Predicting Android App Downloads using Machine Learning

Jeffrey Gugelmann

25/07/2021

Introduction

This report aims to predict how many installs an Android app receives based on app characteristics including the category of the app, its price and how many reviews it has received. The dataset used in this report has been downloaded from Kaggle and can be found here: *https://www.kaggle.com/adityasingh3519/playstore-app-downloads-prediction?select=Train.csv*. The data comes from scraping the Google Playstore, resulting in the number of downloads being banded into categories such as 10,000+ and 100,000+ as this is how it is presented in the Playstore. This means that this is a machine learning classification problem where we need to predict which band of downloads each app falls into. A subset of the data available to me can be seen below:

Offered_By	Category	Rating	Reviews	Size	Price	Content_Rating
ps_id-34729	Education	4.17	477	26M	Free	Everyone
ps_id-1670	Books And Reference	4.72	7554	55M	Free	Everyone
ps_id-7484	Game Adventure	4.18	36772	3.1M	372.1542	Everyone

Last_Updated_On	Release_Version	OS_Version_Required	Downloads
Apr 18 2020	1.3	4.0.3 and up	10,000+
Mar 12 2019	2.0.3	2.3.3 and up	100,000+
Feb 18 2018	1.4.29	4.0.3 and up	100,000+

As can be seen above, there are 10 app characteristics that can be used to predict the downloads an app has. These characteristics include the developer id (*Offered_By*), app category (*Category*), average rating (*Rating*), number of reviews received (*Reviews*), download size (*Size*), price in US Cents (*Price*), content rating (*Content_Rating*), last update date (*Last_Updated_On*), current release name (*Release_Version*), and the Android version required (*OS_Version_Required*).

Analysis

Preparing the Data

The first step that I need to do is to download the dataset from the internet and transform it into a workable dataframe:

```r
# Download the dataset
temp <- tempfile()
download.file(
  "https://raw.githubusercontent.com/JeffG05/AppRatingsHarvardX/master/data.csv", temp)

# Convert to CSV
data_set <- read.csv(temp, header = TRUE)
```

Next, I need to prepare the data so that I can use it in my models. The end goal is to be able to create a matrix that only contains numeric values as this is the easiest form to train a model on. To achieve this, I first removed the *Release_Version* and *Size* columns as I would not be using those for my models. I then converted the *Last_Updated_On* dates into the number of days since epoch and substituted each value in *Offered_By*, *Category*, *Content_Rating* and *OS_Version_Required* with an integer. Finally, I made sure that a value of '*Free*' in the *Price* column is converted to a value of 0.

