AppRatings

John Wilson

8/17/2020

Executive Summary

People are attached to their technology devices more and more every year which leads to them looking for more apps. Often times these apps are for increase in productivity and other times for stress relief in forms of games and other things. By analyzing this data set, it can be determined what is more important in a good rating and where there may be a shortage of available apps. 6 different models were trained on this data before doing a final confirmation on the data set. The matrix factorization analysis proved to be the best model with an RMSE of 0.512 and a final RMSE on the validation set of 0.521. By the increase in RMSE from training to validation, it could be interpreted as over trained.

Data Preparation and Review

This data was downloaded from kaggle and came in a rather tidy format already. The data came all in one file and was prepared to be reviewed.

```r
if(!require(tidyverse)) install.packages("tidyverse", repos =
"http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-
project.org")
if(!require(data.table)) install.packages("data.table", repos =
"http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos =
"http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem", repos =
"http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-
project.org")

library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(recosystem)
library(knitr)

# https://www.kaggle.com/gauthamp10/google-playstore-apps

dl <- tempfile()
download.file("https://github.com/john-h-
wilson09/comfortdata/raw/master/androidapps.zip", dl)
```

```
dat <- read.csv(unzip(dl,"googleplaystore.csv"))
rm(dl)

str(dat) #13 variables, last four variables appear useless

## 'data.frame':    10841 obs. of  13 variables:
##  $ App           : Factor w/ 9660 levels "- Free Comics - Comic Apps",..:
7229 2563 8998 8113 7294 7125 8171 5589 4948 5826 ...
##  $ Category      : Factor w/ 34 levels "1.9","ART_AND_DESIGN",..: 2 2 2 2
2 2 2 2 2 2 ...
##  $ Rating        : num  4.1 3.9 4.7 4.5 4.3 4.4 3.8 4.1 4.4 4.7 ...
##  $ Reviews       : Factor w/ 6002 levels "0","1","10","100",..: 1183 5924
5681 1947 5924 1310 1464 3385 816 485 ...
##  $ Size          : Factor w/ 462 levels "1,000+","1.0M",..: 55 30 368 102
64 222 55 118 146 120 ...
##  $ Installs      : Factor w/ 22 levels "0","0+","1,000,000,000+",..: 8 20
13 16 11 17 17 4 4 8 ...
##  $ Type          : Factor w/ 4 levels "0","Free","NaN",..: 2 2 2 2 2 2 2 2
2 2 ...
##  $ Price         : Factor w/ 93 levels "$0.99","$1.00",..: 92 92 92 92 92
92 92 92 92 92 ...
##  $ Content.Rating: Factor w/ 7 levels "","Adults only 18+",..: 3 3 3 6 3 3
3 3 3 3 ...
##  $ Genres        : Factor w/ 120 levels "Action","Action;Action &
Adventure",..: 10 13 10 10 12 10 10 10 10 12 ...
##  $ Last.Updated  : Factor w/ 1378 levels "1.0.19","April 1, 2016",..: 562
482 117 825 757 901 76 726 1317 670 ...
##  $ Current.Ver   : Factor w/ 2834 levels "","0.0.0.2","0.0.1",..: 121 1020
466 2827 279 115 279 2393 1457 1431 ...
##  $ Android.Ver   : Factor w/ 35 levels "","1.0 and up",..: 17 17 17 20 22
10 17 20 12 17 ...

sum(is.na(dat)) #checking for NAs, many missing values

## [1] 1474

dat1 <- na.omit(dat) #remove NAs
```

From the lines of code above several NA values were noted and then removed to prevent any analysis errors. This data set included over 8000 different apps in the Google playstore in 116 different genres.

```
# Number of unique apps
length(unique(dat1$App))

## [1] 8197

# Number of unique genres
length(unique(dat1$Genres))
```

```
## [1] 116
```

Looking deeper into the applications, it is discovered that most apps are free or under $3.

```
dat1 %>% group_by(Price) %>% summarize(n=n()) %>%
  arrange(desc(n)) %>% head(10) #most apps are free or under $3
```

```
## # A tibble: 10 x 2
##    Price     n
##    <fct> <int>
##  1 0      8719
##  2 $2.99   114
##  3 $0.99   107
##  4 $4.99    70
##  5 $1.99    59
##  6 $3.99    58
##  7 $1.49    31
##  8 $2.49    21
##  9 $5.99    18
## 10 $9.99    16
```

