Movielens Report

Muhammad Ali

05/09/2020

Executive Summary

Movie recommendations have is a standard task in machine learning literature. One common approach is to predict the rating that a user would give if they were to watch a movie, and among those recommend the highest rated movie. This report details a rating prediction model created using the MovieLens dataset. It contains over 10 million movie ratings. Along with each rating the data set contains a user id of the user that rated it, the movie title, the movie id which is given to each movie uniquely, genre of the movie, and timestamp of when the rating occurred.

The approach taken in this report is of the following 3 phases. First explore the data, looking for insights into the relationships between movie features. Then a model will be constructed, which due to size of the dataset and way the data is given, is a repeated -feature selection and residual reduction- model. Because the model selected the exploration phase become critical and will be referred to whenever. Last the model is evaluated and future work is cited.

Methods/Analysis

Prior to exploration, it is clear that both the user effect and movie effect are going to be strong predictors of the rating of a movie. Hence the goal of our exploration will be:

- Identify other features that can help in predicting movie rating

The approach taken in this section is to identify the features that can discriminate the data into groups with different ratings the best.

This approach was chosen because it has to do with how the model will be trained/created. In summary the model will repeatedly select a feature to group by, then calculate and add the effect of those features to the model. Let's start by loading the data:

```r
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

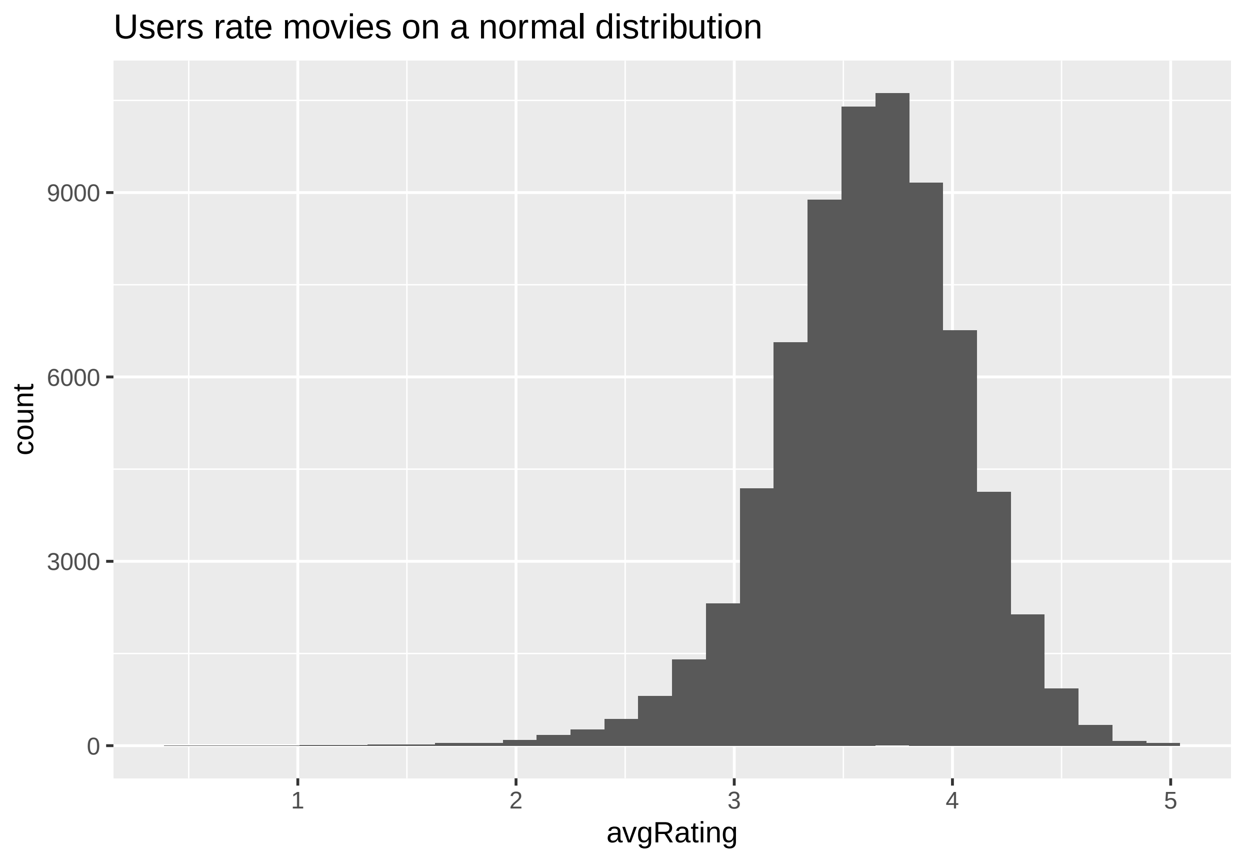
```r
library(tidyr)
library(stringr)
loadData <- function() {
  #I have previously saved edx and validation into rds format
  return (list(readRDS('edx.rds'), readRDS('validation.rds')))
  }
data <- loadData()
edx <- data[[1]]
validation <- data[[2]]
rm(data)
```

What are the groups of users?

I will try and explore high vs low users and picky vs not picky users.

How average user movie rating movies follows a normal distribution.

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

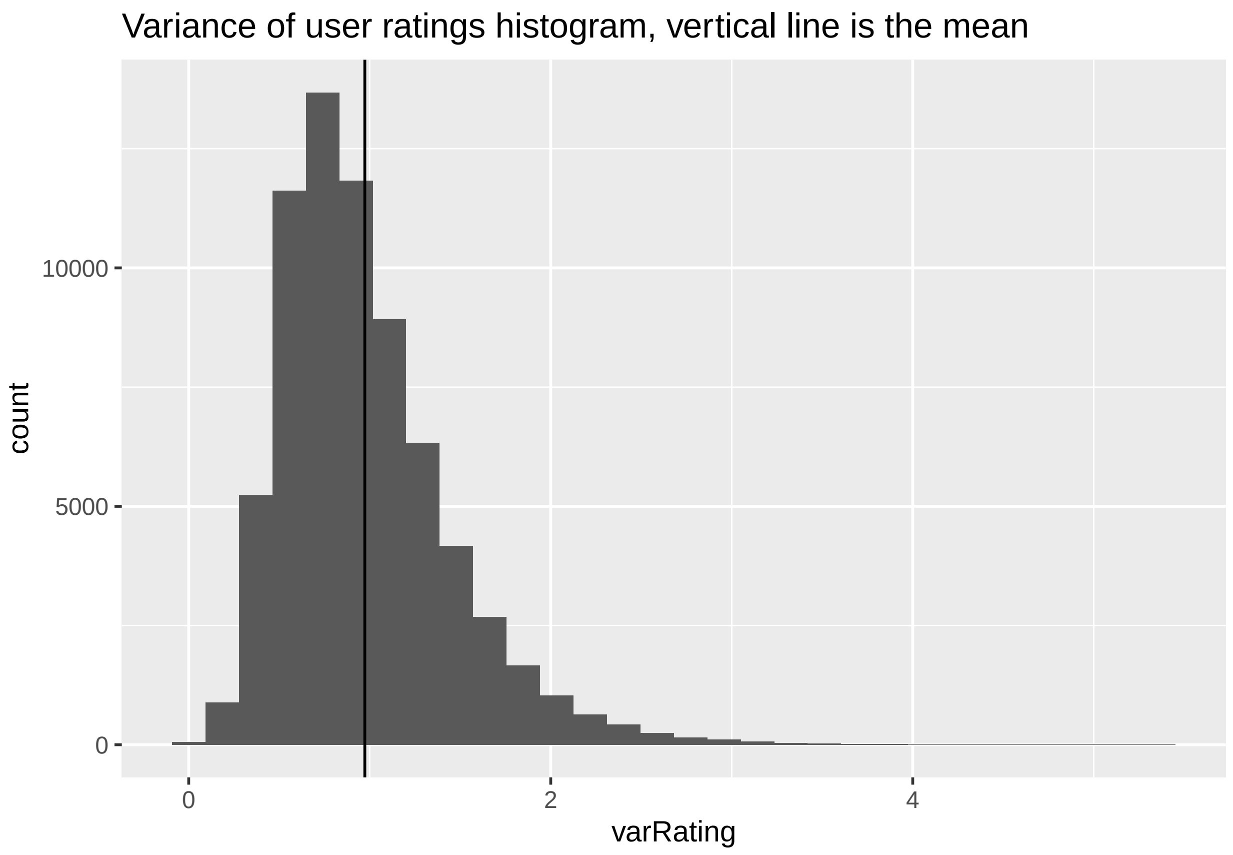


Looking at the summary table, users on average deviate from there average rating by 1 stars, which is not a lot. The vast majority of users on average rate between 3.4-3.8 stars