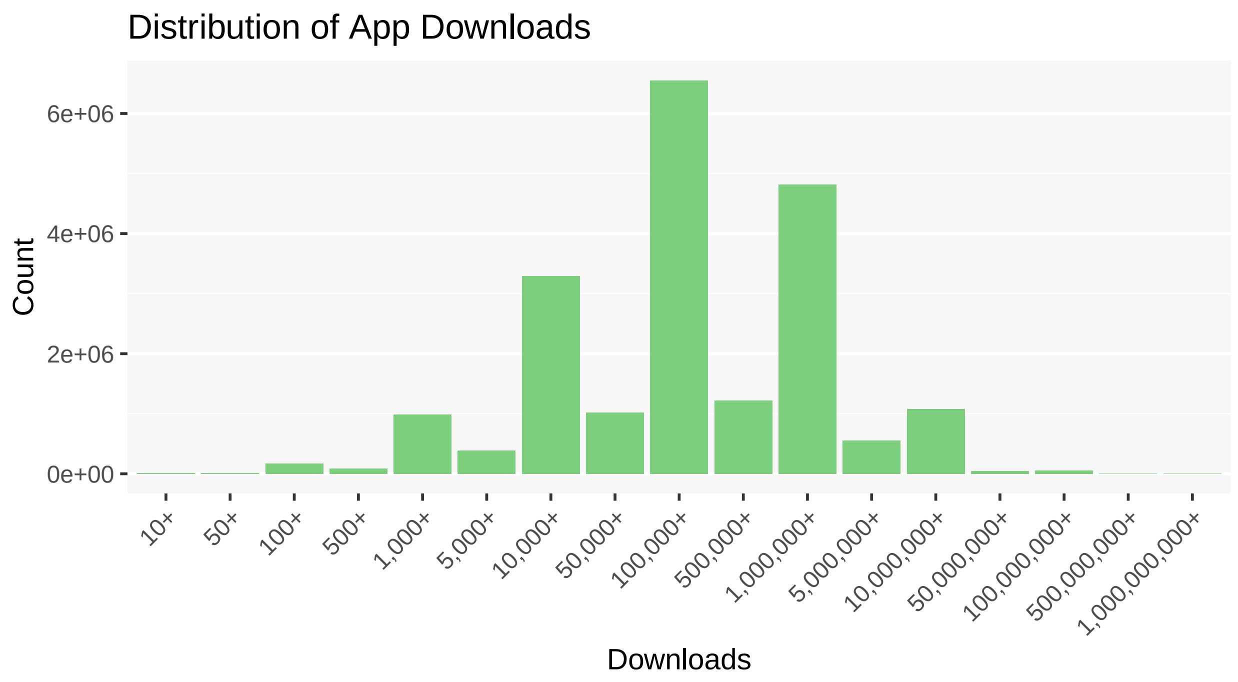
```r
# Select useful columns and convert to numeric values
data_set <- data_set %>%
  select(Reviews, Price, Rating, Category, Last_Updated_On, Content_Rating, Offered_By,
         OS_Version_Required, Downloads) %>%
  mutate(Last_Updated_On = as.numeric(as_date(Last_Updated_On, format = "%b %d %Y")),
         Offered_By = as.numeric(factor(Offered_By)),
         Category = as.numeric(factor(Category)),
         Content_Rating = as.numeric(factor(Content_Rating)),
         OS_Version_Required = as.numeric(factor(OS_Version_Required)),
         Price = replace_na(as.numeric(Price), 0))
```

Now that I have the final dataframe, I can split it into a validation, test and train set. The validation set will contain 10% of the data and will not be involved in the training of any model, which allows me to use it to test how my models perform at the end. The test set will be 10% of the remaining data and will be used during parameter tuning to test the performance of various parameter values. The train set is the remaining data not included in the validation or test set and will be used to train the machine learning models. I chose to use 90/10 splits for the data as this maximised the amount of data I had available to train whilst retaining a large enough set to validate the model with.

```r
# Split dataset into train, test and validation dataframes
data_ind <- createDataPartition(data_set$Downloads, p = 0.9, list = FALSE)
validation_set <- data_set %>% slice(-data_ind)
data_set <- data_set %>% slice(data_ind)

train_ind <- createDataPartition(data_set$Downloads, p = 0.9, list = FALSE)
test_set <- data_set %>% slice(-train_ind)
train_set <- data_set %>% slice(train_ind)

# Convert dataframes to matrices
validation_x <- validation_set %>% select(-Downloads) %>% as.matrix()
validation_y <- validation_set %>% select(Downloads) %>% as.matrix() %>% factor()

test_x <- test_set %>% select(-Downloads) %>% as.matrix()
test_y <- test_set %>% select(Downloads) %>% as.matrix() %>% factor()

train_x <- train_set %>% select(-Downloads) %>% as.matrix()
train_y <- train_set %>% select(Downloads) %>% as.matrix() %>% factor()
```