As it would be expected not all apps have nearly the same amount of installs and consequently varying amounts of reviews. Note that most apps have less than 5 reviews and a rating in the range of 4.3-4.5

```
## # A tibble: 10 x 2
##    Reviews     n
##    <fct>   <int>
##  1 2          83
##  2 3          78
##  3 4          74
##  4 5          74
##  5 1          67
##  6 6          62
##  7 7          62
##  8 8          57
##  9 12         53
## 10 10         46
```

```
dat1 %>% group_by(Rating) %>% summarize(n=n()) %>% arrange(desc(n)) %>%
  head(5) #most apps are rated 4.3-4.5
```

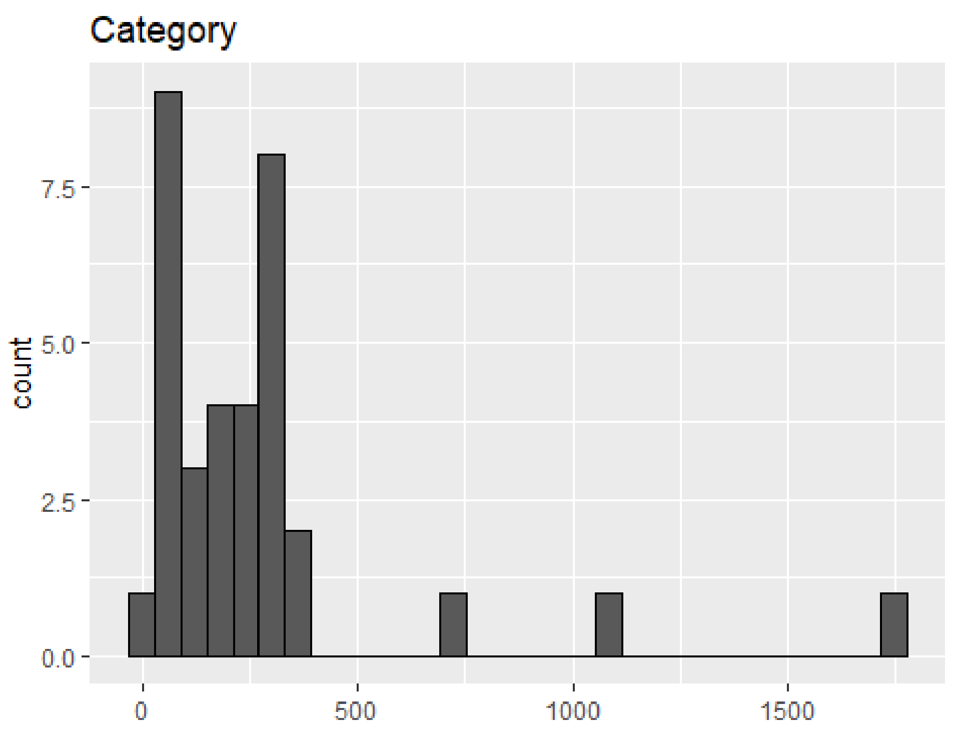
```
## # A tibble: 5 x 2
##   Rating     n
##    <dbl> <int>
## 1    4.4  1109
## 2    4.3  1076
## 3    4.5  1038
## 4    4.2   952
## 5    4.6   823
```

The table below will also display the various categories that represent the top installs. Not coincidentally, most of the top installed applications in the Google play store are Google apps that are installed by default on the android platform.

```
# Seeing the top installed apps
dat1 %>% arrange(desc(Installs)) %>% tail(10) %>%
select(App,Category,Rating,Reviews,Installs,Price)

##                                                App          Category Rating
## 9358                  Maps - Navigate & Explore   TRAVEL_AND_LOCAL    4.3
## 9359 Messenger â\200“ Text and Video Chat for Free      COMMUNICATION
4.0
## 9360                                        Google+            SOCIAL    4.2
## 9361                                         Google             TOOLS    4.4
## 9362                                       Hangouts      COMMUNICATION    4.0
## 9363                                   Google Drive       PRODUCTIVITY    4.4
## 9364                  Skype - free IM & video calls      COMMUNICATION    4.1
## 9365                                  Google Photos        PHOTOGRAPHY    4.5
## 9366                              Google Play Games            FAMILY    4.3
## 9367                                    Google News NEWS_AND_MAGAZINES    3.9
##       Reviews       Installs Price
## 9358  9231613 1,000,000,000+     0
## 9359 56642847 1,000,000,000+     0
## 9360  4828372 1,000,000,000+     0
## 9361  8021623 1,000,000,000+     0
## 9362  3419464 1,000,000,000+     0
## 9363  2728941 1,000,000,000+     0
## 9364 10484169 1,000,000,000+     0
## 9365 10847682 1,000,000,000+     0
## 9366  7168735 1,000,000,000+     0
## 9367   878065 1,000,000,000+     0
```

