on establishing a mathematical equation as a model to represent the interactions between the different variables
    - Linear regression model
    - Discrete choice models
    - Classification and regression models
  - Machine learning techniques: emulate human cognition and learn from training examples to predict future events
    - Neural networks
    - *k*-nearest neigbours
    - Naïve Bayes, …

2. Solution

2.1 Data preparation

2.1.1 Dataset

- The data used in this project has been obtained from Kaggle. It has a publicity dataset that contains 11 thousand app information. Data points include app name, category, number of reviews, size, number of installs, types, content rating, genres, last update, version, and its rating.
- The data file was used in the project: googleplaystore.csv

2.1.2 Data pre-processing

- Data exploration
- Data cleaning

We saw that the dataset contains some features which had null data, so we will clean this dataset by using *fillna()* with the median value.

```
data.isnull().sum()
```

```
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
```

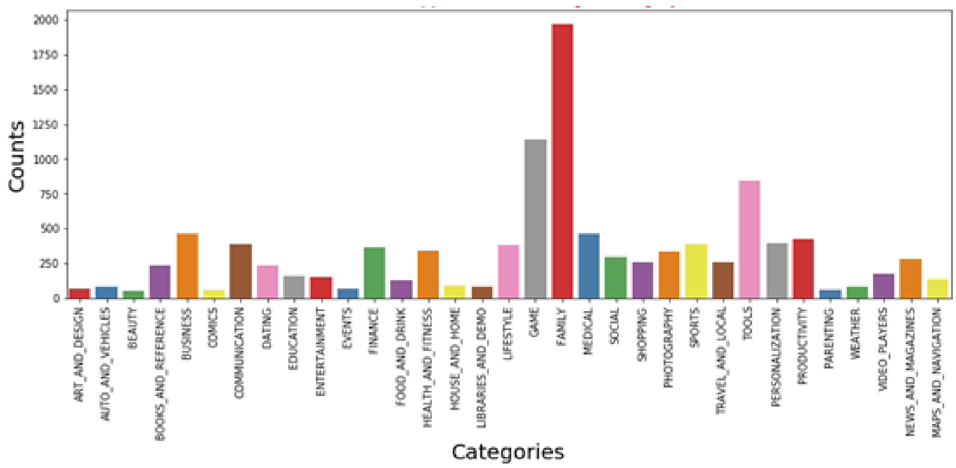
```
# Fill missing values using the median

data['Rating'] = data['Rating'].fillna(data['Rating'].median())
```

We also remove all the Unicode character and unreasonable values.

2.1.3 Data visualization

- Family app and game are the two most popular categories:

