Michael Chung

Kevin Terwelp

Group 7

Predicting the Success of a Google Play Store App

Introduction (Michael Chung and Kevin Terwelp)

The data set chosen from the Kaggle database contained information about mobile apps on the Google App Play Store. The information included variables such as the app's name, the average user rating for the app, the price of the app, the number of user reviews completed for the app and the number of times the app was installed on a user's device. After excluding variables and observations with incomplete data, the data set used for analysis contained 13 variables and 6,935 observations.

After reviewing the data available, the decision was made to analyze how to determine if an app will be successful on the Google Play Store. This led to the data question: How do you predict the app rating on the Google Play Store? Therefore, the *Rating* variable was chosen as the response variable and the other 12 variables were used as predictors. The linear model and tree-based model were used to analyze the data.

Model Outlines (Michael Chung and Kevin Terwelp)

The two models used to attempt predicting the *Rating* response variable are linear regression and decision trees.

Linear regression models perform better than decision trees when the data is linearly separable. The linear regression model also has a higher predictive accuracy at the cost of interpretability. If, however, the dataset has a complex, non-linear boundary then decision trees should provide better results.

Decision trees provide a clear visualization of the data and are easy to interpret. The decision trees also more closely resemble a human's decision-making process compared to other regression models like linear regression.

However, despite its easy interpretability, decision trees are less accurate with prediction compared to other regression models. While random forests will be used to improve the prediction accuracy of the decision trees, the data will need to be analyzed to determine the best fit for prediction accuracy.

Linear Model (Michael Chung)

Model Formula: *Rating ~ Reviews+Installs+Reviews:Installs*

The final linear model consists of 2 main effects *Reviews* and *Installs* and 1 interaction term *Reviews:Installs*. The number of reviews could depend on the number of installs and because of this the interaction term *Reviews:Installs* was constructed and found to be statistically significant which suggests that there is an interaction relationship between the 2.

```
(Intercept)       4.152e+00  6.866e-03 604.729  < 2e-16 ***
Reviews           4.363e-08  8.914e-09   4.894 1.01e-06 ***
Installs          7.820e-10  3.686e-10   2.121  0.03392 *
Reviews:Installs -7.817e-17  2.569e-17  -3.043  0.00235 **
```

The variable *App* was not considered because each app name is unique and thus irrelevant.

Variable *Genres* was dropped because of their many factor levels that proved to be insignificant and additionally could be represented by the variable *Category*.

Initial inspection of the variable *Category* showed that many of the factor levels proved significant, however, after encoding each subgroup into a dummy-variable and cross-validating, the model could not produce any higher-order polynomial terms above linear due to numerical overflow. Additionally, the linear cross-validation results showed no significant improvement over the final model that was chosen and consequently was omitted.

During the stepwise regression process, the variables *Size*, *Type*, *Price*, *Last_Updated*, *Current_Ver*, and *Android_Ver* proved to be insignificant, degraded the model's performance, and theoretically should have no impact on rating. Because of this all 6 were disregarded from the final model.

Collinearity of the final model was tested using the following code:

```
> VIF(lm(Rating~Reviews*Installs,data=googleplaystore))
[1] 1.005904
```

There was little evidence to suggest its presence.

Leave-One-Out CV was performed up to the 5th polynomial to determine MSE

```
> cv.error=rep(0,5)
> for (i in 1:5){
+     print(i)
+     glm.fit=glm(Rating~poly(Reviews*Installs,i),data=googleplaystore)
+     cv.error[i]=cv.glm(googleplaystore,glm.fit)$delta[1]
+ }
> cv.error
[1]    0.3145369    0.3361602    1.3472212   26.3551544 4356.4024421
```

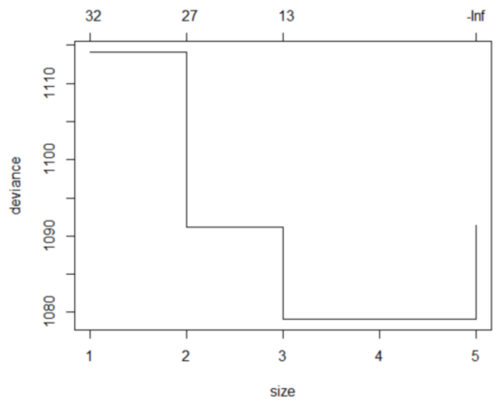
LOOCV shows little evidence that cubic or any higher-order polynomial terms lower test error compared to a linear or even quadratic fit. When compared, the polynomial with the lowest MSE obtained is 0.3145 and is attributed to the linear fit. Average MSE after 10 iterations of testing prediction error is 0.3206 for the linear fit model.