```
## # A tibble: 1 x 4
##   meanRating varoxfRating meanOfVarRating varOfVarofRating
##        <dbl>        <dbl>           <dbl>            <dbl>
## 1       3.61        0.220           0.973            0.220
```

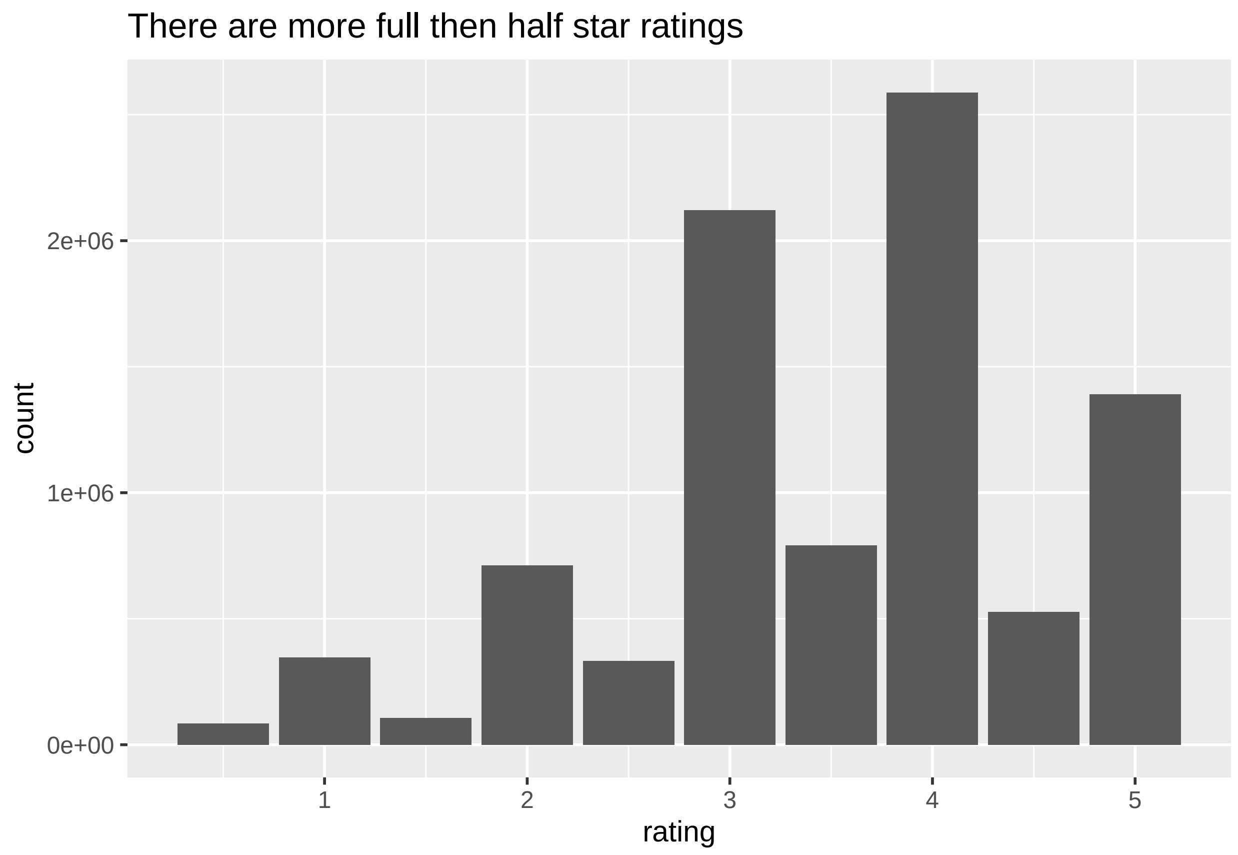
The variance of user ratings is right skewed, this means that there are plenty of picky raters that will rate some movies highly and others very low.

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



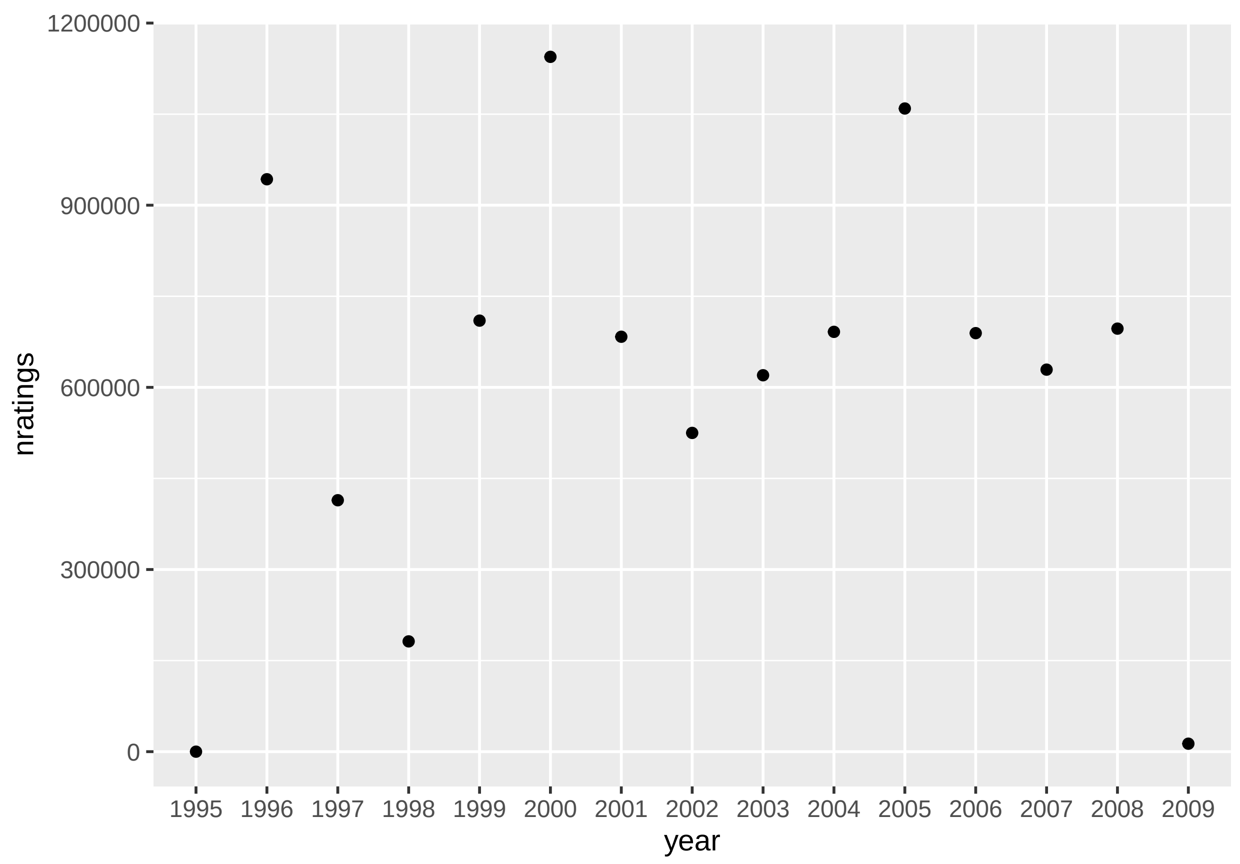
What is the distributions of ratings?