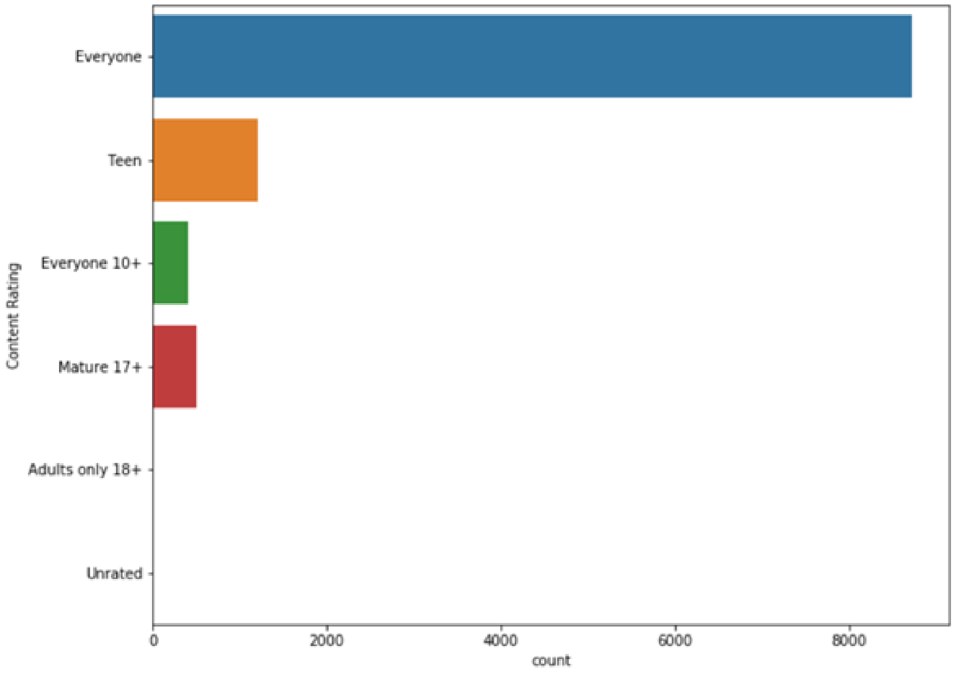


[1]

- Most of the app are free

- Most of the app are for everyone

[1] Count



2.2 Build model

Decision tree is a model that can handle high dimensional data with good accuracy. Random forest can handle data without preprocessing. Random forest algorithm has been used in prediction and probability estimation. It operates by constructing a multitude of decision tree at training time and output the mean prediction of individual trees.

In this project we have tried Decision Tree and Random Forest technique to build an app rating prediction system.

Main steps in analyzing:

- Import library and load the Dataset into a Data Frame

```
import re
import sys

import pandas as pd
import numpy as np

import time
import datetime

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn import preprocessing

from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

from sklearn import metrics
from sklearn.metrics import accuracy_score, mean_squared_error,mean_absolute_error

from sklearn import tree
from sklearn.model_selection import cross_val_score
```

```
from google.colab import drive
drive.mount('/content/drive')

data = pd.read_csv("/content/drive/My Drive/DS/googleplaystore.csv")
```

- Encodes data to the numeric form

```
[ ] # App values encoding

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

```
[ ] # Category features encoding

    CategoryList = data['Category'].unique().tolist()
    CategoryList = ['cat_' + word for word in CategoryList]
    data = pd.concat([data, pd.get_dummies(data['Category'], prefix='cat')], axis=1)
```

```
[ ] # Type encoding

    data['Type'] = pd.get_dummies(data['Type'])
```

```
[ ]  data.dtypes
```

```
App                  int64
Category            object
Rating             float64
Reviews             object
Size               float64
Installs            object
Type                 int64
Price               object
Content Rating       int64
Genres               int64
Last Updated       float64
Current Ver        float64
```

- Split dataset into the training set and test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state=10)
```

- Try with the Ridge model (kind of linear regression)

```
r_clf = Ridge(alpha=0.1, random_state=42)
r_clf.fit(X_train, y_train)
```

```
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=42, solver='auto', tol=0.001)
```

```
r_accuracy = r_clf.score(X_test, y_test)
r_accuracy
```

```
-2.79275523426645e+33
```

```
r_predict = r_clf.predict(X_test)
'Mean Absolute Error:', metrics.mean_absolute_error(y_test, r_predict)
```

```
('Mean Absolute Error:', 1630619667530355.5)
```

In the first time we have tried with the Ridge regression model, it had high accuracy for prediction. But when we evaluated this model by using cross-validation method, the average accuracy is negative. Then, we found this model is not good for our dataset by splitting the dataset again. We thought that the reason making the failure of this model is the non-linearity of our dataset. Therefore, we decided to use the Random forest algorithm, the popular way used to deal with non-linear case, to train our dataset.