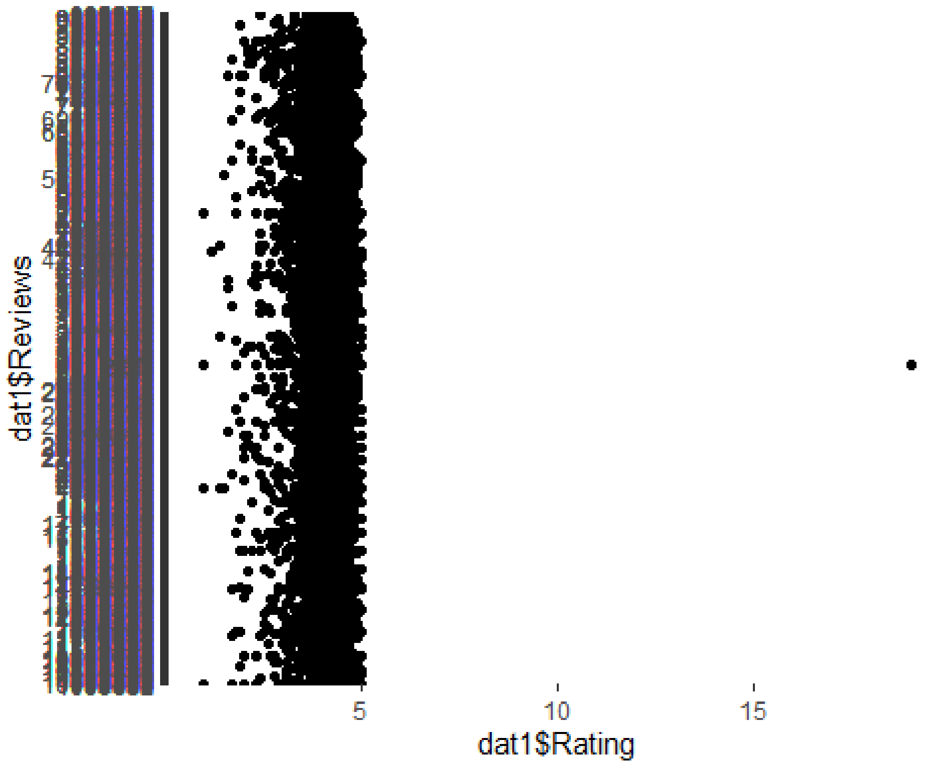
The applications that appear most often and therefore, have the highest competition are in the family and game sectors. By the chart and table, it is easily seen the different levels of activity in the categories.



```
dat1 %>% group_by(Category) %>% summarize(n=n()) %>% arrange(desc(n)) %>%
  head(5) #top 5 categories for playstore apps
```

```
## # A tibble: 5 x 2
##   Category         n
##   <fct>        <int>
## 1 FAMILY        1747
## 2 GAME          1097
## 3 TOOLS          734
## 4 PRODUCTIVITY   351
## 5 MEDICAL        350
```

With a quick plot of reviews vs ratings, an outlier could be seen. The outlier was filtered out by only keeping ratings of 5 or less. Also the chart does not display there being any relationship between the number of reviews and ratings.



```r
dat1 <- dat1 %>% filter(Rating<6) #filter out outlier
```

To provide data for the training and validation of models the data was split into practice and validation sets. 15% of the data points were set aside for validation and then it was confirmed that the categories and genres in validation existed in the practice set.

```r
# Dividing the data to have a practice set and and validation set
set.seed(15,sample.kind = "Rounding")
valid_index <- createDataPartition(dat1$Rating,times = 1,list = FALSE,p=0.15)
practice <- dat1[-valid_index,]
validation <- dat1[valid_index,]

# Making sure valid set info is in practice set also
validation <- validation %>% semi_join(practice, by = "Genres") %>%
semi_join(practice, by = "Category")
```