There are a lot more full stars then half stars.

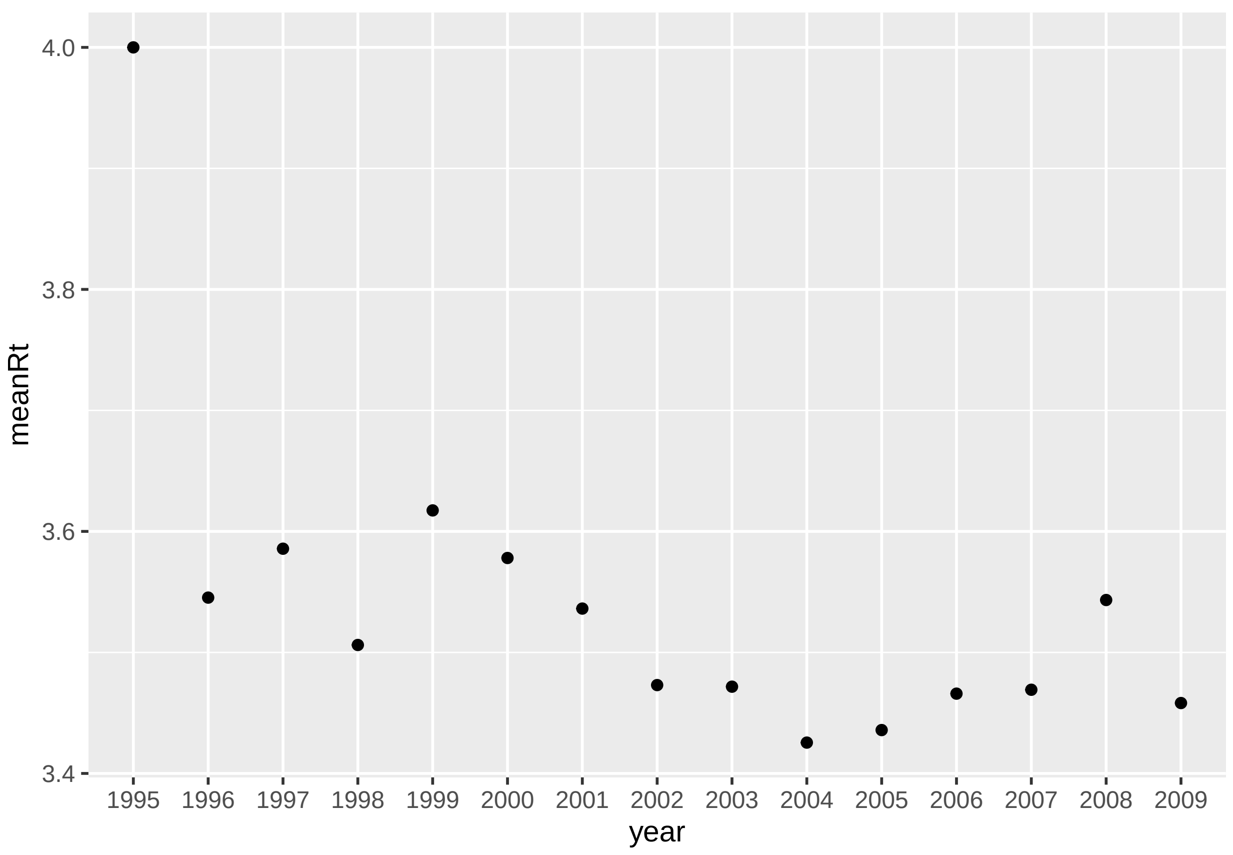


Are there Annual Trends?

Number of ratings per year has leveled out over time. 1995 and 2009 have an exceptionally low number of ratings.

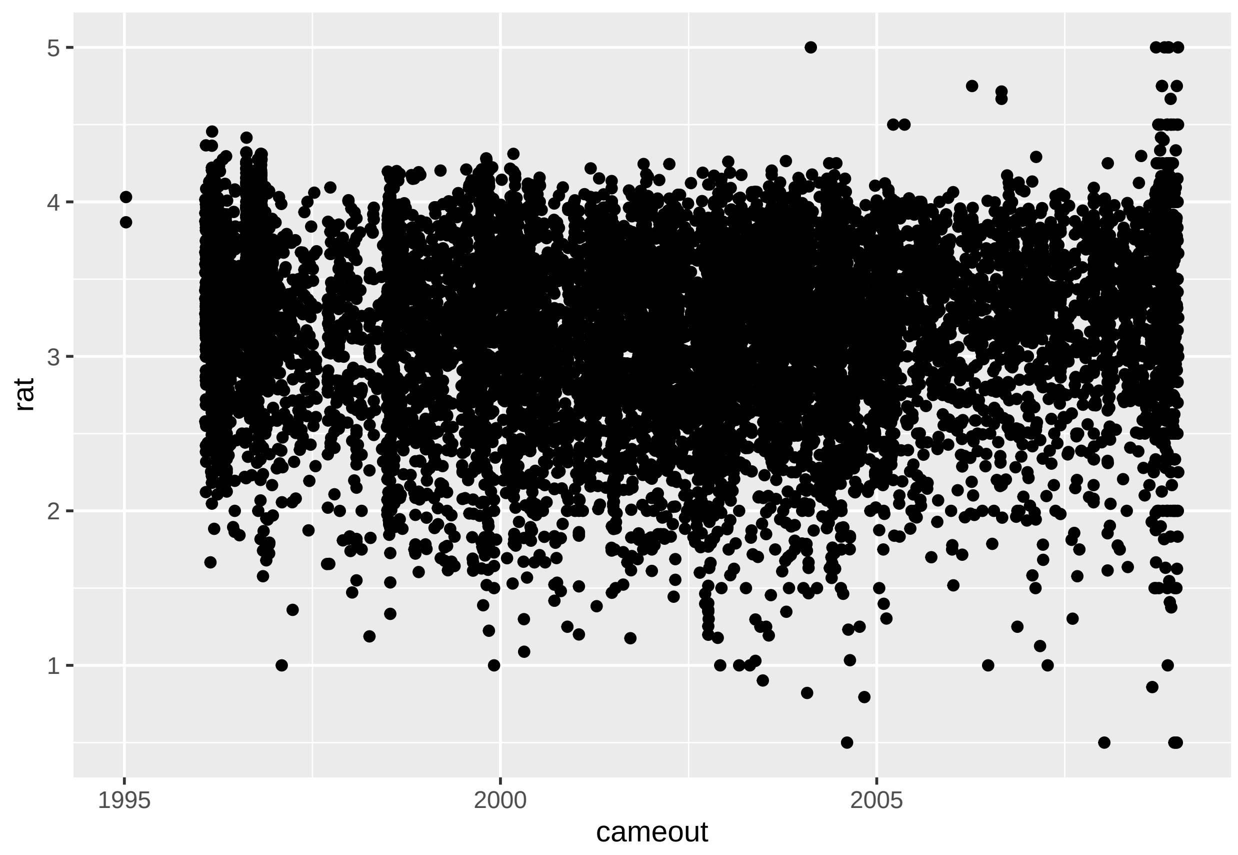


The ratings show trends year over year, as there seem to be two peaks in the data. Particularly 1997, 1999, 2008, and 2000 have higher ratings than other years (1995 is an outlier because only a few movies were rated that year). But the trend is not that strong as the mean stays within 3.4-3.6 stars.