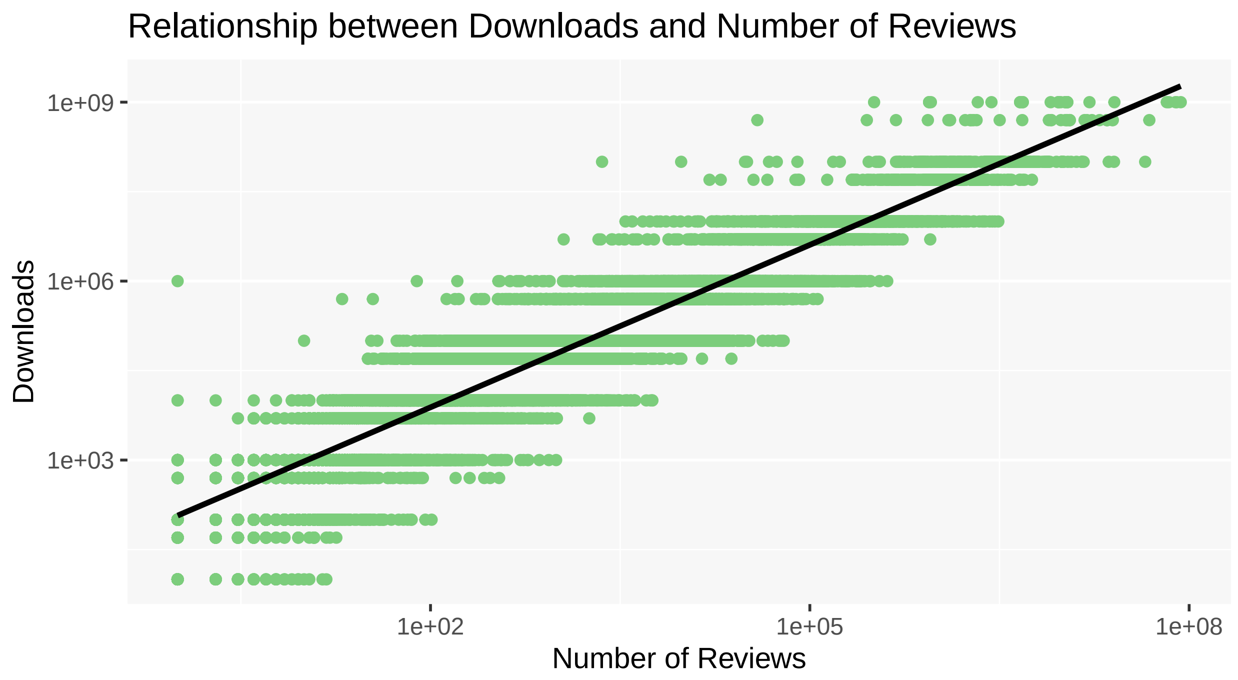
Exploring the Data

As mentioned earlier, the number of downloads an app has is banded into different groups. The bands that are present in the dataset as well as their distribution can be seen below:

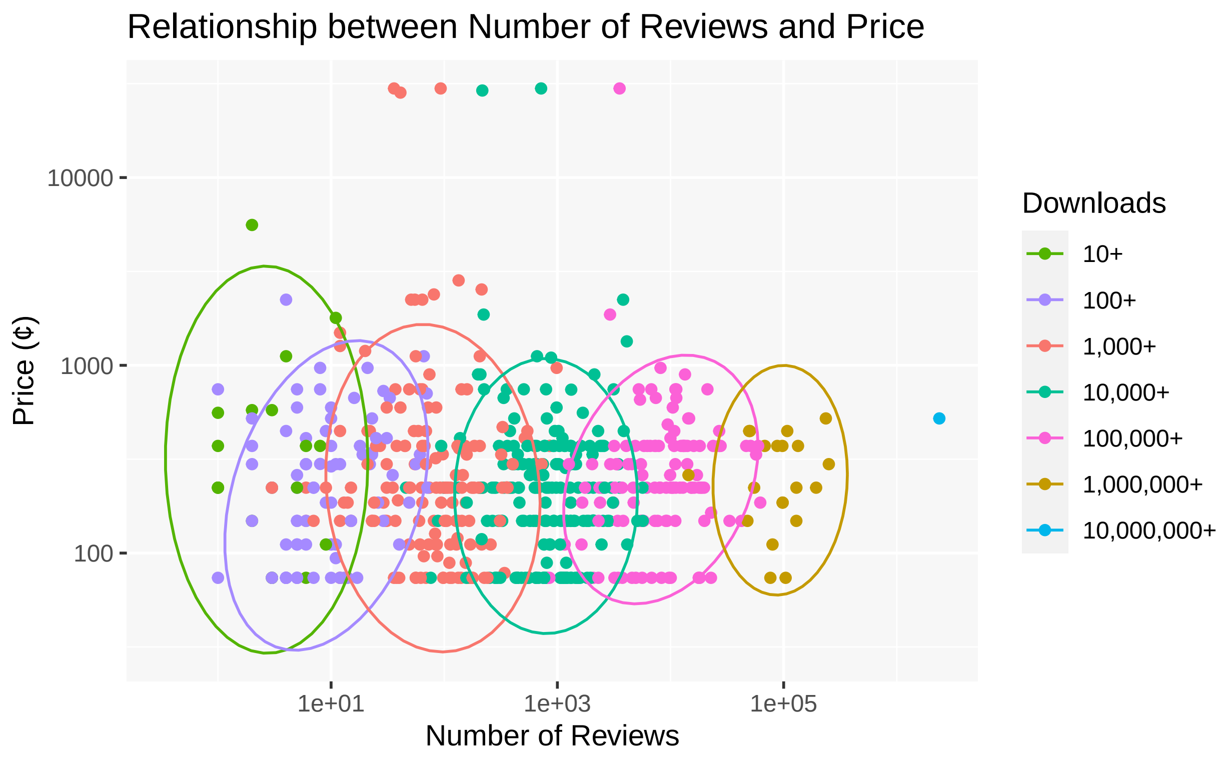


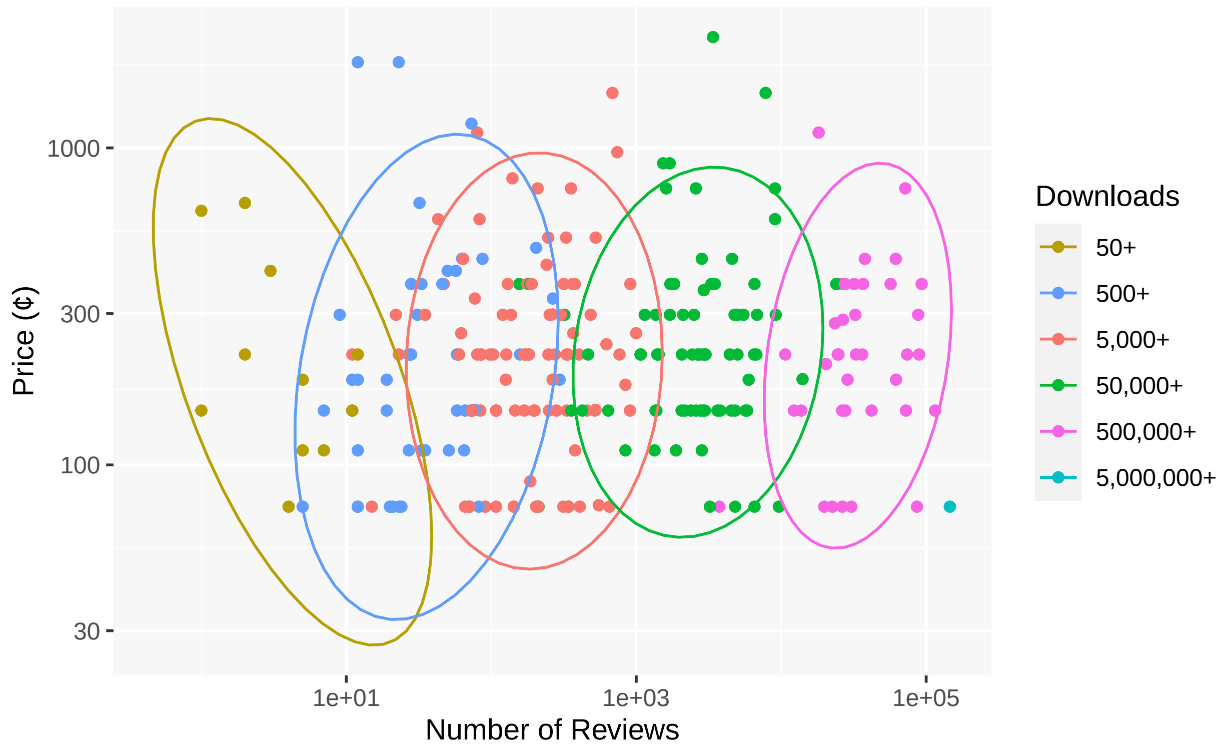
One immediate observation from the graph above is that there are far more apps in download bands that are in the form 1×10^x (e.g. 10,000+) than those that are in the form of 5×10^x (e.g. 50,000+).

When looking at how some of the app characteristics affect the number of downloads, I noticed that downloads and number of reviews received were highly correlated:



Another interesting find was when plotting *Reviews* against *Price* for all paid apps, you are able to group the different download bands into ellipses. Although these groups heavily overlap with each other, it might be a good starting point for a classification algorithm. I have separated the graph below into two by the starting digit of the download band to prevent the graph becoming too overcrowded.