The practice data set was further dissected into training and testing sets with only 20% of the set being placed in the test set. Again, it was confirmed that the genres and categories in the test set also existed in the training set.

```r
# Dividing the practice set to have data to train and test with
test_index <- createDataPartition(practice$Rating,times = 1,list =
FALSE,p=0.2)
train_set <- practice[-test_index,]
test_set <- practice[test_index,]

# Making sure the test set info is also in the train set
```

```
test_set <- test_set %>% semi_join(train_set, by = "Genres") %>%
semi_join(train_set, by = "Category")
```

Now that the data has been reviewed thoroughly and divided up for analysis, processing can begin.

Technical Analysis

The first approach was to just apply the average from the train set to the test set, as a ‘naive’ approach. Root mean square error(RMSE) will be used as a grade for best model, thus a table was created to store the results from each model in.

```
mu <- mean(train_set$Rating)
avg_pred <- rep(mu,length(test_set$Rating))
avg_rmse <- RMSE(avg_pred,test_set$Rating)
RMSE_result0 <- data_frame(Model = "Naive-Avg", RMSE = avg_rmse) #create
table for results
```

As stated previously in this report, categories and genres are not all evenly represented so there could be a bias there. Most visitors to the app store are usually looking for an app within a sector for a purpose so some genres and categories are likely graded harder. In addition to genres and categories, another analysis was performed to access a content rating bias.

```
cat_avg <- train_set %>% group_by(Category) %>% summarize(b_c=mean(Rating-
mu))
cat_pred <- test_set %>% left_join(cat_avg,by='Category') %>%
  mutate(pred=mu+b_c) %>% pull(pred)
cat_rmse <- RMSE(cat_pred,test_set$Rating)
RMSE_result1 <- bind_rows(RMSE_result0, data_frame(Model="Category Effect
Model", RMSE = cat_rmse ))

genre_avg <- train_set %>% left_join(cat_avg, by='Category') %>%
  group_by(Genres) %>% summarize(b_g=mean(Rating-mu-b_c))

genre_pred <- test_set %>% left_join(cat_avg,by='Category') %>%
  left_join(genre_avg,by='Genres') %>% mutate(pred=mu+b_c+b_g) %>%
  pull(pred)
genre_rmse <- RMSE(genre_pred,test_set$Rating)
RMSE_result1 <- bind_rows(RMSE_result1, data_frame(Model="Category+Genre
Effect Model", RMSE = genre_rmse ))

content_avg <- train_set %>% left_join(cat_avg,by='Category') %>%
  left_join(genre_avg,by='Genres') %>% group_by(Content.Rating) %>%
  summarize(b_r = mean(Rating-mu-b_c-b_g))
content_pred <- test_set %>% left_join(cat_avg,by='Category') %>%
  left_join(genre_avg,by='Genres') %>%
  left_join(content_avg,by='Content.Rating') %>% mutate(pred=mu+b_c+b_g+b_r)
%>%
  pull(pred)
```

```r
content_rmse <- RMSE(content_pred,test_set$Rating)
RMSE_result1 <- bind_rows(RMSE_result1, data_frame(Model="Cat+Genre+Content
Model", RMSE = content_rmse ))
```

In order to not over train or lose an unknown correlation in the data shuffle, a regularized model was tried with lambdas ranging 2 to 10.

```r
lambdas <- seq(2,10,0.25)
RMSE_reg <- sapply(lambdas, function(l){
  cat_avg <- train_set %>% group_by(Category) %>% summarize(b_c=sum(Rating-
mu)/(n()+l))
  genre_avg <- train_set %>% left_join(cat_avg, by='Category') %>%
    group_by(Genres) %>% summarize(b_g=sum(Rating-mu-b_c)/(n()+l))

  preds <- test_set %>% left_join(cat_avg,by='Category') %>%
    left_join(genre_avg,by='Genres') %>% mutate(pred=mu+b_c+b_g) %>%
    pull(pred)

  RMSE(preds,test_set$Rating)
})
lambda <- lambdas[which.min(RMSE_reg)] #best lamda
RMSEreg_eff <- RMSE_reg[[which.min(RMSE_reg)]]
RMSE_result2 <- bind_rows(RMSE_result1, data_frame(Model="Regularized Eff
Model", RMSE = RMSEreg_eff ))
```