- Using Grid Search to determine the best value for max_depth and min_samples_leaf

```python
from sklearn.model_selection import GridSearchCV
grid_param = {
    'max_depth': [3, 4, 5, 6, 8],
    'min_samples_leaf': [3, 4, 5, 6, 8]
}
```

```python
classifier = tree.DecisionTreeRegressor(criterion='mae', random_state=42)
gd_sr = GridSearchCV(estimator=classifier,
                     param_grid=grid_param,
                     cv=5,
                     n_jobs=-1)
gd_sr.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeRegressor(criterion='mae', max_depth=None,
                                             max_features=None,
                                             max_leaf_nodes=None,
                                             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=42,
                                             splitter='best'),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [3, 4, 5, 6, 8],
                         'min_samples_leaf': [3, 4, 5, 6, 8]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```python
best_parameters = gd_sr.best_params_
print(best_parameters)
```

```
{'max_depth': 5, 'min_samples_leaf': 6}
```

Although the best value for (max_depth, min_samples_leaf) = (5, 6), we found that it has the same accuracy and MAE with the value (5, 5).

Therefore, we choose max_depth = 5 and min_samples_leaf = 5.

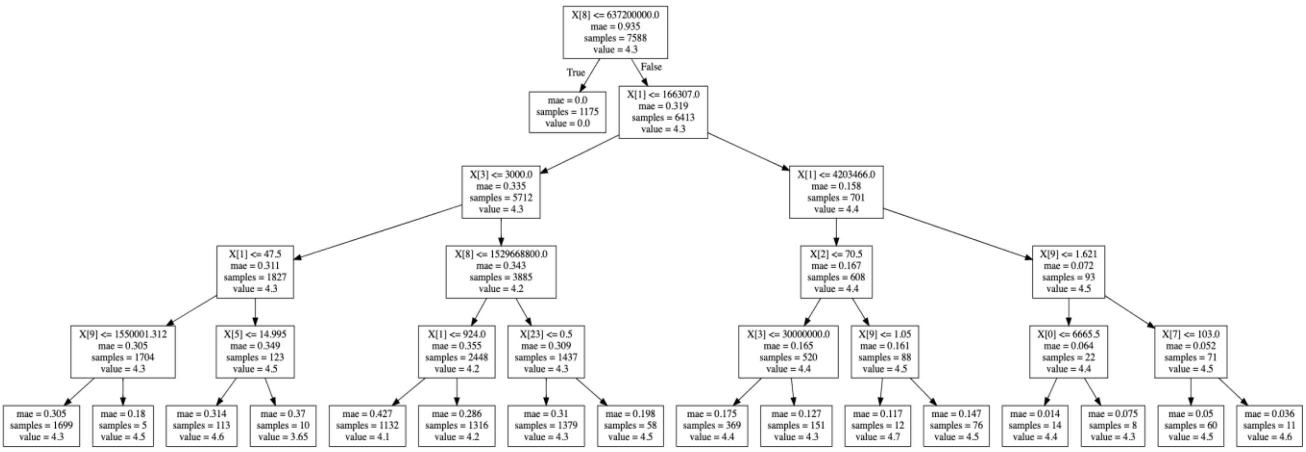
- Create a decision tree model and fit it to the training data

```python
dt_clf = tree.DecisionTreeRegressor(criterion='mae', max_depth=5, min_samples_leaf=5, random_state=42)
```

```python
dt_clf.fit(X_train, y_train)
```

```
DecisionTreeRegressor(criterion='mae', max_depth=5, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=5,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=42, splitter='best')
```

- Plot the tree



- Calculate the accuracy of the tree

```
dt_accuracy = dt_clf.score(X_test, y_test)
dt_accuracy
```

```
0.9258017869284529
```

```
dt_predict = dt_clf.predict(X_test)
'Mean Absolute Error:', metrics.mean_absolute_error(y_test, dt_predict)
```

```
('Mean Absolute Error:', 0.25176814268142683)
```