Tree-Based Model (Kevin Terwelp)

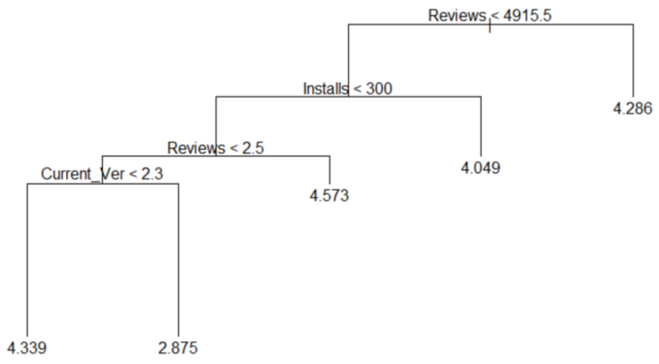
Model formula: *Rating* ~ *App* + *Category* + *Reviews* + *Size* + *Installs* + *Type* + *Price* + *Content_Rating* + *Genres* + *Last_Updated* + *Current_Ver* + *Android_Ver*

The decision tree created from the data set only had 5 terminal nodes. Tree pruning is generally used after the creation of the decision tree to determine if a decision tree with a smaller number of terminal nodes is a better fit for the data. In tree pruning, a large decision tree is formed and then pruned back to a smaller subtree with less terminal nodes. The data from tree pruning is then plotted to determine which number of terminal nodes provides the lowest variance resulting in a better prediction performance.

Since the initial decision tree only contained 5 terminal nodes, tree pruning was not able to improve the results. The plot below shows that a decision tree with 3, 4 or 5 terminal nodes will provide the lowest variance and the best prediction performance. Therefore, the original decision tree will be used for the analysis.



The plot of the original decision tree is below.



The decision tree above shows only 3 variables were important in predicting the response variable *Rating*. This means only 3 variables out of 12 are useful in predicting the value of each Google Play Store app's rating. These 3 variables are *Reviews, Installs* and *Current_Ver.* The *Reviews* variable is the number of reviews completed for each app. The *Installs* variable is the number of times the app was installed on users' devices. The *Current_Ver* variable is the most recent version available for each mobile app.

Using the information from the above decision tree, the test prediction error was calculated with the following code:

```
> google_apps.test = googleplaystore[-train, "Rating"]
> yhat = predict(tree.google_apps, newdata = googleplaystore[-train,])
> mean((yhat-google_apps.test)^2)
[1] 0.2930617
```

The output from the code above is the mean sum of squared errors (MSE) for the test set. This value is 0.2931. To calculate the test prediction error, the square root of the MSE must be obtained. Therefore, the test prediction error is 0.5414. This means this tree-based model leads to test predictions within 0.5414 of the true rating for the app on the Google Play Store.

To improve this prediction accuracy, the random forests method must be used. The random forests method will grow large un-pruned trees and pick a random subset of predictors at each tree split. This method will reduce the chance of one variable dominating each tree and will stabilize the variance of the prediction.

The random forests method is used on the previous decision tree with the following code:

```
> rf.google_apps = randomForest(Rating~.,
+                               data = googleplaystore,
+                               subset = train,
+                               importance = TRUE)
 > yhat.rf = predict(rf.google_apps, googleplaystore[-train,])
> mean((yhat.rf-new_google_apps.test)^2)
[1] 0.2745919
```

As shown above, the MSE value for the test set is 0.2746. By calculating the square root of this value, the test prediction error is 0.5240. This means the tree-based model using the random forests method leads to test predictions within 0.5240 of the true rating for the app on the Google Play Store. This is only a slight improvement from the original test prediction error above of 0.5414.

The values obtained above were calculated by setting the seed in R Studio to 1. The test prediction error was lowest when the seed was set to 1. However, the calculations were completed again by setting the seed for nine more values from 2 through 10. This process will provide a more accurate calculation for the test prediction error by using different information from the data set for each seed and calculating the average test prediction error.

After setting the seed nine more times using numbers 2 through 10, the average MSE for all calculations is 0.3027 for the initial decision tree. Therefore, the test prediction error average is 0.5502. This means this tree-based model leads to test predictions within 0.5502 of the true rating for the app on the Google Play Store. The average MSE using the random forests method is 0.3373 and the test prediction error calculated is 0.5808. This means this tree-based model leads to test predictions within 0.5808 of the true rating for the app on the Google Play Store when using the random forests method. By averaging the values from all 10 seeds, the initial decision tree test prediction error outperforms the test prediction error obtained with the random forests method.