The final approach was to use matrix factorization. A residual data set was formed by removing all bias from the prior calculations and saving it. This model accesses the user-index interface using algorithms and 2 smaller rectangular matrices to form a correlation. Tuning parameters were provided and the best was determined from the training set to apply to the test set.

```r
# Matrix Factorization Model
# Calculating residuals by removing movie, user and genres effect
residual_set <- train_set %>% left_join(cat_avg,by='Category') %>%
  left_join(genre_avg,by='Genres') %>% mutate(resid=Rating-mu-b_c-b_g)

# Setting data and tuning parameters - used R documentation for parameters
reco <- Reco()
train_reco <- data_memory(user_index=residual_set$Category,
item_index=residual_set$Genres,
                          rating=residual_set$resid, index1=TRUE)
test_reco <- data_memory(user_index=test_set$Category,
item_index=test_set$Genres, index1=TRUE)

# This step takes a couple of minutes
opts <- reco$tune(train_reco, opts = list(dim = c(10,20,30,40,50),
                                          lrate = c(0.1, 0.2),
                                          costp_l1=0, costq_l1=0,
                                          nthread = 1, niter = 10))
```

```
# Train the model of reco
reco$train(train_reco, opts = c(opts$min, nthread=1, niter=20))
```

```
## iter      tr_rmse          obj
##    0       0.5046  1.6380e+003
##    1       0.5018  1.6051e+003
##    2       0.4999  1.5918e+003
##    3       0.4997  1.5904e+003
##    4       0.4996  1.5895e+003
##    5       0.4996  1.5897e+003
##    6       0.4995  1.5895e+003
##    7       0.4995  1.5892e+003
##    8       0.4995  1.5894e+003
##    9       0.4995  1.5893e+003
##   10       0.4995  1.5894e+003
##   11       0.4994  1.5889e+003
##   12       0.4994  1.5887e+003
##   13       0.4994  1.5888e+003
##   14       0.4994  1.5889e+003
##   15       0.4994  1.5888e+003
##   16       0.4993  1.5885e+003
##   17       0.4993  1.5886e+003
##   18       0.4993  1.5882e+003
##   19       0.4993  1.5884e+003
```

```
# Predict results for the test set
reco_pred <- reco$predict(test_reco, out_memory())
preds <- cbind(test_set, reco_pred) %>%
  left_join(cat_avg,by='Category') %>% left_join(genre_avg,by='Genres') %>%
  mutate(pred = mu + b_c + b_g + reco_pred) %>% pull(pred)
rmseMF <- RMSE(preds, test_set$Rating)
RMSE_result3 <- bind_rows(RMSE_result2, data_frame(Model="Matrix
Factorization", RMSE = rmseMF )) %>%
  arrange(desc(RMSE)) #arranging RMSE from largest to smallest to see best
model
```