- Use cross validation to choose the number of trees for the forest

```
n_trees = [20, 50, 200, 300]
ntree_scores = []
for n in n_trees:
  print(n)
  rf_model = RandomForestRegressor(n_estimators = n, n_jobs=-1, random_state=10)
  rf_scores = cross_val_score(rf_model, X_train, y_train, cv=5)
  ntree_scores.append(rf_scores)
ntree_scores
```

```
20
50
200
300
[array([0.9280866 , 0.92562393, 0.91292065, 0.91384588, 0.91992858]),
 array([0.93194409, 0.92719752, 0.9166499 , 0.9147995 , 0.92371791]),
 array([0.93225955, 0.92836703, 0.91797859, 0.91611285, 0.92553416]),
 array([0.93255529, 0.92817901, 0.91821648, 0.91625103, 0.92576244])]
```

We realized that the accuracy does not increase from 200 to 300, so we choose the number of decision trees is 200

- Create a random forest model and fit it to the training data

```
model = RandomForestRegressor(n_estimators = 200, n_jobs=-1, random_state=10)
model.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=-1,
                      oob_score=False, random_state=10, verbose=0,
                      warm_start=False)
```

- Calculate accuracy and some errors

```
accuracy = model.score(X_test, y_test)
accuracy
```

```
0.9366393168860417
```

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

```
('Mean Absolute Error:', 0.24574338868388707)
```

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

```
('Mean Squared Error:', 0.162998591097786)
```

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

```
('Root Mean Squared Error:', 0.4037308399141512)
```