Prediction Comparison (Michael Chung and Kevin Terwelp)

After comparing the MSE between the linear model and the tree-based model, the tree-based model is a better fit for the data set with the lowest MSE of 0.2746. The lowest MSE obtained by the linear model with value 0.2892. The average MSE of the linear fit model of 0.3206 is even higher than the average MSE of the tree-based model with an average value of 0.3027 using 10 different iterations of prediction error testing.

The lower MSE for the tree-based model means it would have a lower error rate when predicting the rating of an app on the Google Play Store. Despite the tree-based model's better performance, however, further evaluation needs to be done about the model's ability to predict the app's rating in the Google Play Store.

Conclusion (Michael Chung and Kevin Terwelp)

The tree-based model outperformed the linear model with a lower MSE but the tree-based model provided inconclusive data for the data question posed. One of the benefits of using the tree-based model is its easy interpretability by observing the tree diagram above. The decision tree formed from the data set only contained 5 terminal nodes and the difference between the value at each terminal node was minor.

Since the difference between most predictions is less than 0.5 and most predictions have a rating of greater than 4, the information would not be useful to a potential mobile app developer. From the decision tree diagram above, most predictions on the tree only range from 4.049 to 4.573. The only exception is one terminal node with a value of 2.875. If a developer was provided with the information from the tree-based model, he or she cannot easily determine if an app's predicted rating on the Google Play Store would be high or low due to a lack of variation with the predicted values. Therefore, the developer is unable to use the data from the tree-based model to determine the potential success or failure for a future mobile app on the Google Play Store. This means the answer to the data question "How do you predict the app rating on the Google Play Store?" is inconclusive.

Due to the formation of a small decision tree with little variation in the prediction values, the 12 predictor variables' data from the data set do not appear to be good predictors of the response variable *Rating*. Additionally, further analysis needs to be done with the data set such as fitting and testing an artificial neural network and determining what other questions the data may be able to answer. For example, “Can you predict the number of mobile app installs on the Google Play Store?” or “Can you predict the number of reviews on the Google Play Store?” are data questions that might provide more insight regarding the success of a mobile app on the Google Play Store. Through further analysis of the data available, mobile app developers may be able to determine how to create the next popular app on the Google Play Store.

```
#Linear model code

library(ISLR)
library(boot)
RNGkind(sample.kind = "Rounding")
set.seed(1)

#Converting number of Installs from factor to numeric
googleplaystore$Installs<-as.numeric(gsub(",", "", googleplaystore$Installs))

#Convert Genres to factor
googleplaystore$Genres<-factor(googleplaystore$Genres)


#Convert Category variables to dummy variables
googleplaystore<-fastDummies::dummy_cols(googleplaystore,select_columns = "Ca
tegory")

#Final linear model that was decided on
lm.fit=lm(Rating~Reviews*Installs, data=googleplaystore,subset=googleplaystor
e)

#Check for collinearity
VIF(lm(Rating~Reviews*Installs,data=googleplaystore))

#LOOCV of model
cv.error=rep(0,5)
for (i in 1:5){
print(i)
glm.fit=glm(Rating~poly(Reviews*Installs,i),data=googleplaystore)
cv.error[i]=cv.glm(googleplaystore,glm.fit)$delta[1] #Always roll with $delta
[1]
}
cv.error


#Finding MSE of linear fit model
#Repeated with 10 different seeds for 10 iterations of prediction testing
set.seed(1)
train = sample(6935 ,5548)
lm.fit=lm(Rating~Reviews*Installs, data=googleplaystore,subset=train)
mean((Rating-predict(lm.fit,googleplaystore))[-train]^2)
```

```
#Decision tree code

library(tree)
set.seed(1)
train = sample(1:nrow(googleplaystore), nrow(googleplaystore)/2)
tree.google_apps = tree(Rating~., data = googleplaystore, subset = train)
summary(tree.google_apps)
tree.google_apps

#Decision tree plot
plot(tree.google_apps)
text(tree.google_apps, pretty = 0)

##Pruning the tree is not helpful since size
##is only 5 terminal nodes
cv.google_apps = cv.tree(tree.google_apps)
plot(cv.google_apps)
google_apps.test = googleplaystore[-train, "Rating"]
yhat = predict(tree.google_apps, newdata = googleplaystore[-train,])
plot(yhat, google_apps.test)
abline(0,1)

#MSE of decision tree
mean((yhat-google_apps.test)^2)

#Random forests code
library(randomForest)
set.seed(1)
rf.google_apps = randomForest(Rating~.,
                              data = googleplaystore,
                              subset = train,
                              importance = TRUE)
yhat.rf = predict(rf.google_apps, googleplaystore[-train,])

#MSE with random forest method
mean((yhat.rf-new_google_apps.test)^2)
```