Does when movie was came out effect rating?

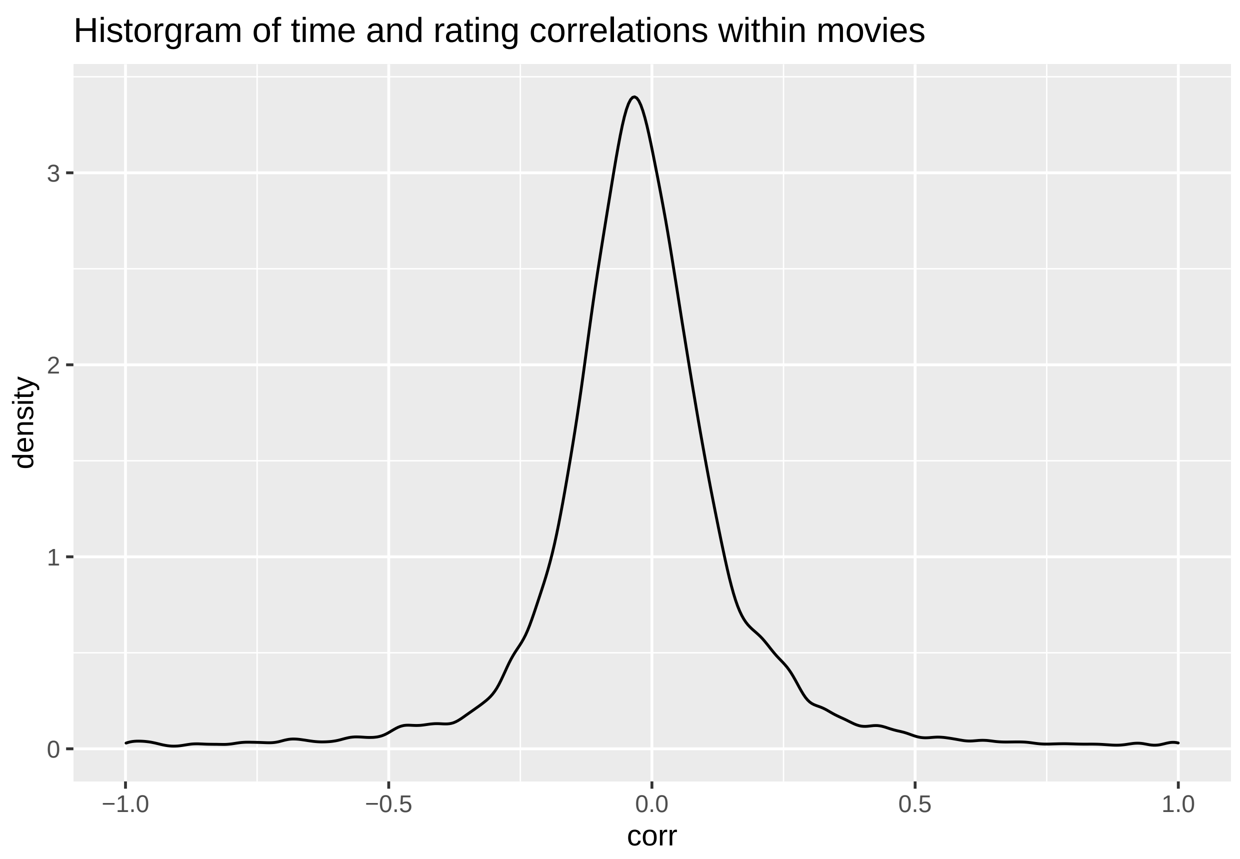
No, when a movie comes out shows no distribution/correlation with ratings.



Do ratings increase with time?

No. I have summarized each movie with correlation between time and rating. Looking at the density the correlations are highly centered around the -0.02, so the ratings show a negligible correlation with when the movies were released.I have only calculated correlations for movies with at least 3 ratings.