2.3 Feature selection

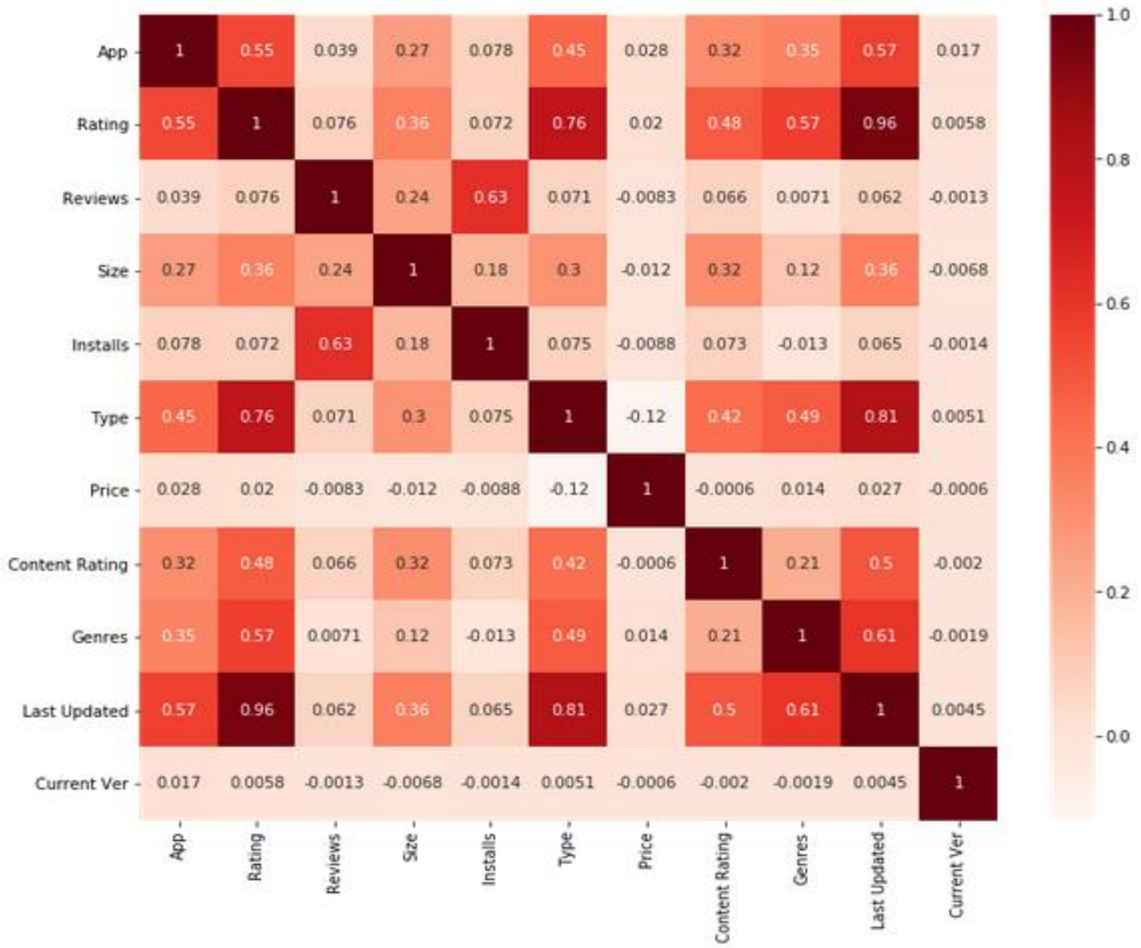
2.3.1 Filter methods

Filter and take only the subset of the relevant features. The model is built after selecting the features. The filtering here is done using correlation matrix and it is most commonly done using Pearson correlation.

- Plot the Pearson correlation heatmap and see the correlation of independent variables with the output variable Rating

```
features =['App', 'Reviews', 'Size', 'Installs', 'Type', 'Price', 'Content Rating', 'Genres', 'Last Updated', 'Current Ver']
```

```
#  Filter
plt.figure(figsize=(12,10))
data1 = data.iloc[:,0:13]
# reviews cleaning
data1['Reviews'] = data1['Reviews'].astype(int)
data1['Installs'] = data1['Installs'].astype(int)
data1['Price'] = data1['Price'].astype(float)
cor = data1.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```



- Select features which has correlation of above 0.5 and check the correlation of selected features with each other

```
[1821] #Correlation with output variable
       cor_target = abs(cor["Rating"])#Selecting highly correlated features
       relevant_features = cor_target[cor_target>0.5]
       relevant_features
```

```
App              0.545810
Rating           1.000000
Type             0.761784
Genres           0.569438
Last Updated     0.957377
Name: Rating, dtype: float64
```

```
[1822] print(data1[["App","Type"]].corr())
       print(data1[["Type","Genres"]].corr())
       print(data1[["Genres","Last Updated"]].corr())
       print(data1[["Last Updated","App"]].corr())
```

```
           App      Type
App   1.000000  0.446999
Type  0.446999  1.000000
            Type    Genres
Type    1.000000  0.488121
Genres  0.488121  1.000000
                Genres  Last Updated
Genres        1.000000      0.607336
Last Updated  0.607336      1.000000
              Last Updated       App
Last Updated      1.000000  0.566759
App               0.566759  1.000000
```

- Hence, we would keep App, Type, Last Updated for the training model

Result after using filter methods:

- Decision tree

```
[ ] ac = clf.score(x_ft, y_ft)
    ac
```

```
0.9215619744189013
```

- Random forest

```
[ ] accuracy = model.score(x_ft, y_ft)
    accuracy
```

```
0.9214301587854323
```

2.3.2 Wrapper methods - RFE (Recursive Feature Elimination)

A wrapper method is an iterative and computationally expensive process but it is more accurate than the filter method.

The Recursive Feature Elimination (RFE) method works by recursively removing attributes and building a model on those attributes that remain. It takes the model to be used and the number of required features as input, then gives the ranking of all the variables, 1 being most important.