Training the Model

Since this is a classification problem, I decided to test out the KNN and Random Forest algorithms on my data. My dataset has eight variables that can be used to train and since using different variables in the model can have an effect on the result, I decided to test a few different models for each algorithm. I started with training models that only looked at the *Reviews* and *Price* characteristics as during my exploration these seemed to keep the download bands grouped together. For subsequent models, I kept adding a variable to the previous model until I had a model using all eight variables. Therefore, I ended up training seven different models for both of the algorithms. The combination of variables used for the seven models are as follows:

[1] Reviews,Price
[2] Reviews,Price,Rating
[3] Reviews,Price,Rating,Category
[4] Reviews,Price,Rating,Category,Last_Updated_On
[5] Reviews,Price,Rating,Category,Last_Updated_On,Content_Rating
[6] Reviews,Price,Rating,Category,Last_Updated_On,Content_Rating,Offered_By
[7] Reviews,Price,Rating,Category,Last_Updated_On,Content_Rating,Offered_By,OS_Version_Required

The code to train and evaluate the accuracy of all the models can be seen below:

```r
# Get accuracies for all KNN models
knn_accuracy <- sapply(seq(2, 8), function(col) {
  # Select columns to train model on
  knn_x <- train_x[, 1:col]
  knn_x_test <- test_x[, 1:col]

  # Tune parameters and train a final model
  index <- sample(nrow(knn_x), 2000)
  train_knn <- train(knn_x[index,],
                     train_y[index],
                     method = "knn",
                     tuneGrid = data.frame(k = seq(3, 200, 2)))
  fit_knn <- knn3(knn_x, train_y, k = train_knn$bestTune$k)

  # Test model on the test set and return the accuracy
  y_hat_knn <- predict(fit_knn, knn_x_test, type = "class")
  cm_knn <- confusionMatrix(y_hat_knn, test_y)
  cm_knn$overall["Accuracy"]
})

# Get accuracies for all Random Forest models
rf_accuracy <- sapply(seq(2, 8), function(col) {
  # Select columns to train model on
  rf_x <- train_x[, 1:col]
  rf_x_test <- test_x[, 1:col]

  # Tune parameters and train a final model
  train_rf <- train(rf_x,
                     train_y,
                     method = "rf",
                     trControl = trainControl(method = "cv", number = 5),
                     tuneGrid = data.frame(mtry = seq(5, 100, 5)),
                     nSamp = 5000)
  fit_rf <- randomForest(rf_x, train_y, minNode = train_rf$bestTune$mtry)
  # Test model on the test set and return the accuracy
  y_hat_rf <- predict(fit_rf, rf_x_test, type = "class")
  cm_rf <- confusionMatrix(y_hat_rf, test_y)
  cm_rf$overall["Accuracy"]
})
```

The results for these models were:

	KNN	RF
Reviews, Price	0.5246788	0.4651792
+ Rating	0.5267072	0.4895199
+ Category	0.5118323	0.5665991
+ Last_Updated_On	0.5037187	0.5632184
+ Content_Rating	0.5084517	0.5753888
+ Offered_By	0.4658553	0.5807978
+ OS_Version_Required	0.4597701	0.5747126

Accuracy in this report is measured as the proportion of apps where the model correctly identified the download band. The results show that the best model used the Random Forest algorithm and took into consideration all variables available except the *OS_Version_Required*, resulting in an accuracy of 0.581. An interesting pattern also arose as the KNN algorithm outperformed the Random Forest algorithm when few variables were used, however this switched as the number of variables increased.