```
timeRating <- edx %>% group_by(movieId) %>% filter(n() >=3 ) %>% summarize(corr = cor(timestamp,rating)
timeHist <- timeRating %>% ggplot(aes(corr)) + geom_density()+ggtitle('Historgram of time and rating co
timeHist
```



What is relationship between movie ratings and number of ratings?

Yes, how many ratings a movie gets has a correlation with the number ratings a movie gets, especially with movies over 1000 ratings. The overall correlation is slightly positive, about 0.21.

```
numRating<- edx %>% group_by(movieId) %>% summarize(numRating = n(),
        mrating=mean(rating)) %>% ggplot(aes(mrating,
          numRating )) + geom_point() + geom_smooth()
summ <- edx %>% group_by(movieId) %>% summarize(nratings=n() , mr=mean(rating))
correlationNumRatingAndRatinngs <- cor(summ$nratings, summ$mr)
cat('correlation between number of ratings and movie rating ', correlationNumRatingAndRatinngs)
```

```
## correlation between number of ratings and movie rating  0.2114161
```

Preprocessing the data

I wanted to do this later on but because the last feature to explore is genre which must be extracted first, I will do it now. I also extracted the year feature because we found it useful in one of our previous explorations.

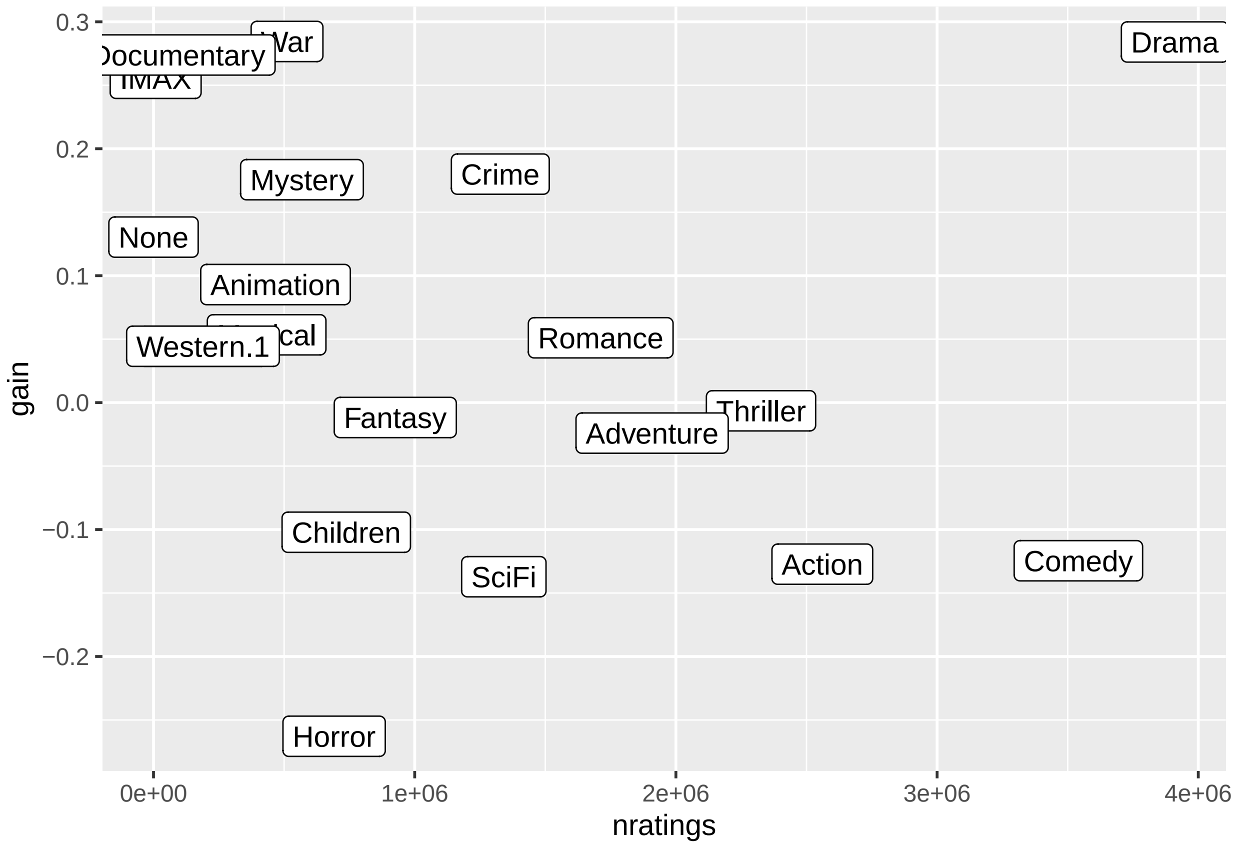
```
preprocessYear <- function(dat) {
  return (dat %>% mutate(year = format(as.Date.POSIXct(timestamp, origin = "1970-01-01"),'%Y') ))
}
edx <- preprocessYear(edx)
validation <- preprocessYear(validation)