Results

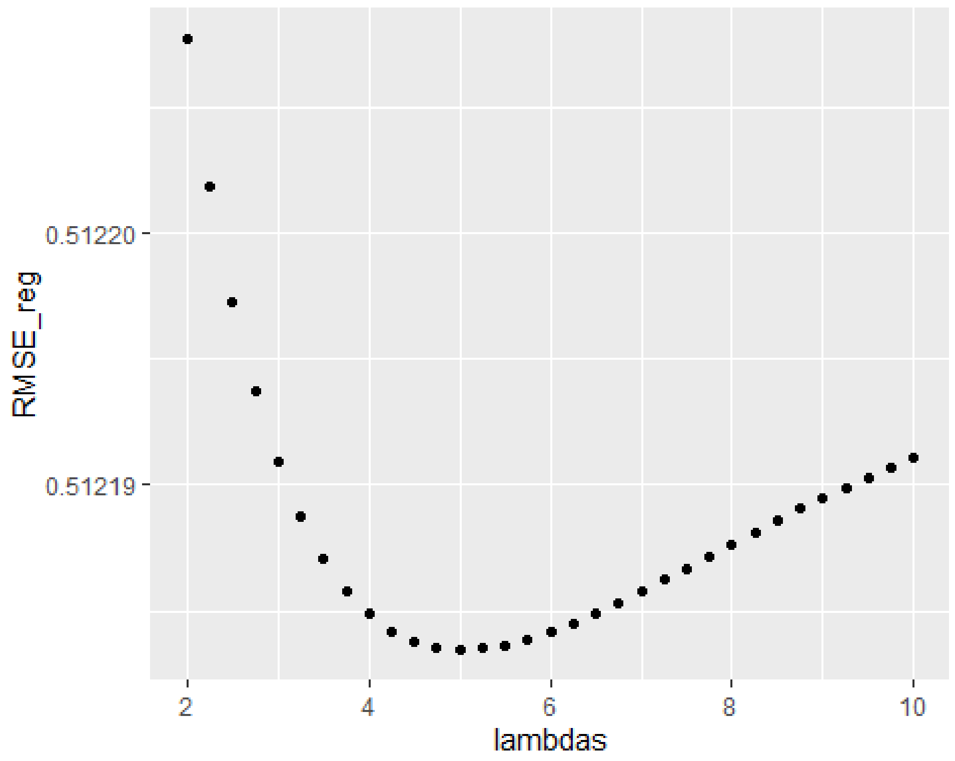
This data set was not an astronomical size so results did not take lengthy periods of calculations. The initial model of assuming the average applies resulted in an RMSE of 0.515817. From here, the goal was to continuously improve the models each step. Next the bias models were formed and layered together. Beginning with category, then adding genre and finally content rating. Surprisingly, adding the third bias slightly increased the RMSE.

```
kable(RMSE_result1)
```

Model	RMSE
Naive-Avg	0.5158170
Category Effect Model	0.5137402

Category+Genre Effect Model	0.5124689
Cat+Genre+Content Model	0.5129219

Next, was the training of the regularized model with the various lambdas between 2 and 10. This resulted in the optimum lambda being 5. Thus far, this model became the best RMSE though by only a small margin.



```r
kable(RMSE_result2)
```

Model	RMSE
Naive-Avg	0.5158170
Category Effect Model	0.5137402
Category+Genre Effect Model	0.5124689
Cat+Genre+Content Model	0.5129219
Regularized Eff Model	0.5121835

Lastly, was the more computing intensive model of matrix factorization which took a couple minutes to run. This yielded the best RMSE of the 6 models with an RMSE of 0.5120195. The table below was arranged so that the best model would be on the bottom line.

```r
kable(RMSE_result3)
```

Model	RMSE
Naive-Avg	0.5158170

Category Effect Model	0.5137402
Cat+Genre+Content Model	0.5129219
Category+Genre Effect Model	0.5124689
Regularized Eff Model	0.5121835
Matrix Factorization	0.5120195

This model was then implemented with the validation data set after confirming its variables were also in the training set. The matrix factorization tuned variables were applied to this data.

```
# Apply best model validation data
valid <- validation %>% semi_join(train_set, by = "Genres") %>%
semi_join(train_set, by = "Category")
valid_reco <- data_memory(user_index=valid$Category, item_index=valid$Genres,
index1=TRUE)
reco_val <- reco$predict(valid_reco, out_memory())
valid_pred <-  cbind(valid, reco_val) %>%
  left_join(cat_avg,by='Category') %>% left_join(genre_avg,by='Genres') %>%
  mutate(vpred = mu + b_c + b_g + reco_val) %>% pull(vpred)
valid_rmse <- RMSE(valid_pred,valid$Rating)
```

Unexpectantly, the RMSE for the validation set was above any RMSE calculated during the training of the models at an RMSE of 0.5212362.

Conclusion

More apps are being developed everyday, especially with people having more time at home and to work on ideas they previously thought they didn't have time for. This project showed the variations of apps and how they were received within the Google playstore. It would have been beneficial to have a data set including the Apple app store also to see how the data compares. Also, a bigger data set would have beeen a help. It could be considered that the models were over trained by the validation set having a higher RMSE than any of the training model RMSE. A complication that was observed was dealing with most of the variables being of the factor class and not converting easily to the numeric class; for example, the Reviews variable. Perhaps, another approach would be to have ratings of only whole and half numbers which could easily be factors to perform linear regressions on. Overall, it was a good learning experience and insightful data about the application world.