I then tested an ensemble model where I averaged the probabilities from the best KNN and best Random Forest models. For each app, the download band that had the highest average probability of being correct was chosen. The code and result of this ensemble model can be seen below:

```
# Select columns that performed best in KNN models
best_knn_x <- train_x[, 1:3]
best_knn_x_test <- test_x[, 1:3]

# Retrain the best KNN model
best_index <- sample(nrow(best_knn_x), 2000)
best_train_knn <- train(best_knn_x[best_index,],
                train_y[best_index],
                method = "knn",
                tuneGrid = data.frame(k = seq(3, 200, 2)))
best_fit_knn <- knn3(best_knn_x, train_y, k = best_train_knn$bestTune$k)

# Obtain probability values for the test set using the best KNN model
best_p_hat_knn <- predict(best_fit_knn, best_knn_x_test, type = "prob")

# Select columns that performed best in RF models
best_rf_x <- train_x[, 1:7]
best_rf_x_test <- test_x[, 1:7]

# Retrain the best RF model
best_train_rf <- train(best_rf_x,
                train_y,
                method = "rf",
                trControl = trainControl(method = "cv", number = 5),
                tuneGrid = data.frame(mtry = seq(5, 100, 5)),
                nSamp = 5000)
best_fit_rf <- randomForest(best_rf_x, train_y, minNode = best_train_rf$bestTune$mtry)

# Obtain probability values for the test set using the best RF model
best_p_hat_rf <- predict(best_fit_rf, best_rf_x_test, type = "prob")

# Obtain ensemble predictions using averaged KNN and RF probabilities
ensemble_p_hat <- (best_p_hat_knn + best_p_hat_rf) / 2
ensemble_max_cols <- max.col(ensemble_p_hat, ties.method = "random")
ensemble_y_hat <- colnames(ensemble_p_hat)[ensemble_max_cols] %>% factor()
cm_ensemble <- confusionMatrix(ensemble_y_hat, test_y)
```

	Accuracy
Best KNN	0.5267072
Best RF	0.5807978
Ensemble	0.5625423

The results of this ensemble shows that although it performs better than the KNN model, it does not outperform the Random Forest model. This means that currently the Random Forest model is still the strongest model I have trained. In a final attempt to improve on the Random Forest model, I decided to retrain the KNN and Random Forest models but this time I removed any dataset that was in a download band in the form of 5×10^x. This is because we saw earlier that these download bands were a lot less common than the bands in the form of 1×10^x and therefore I hoped that the overall accuracy would increase by forcing the algorithm to predict the more common download bands.

```
# Remove download bands from training dataset
train_x <- train_set %>%
  filter(str_starts(Downloads, "1")) %>%
  select(-Downloads) %>%
  as.matrix()
train_y <- train_set %>%
  filter(str_starts(Downloads, "1")) %>%
  select(Downloads) %>%
  as.matrix() %>%
  factor()

# Get accuracies for all KNN models
knn_accuracy <- sapply(seq(2, 8), function(col) {
  knn_x <- train_x[, 1:col]
  knn_x_test <- test_x[, 1:col]

  index <- sample(nrow(knn_x), 2000)
  train_knn <- train(knn_x[index,],
                     train_y[index],
                     method = "knn",
                     tuneGrid = data.frame(k = seq(3, 200, 2)))
  fit_knn <- knn3(knn_x, train_y, k = train_knn$bestTune$k)

  y_hat_knn <- predict(fit_knn, knn_x_test, type = "class")
  cm_knn <- confusionMatrix(y_hat_knn, test_y)
  cm_knn$overall["Accuracy"]
```

```r
})

# Get accuracies for all RF models
rf_accuracy <- sapply(seq(2, 8), function(col) {
  rf_x <- train_x[, 1:col]
  rf_x_test <- test_x[, 1:col]

  train_rf <- train(rf_x,
                    train_y,
                    method = "rf",
                    trControl = trainControl(method = "cv", number = 5),
                    tuneGrid = data.frame(mtry = seq(5, 100, 5)),
                    nSamp = 5000)
  fit_rf <- randomForest(rf_x, train_y, minNode = train_rf$bestTune$mtry)

  y_hat_rf <- predict(fit_rf, rf_x_test, type = "class")
  cm_rf <- confusionMatrix(y_hat_rf, test_y)
  cm_rf$overall["Accuracy"]
})
```

	KNN	RF
Reviews, Price	0.5152130	0.4678837
+ Rating	0.5145368	0.4861393
+ Category	0.5104801	0.5490196
+ Last_Updated_On	0.5003381	0.5496957
+ Content_Rating	0.5023665	0.5530764
+ Offered_By	0.4557133	0.5496957
+ OS_Version_Required	0.4617985	0.5524003

The results above show that this model does not improve on the previous best accuracy of 0.581, achieved by the Random Forest Model. However, it was interesting to see that the accuracies achieved when removing nearly half the possible download bands from the training dataset did not differ by a lot when compared to the models where all download bands were present. This is impressive since, by removing half of the possible download bands, the model is accepting that it is guaranteed to predict some app downloads wrong - the maximum accuracy that the models could have achieved on the test set was 0.698.