preprocess_ExtractGenre<- function(d) {
  d <- d %>% mutate(Mystery = str_detect(genres, 'Mystery') ,
                    Western = str_detect(genres, 'Western') ,
                    Action = str_detect(genres, 'Action') ,
                    Animation = str_detect(genres, 'Animation') ,
                    Fantasy = str_detect(genres, 'Fantasy') ,
                    IMAX = str_detect(genres, 'IMAX') ,
                    War = str_detect(genres, 'War') ,
                    Comedy = str_detect(genres, 'Comedy') ,
                    Romance = str_detect(genres, 'Romance') ,
                    Thriller = str_detect(genres, 'Thriller') ,
                    SciFi = str_detect(genres, 'Sci-Fi') ,
                    Musical = str_detect(genres, 'Musical') ,
                    Western = str_detect(genres, 'Western') ,
                    Crime = str_detect(genres, 'Crime') ,
                    Adventure = str_detect(genres, 'Adventure') ,
                    Documentary = str_detect(genres, 'Documentary') ,
                    Drama = str_detect(genres, 'Drama') ,
                    Horror = str_detect(genres, 'Horror') ,
                    Children = str_detect(genres, 'Children'),
                    FilmNoir = str_detect(genres , "FilmNoir"),
                    None = str_detect(genres , "(no genres listed)")
  )
  return (d)
}
genress <- c('Mystery','Western','Action','Animation','Fantasy',
            'IMAX','War','Comedy','Romance','Thriller','SciFi',
            'Musical','Western','Crime','Adventure','Documentary',
            'Drama', 'Horror', 'Children', 'None', 'FilmNoir')
#I will store the preprocessed input as dedx instead of edx
dedx <- preprocess_ExtractGenre(edx)
validation <- preprocess_ExtractGenre(validation)
rm(edx)
saveRDS(dedx, 'dedx.rds')
saveRDS(validation, 'validation.rds')
```

What is the effect of genre on movie rating?

Surprisingly genre is not an influencial feature. This is because when we adjust for the number of ratings, genres with higher ratings also have more ratings. In other words genre is correlated with the number of ratings that we already explored was a good predictor of ratings. Hence I will not use genre within the model, because the number of ratings a movie gets already contains the effect of genre.

IMAX, War, Documentary, show extreme influence (0.3 stars) Crime, and mystery show slight positive influence (0.2 stars) while comedy action and SciFi show a slight negative influence (-0.1 stars). About 0.2 stars. The others show no difference.