```
#RFE selection
from sklearn.feature_selection import GenericUnivariateSelect, chi2
from sklearn.feature_selection import RFE

from sklearn.svm import SVR
estimator = SVR(kernel="linear")
selector = RFE(estimator, 5, step=1)
selector = selector.fit(X, y)
selector.support_
selector.ranking_
```

```
array([ 1,  1,  3,  1,  5,  6,  4,  2,  1,  1, 35, 36, 38, 32, 23, 39, 16,
       19, 20, 21, 37,  7, 22, 15, 26, 30, 17, 29, 25, 12, 28,  8, 11,  9,
       13, 10, 27, 18, 34, 31, 33, 14, 24])
```

From the output, feature 'App','Reviews', 'Installs','Last Updated','Current Ver' would be keep to train model.

Result after using RFE:

- Decision tree

- Random forest

```
[ ] accuracy = model.score(x_rfe, y_rfe)
    accuracy
```

```
0.9341968951341199
```

3. Evaluation

➢ MAE (Mean Absolute Error)

$$mae = \frac{\sum_{i=1}^{n} abs\left(y_i - \lambda(x_i)\right)}{n}$$

In the regression problem, the mean absolute error is the important value to evaluate the efficiency of a model.

- Decision tree

```
dt_predict = dt_clf.predict(X_test)
'Mean Absolute Error:', metrics.mean_absolute_error(y_test, dt_predict)
```

```
('Mean Absolute Error:', 0.2542127921279213)
```

- Random forest

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

```
('Mean Absolute Error:', 0.245743388683887)
```

➢ Accuracy:

The accuracy is always the important result for prediction system.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

- Decision tree

```
dt_accuracy = dt_clf.score(X_test, y_test)
dt_accuracy
```

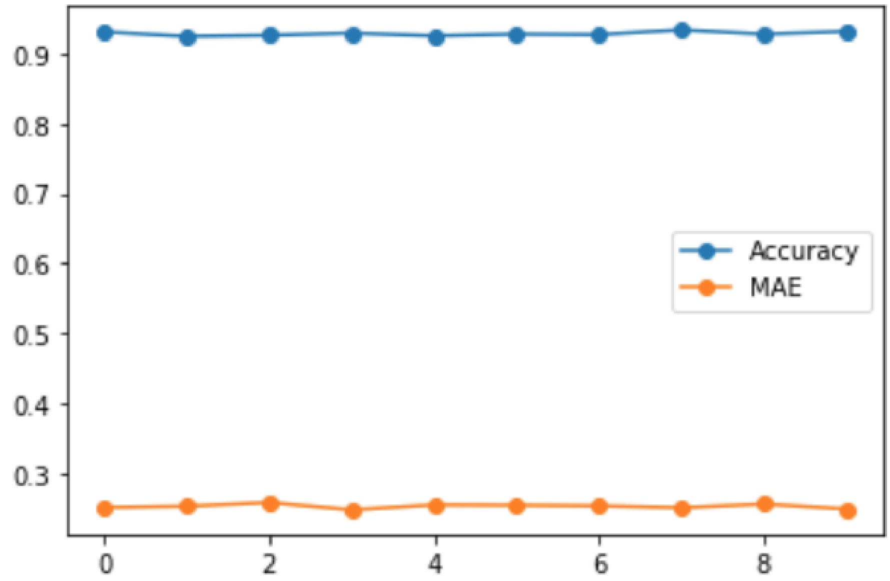
```
0.9216974647212245
```

- Random forest

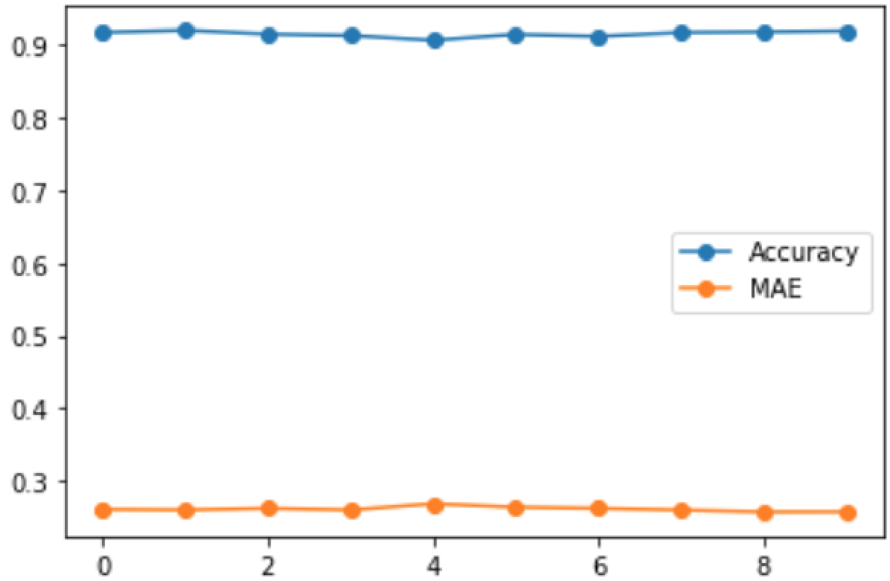
```
accuracy = model.score(X_test, y_test)
accuracy
```