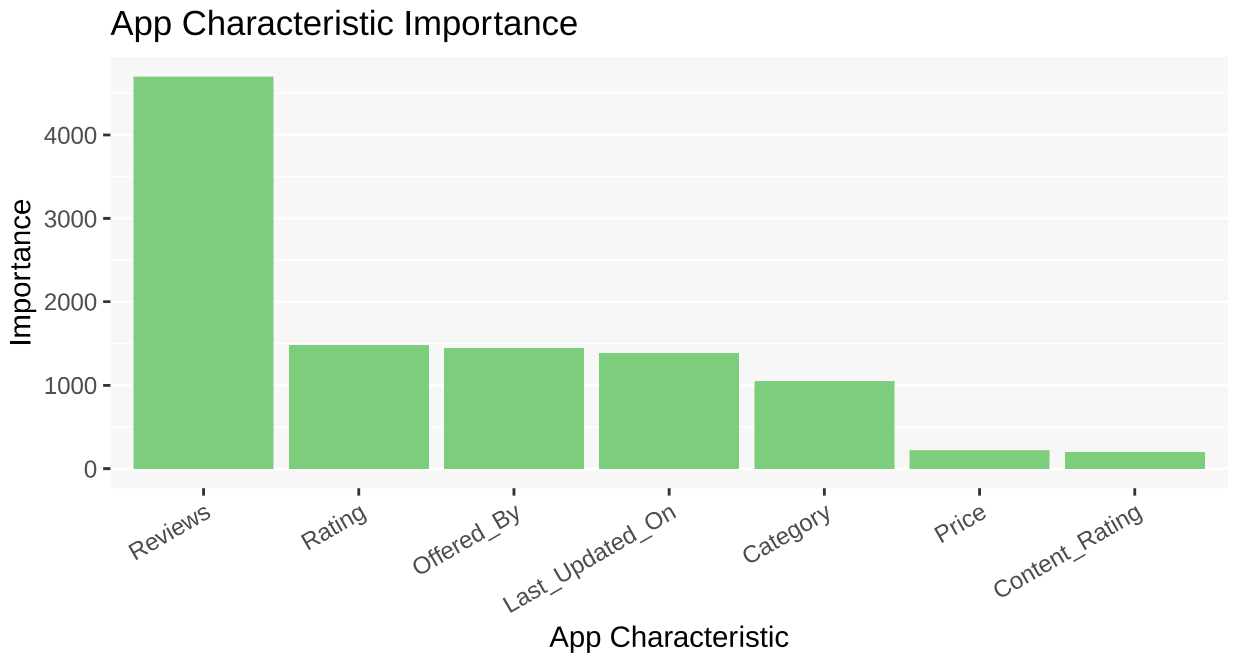
Results

After training all the models, I have found that the best model used all download bands when training, used the Random Forest algorithm and used seven app characteristics (Reviews, Price, Rating, Category, Last Updated Date, Content Rating & App Developer). This model had an accuracy of 0.581 on the test set, however in order to see the true accuracy of the model, I had to run it on the validation set. This is because the test set was involved in the parameter tuning in the model and therefore often makes the model perform better than it does on average. After running the model on the validation set, I obtain a final accuracy of:

Final Accuracy: 0.5723844

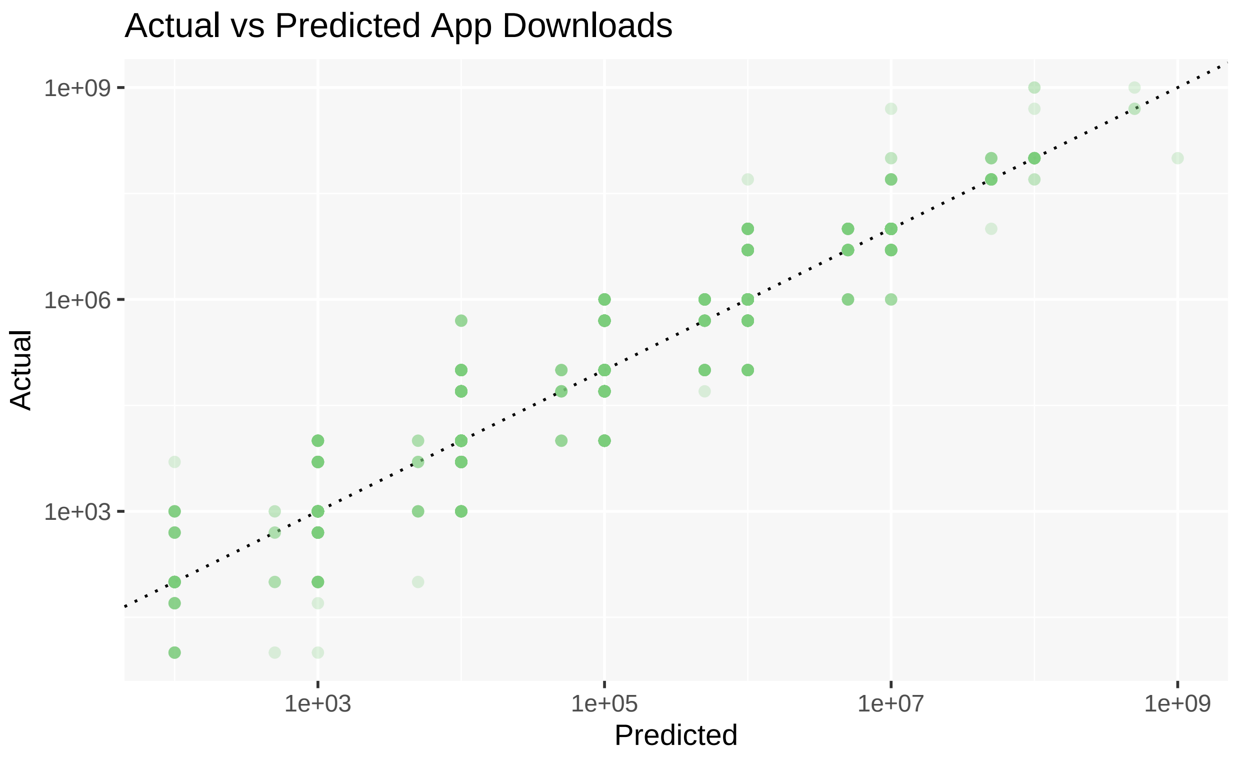
As expected, the final accuracy dropped slightly but not by a huge amount. This does not seem like a very high accuracy score since only 57% of apps had their download band correctly predicted. Therefore, it is important to explore the final model to understand its strengths and weaknesses.

To give an insight in which app characteristics are most influential, I can plot each app characteristic used in order of importance in the model:



The barplot above shows how the *Reviews* characteristic is by far the most influential app characteristic in the final model. This makes sense as I noticed the strong correlation between *Reviews* and *Downloads* earlier in the report. The *Price* and *Content Rating* ended up having the lowest importance in the final model.

Plotting the predicted downloads against the actual app downloads reveals that the model might be a lot stronger than the final accuracy might suggest:



The actual vs predicted graph clearly follows the dotted reference line which shows that the model is relatively good at predicting roughly the downloads an app has. This also highlights how the relatively low accuracy of 0.572 might be misleading due to the difficulty of predicting the correct download band out of a possibility of 17 bands - an algorithm that just guessed would have an accuracy of around 0.0588. Perhaps a more telling accuracy score would be if we allowed the algorithm to predict one band above and below the actual download band - this yields an accuracy of 0.883, which is relatively high and proves that this model is actually reasonably good at predicting app downloads.

Conclusion

The aim of this report was to create a machine learning algorithm, capable of predicting the downloads an app will receive given some of the characteristics of the app. Now that I have trained and selected my final model, I am confident that I have achieved this aim. My model was able to predict the correct download band for 57% of the apps and that accuracy increases to 88% if you allow an error margin of one download band either side. This model provides app developers with insights into what features of their app influence app downloads, with one key insight from the app characteristic importance plot being that the price of an app does not have a huge impact on app downloads.

Although the final model does not have any clear weaknesses, there is always the possibility that a model using a different algorithm could be more accurate. Moving forward, I would want to explore interpreting this task as a regression problem instead of a classification problem and see whether this could outperform the current model, however a dataset with precise app downloads would be more beneficial for such a regression model.