```
##                   gain nratings      genre
## Mystery      0.175626487   568332    Mystery
## Western      0.044386678   189394    Western
## Action      -0.127269034  2560545     Action
## Animation    0.093006157   467168  Animation
## Fantasy     -0.011724851   925637    Fantasy
## IMAX         0.255460447     8181       IMAX
## War          0.284505501   511147        War
## Comedy      -0.124565396  3540930      Comedy
## Romance      0.051061851  1712100     Romance
## Thriller    -0.006458555  2325899    Thriller
## SciFi       -0.137161815  1341183       SciFi
## Musical      0.053409552   433080     Musical
## Western.1    0.044386678   189394   Western.1
## Crime        0.180016691  1327715       Crime
## Adventure   -0.024014787  1908892   Adventure
## Documentary  0.273853598    93066 Documentary
## Drama        0.284090122  3910127       Drama
## Horror      -0.262844929   691485      Horror
## Children    -0.102123767   737994    Children
## None         0.130392043        7        None
## FilmNoir              NA       NA    FilmNoir
```



Results

The metric for the model will be root mean squared error. The strategy I will employ is when given a model to repeatedly

1. Make new predictions.
2. Calculate the residuals of those predictions.
3. Identify a feature or grouping that can decrease the residuals. This feature+old model will be our new model.
4. Repeat step 1.

There are two reasons I take this approach, first because the size of the dataset prevents estimating slopes and other coefficients in general. Second, each user and movie is a categorical variable, and because there are over ten thousand movies and users there is computational difficulty for a model to account for each user/movie. Let's start by loading the data.

```r
rm(list=ls())
library(tidyverse)
library(tidyr)
library(stringr)
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
loadData <- function() {
  dedx <- readRDS('dedx.rds')
  #note that the test set is saved as the validation.rds
  test <- readRDS('validation.rds')
  valIndex <- createDataPartition(dedx$rating,times=1, p=0.2, list = F)
  train_set <- dedx[-valIndex , ]
  validation <- dedx[valIndex , ]
  return (list(train_set,validation, test )  )
}
genress <- c('Mystery','Western','Action','Animation','Fantasy',
             'IMAX','War','Comedy','Romance','Thriller','SciFi',
             'Musical','Western','Crime','Adventure','Documentary',
             'Drama', 'Horror', 'Children', 'None', 'FilmNoir')
data <- loadData()
train_set <- data[[1]]
validation <- data[[2]]
test <- data[[3]]
rm(data)
```

Model 1, mean rating

I was originally going to not include this, but because there are movies and users in the validation/test set that are not in the train set, such movies will not have a rating. Hence the mean serves as a good baseline for such movies.

```r
meanRating <- mean(train_set$rating)
train_set$predictions <- meanRating
train_set$residual <-  train_set$rating - train_set$predictions

valPredictions <- rep(meanRating , dim(validation)[1])
testPredictions <- rep(0 , dim(test)[1]) #I will not use this until
#the end but because I created a single function for all 3 datasets I had to create a
#placeholder
modelTable <- data.frame(name = 'Baseline' ,
                         rmse = RMSE(valPredictions , validation$rating))
```

Helper Functions

Because my approach to building the model is repetitive I have created 2 new functions, one that updates all predictions given a feature to group by and another for the best regularization hyperparameter using cross validation.