```
0.9320085377856966
```

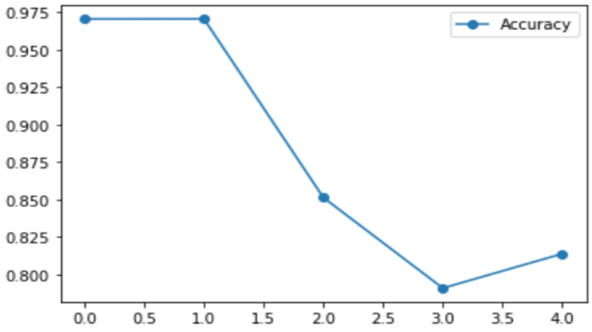
- Repeated hold-out
  - Decision tree



  - Random forest



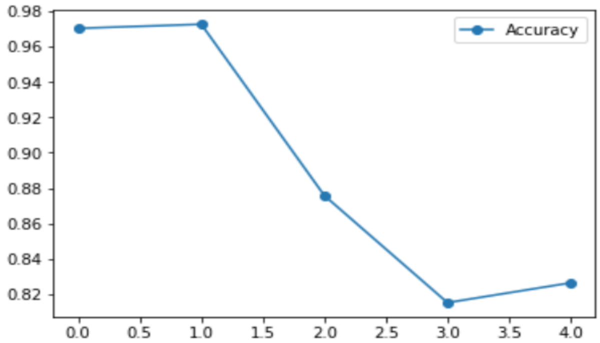
- Cross-validation
  - Decision tree




```
print("Accuracy: %0.2f (+/- %0.2f)" % (dt_scores.mean(), dt_scores.std() * 2))
```

```
Accuracy: 0.88 (+/- 0.15)
```

- Random forest




```
print("Accuracy: %0.2f (+/- %0.2f)" % (rf_scores.mean(), rf_scores.std() * 2))
```

```
Accuracy: 0.89 (+/- 0.14)
```

4. Conclusions

Below is the accuracy after evaluating implemented models.

	Decision Tree	Random Forest
Before feature selection	92.16%	93,20%
After feature selection	92.64%	93.41%

Based on the results, it can be concluded that in our case, Random Forest is a good method for predicting the app's rating with high accuracy.

5. Members Roles

- **Nguyễn Đức Dũng**
    - Building Machine Learning Model
    - Evaluation
- **Trần Đức Phụng**
    - Building Machine Learning Model
    - Evaluation
- **Đào Ngọc Thành**
    - Data Cleaning
    - Building Machine Learning Model
- **Trần Bích Ngọc**
    - Plotting
    - Feature Selection
- **Vũ Thanh Tùng**
    - Categorical Data Encoding
    - Feature Selection

6. References:

- Get crawled data: https://www.kaggle.com/lava18/google-play-store-apps
- Train data:https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d
- Scikit-learn: https://scikit-learn.org/
- Feature-selection:https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b
- Slides of Course: Introduction to Data Science