```r
addEffect <- function(groupName , oldValPrediction, oldTestPrediction ,
                      train_set, validation, test, regularizer=1.0) {
  #First, calculate the effect
  effectGroup <- train_set %>% group_by_(c(groupName)) %>%
    summarize(eff = mean(residual ) * regularizer)
  #Later in the notebook I will be using a genre feature. Because I want the
  #genre effect to only effect genre of that type, I must set the nonmatching
  #genre effect to zero.
  if (groupName %in% genress) {
    #remove the effect of not in genre
    effectGroup[effectGroup[,groupName]==FALSE, 'eff']=0
  }
  #Second, predict on all three data sets, and update the residuals
  train_set$predictions <- left_join(train_set,effectGroup,
                                     by=groupName)$eff +
    train_set$predictions
  train_set <- train_set %>% mutate(residual = rating - predictions)
  modelPredictionsVal <- predictt(validation,groupName,effectGroup,
                                  oldValPrediction)
  modelPredictionsTest <- predictt(test,groupName,effectGroup,oldTestPrediction)

  #Notice how the returned RMSE is on the validation set rmse not training set rmse
  return (list(valRMSE =RMSE(modelPredictionsVal, validation$rating) ,
               train = train_set,
               valPredictions = modelPredictionsVal,
               testPredictions = modelPredictionsTest))
}
predictt <- function(data, groupName, effectGroup, oldPredictions) {
  effectPredicitons <- left_join(data , effectGroup, by=groupName) %>%
    .$eff
  effectPredicitons[is.na(effectPredicitons)] <- 0 #set the effect for data points
  #missing features to zero
  return (effectPredicitons + oldPredictions)

}
findBestRegularizer <-function(groupName , oldValPrediction, oldTestPrediction ,
                               train_set, validation, test) {
  bestReg <- 1.0
  leastRMSE <- 1.0
  for (regularizer in c(1.0,0.75,0.5,0.3)) {
  result <- addEffect(groupName , oldValPrediction, oldTestPrediction ,
                               train_set, validation, test, regularizer)
  if (leastRMSE > result$valRMSE) {
    leastRMSE <- result$valRMSE
    bestReg <- regularizer
  }
  }
  return (bestReg)
}
```

Model 2, Movie Effect + Model 1

Test predictions are although made at every subsequent model I only calculate the rmse of the predictions at the end. Hence only the training set is updated (updated prediction and residual columns) and validation predictions are updated. I also store a model results table. This procedure will repeat for each model.

```
print(names(train_set))
```

```
##  [1] "userId"      "movieId"     "rating"      "timestamp"   "title"
##  [6] "genres"      "year"        "Mystery"     "Western"     "Action"
## [11] "Animation"   "Fantasy"     "IMAX"        "War"         "Comedy"
## [16] "Romance"     "Thriller"    "SciFi"       "Musical"     "Crime"
## [21] "Adventure"   "Documentary" "Drama"       "Horror"      "Children"
## [26] "FilmNoir"    "None"        "predictions" "residual"
```

```
result <- addEffect('movieId' , valPredictions, testPredictions ,
                    train_set, validation, test)
train_set <- result$train
valPredictions <- result$valPredictions
modelTable <- rbind(modelTable , data.frame(name='Movie Effect',
                            rmse = RMSE(valPredictions , validation$rating)))
modelTable
```

```
##           name      rmse
## 1     Baseline 1.0611602
## 2 Movie Effect 0.9449075
```

Model 3, User Effect + Model 2

Adding the user effect decreases the model substantially.

```
result <- addEffect('userId' , valPredictions, testPredictions ,
                    train_set, validation, test)
train_set <- result$train
valPredictions <- result$valPredictions
modelTable <- rbind(modelTable , data.frame(name='User Effect',
                            rmse = RMSE(valPredictions , validation$rating)))
modelTable
```

```
##           name      rmse
## 1     Baseline 1.0611602
## 2 Movie Effect 0.9449075
## 3  User Effect 0.8673527
```

Model 4, nratings + Model 3

```
train_set <- train_set %>% group_by(movieId) %>% mutate(nratings = n())
validation <- validation %>% group_by(movieId) %>% mutate(nratings = n())
test <- test %>% group_by(movieId) %>% mutate(nratings = n())
reg <- findBestRegularizer('nratings' , valPredictions, testPredictions ,
                           train_set, validation, test) #value is 0.3
result <- addEffect('nratings' , valPredictions, testPredictions ,
                    train_set, validation, test, 0.3)
train_set <- result$train
valPredictions <- result$valPredictions
testPredictions <- result$testPredictions
modelTable <- rbind(modelTable , data.frame(name='Number of Rating Effect',
                         rmse = RMSE(valPredictions , validation$rating)))
modelTable
```

```
##                      name      rmse
## 1                Baseline 1.0611602
## 2            Movie Effect 0.9449075
## 3             User Effect 0.8673527
## 4 Number of Rating Effect 0.8672420
```

Model 5, year + Model 4

```
reg <- findBestRegularizer('None' , valPredictions, testPredictions ,
                    train_set, validation, test) #value is 0.3
result <- addEffect('year' , valPredictions, testPredictions ,
                    train_set, validation, test, 0.3)
train_set <- result$train
valPredictions <- result$valPredictions
testPredictions <- result$testPredictions
modelTable <- rbind(modelTable , data.frame(name='Year Effect',
                         rmse = RMSE(valPredictions , validation$rating)))
modelTable
```

```
##                      name      rmse
## 1                Baseline 1.0611602
## 2            Movie Effect 0.9449075
## 3             User Effect 0.8673527
## 4 Number of Rating Effect 0.8672420
## 5             Year Effect 0.8672367
```

Model 6, genre + Model 5

First I find the residuals within each genre of movies. We discover that the residuals in all of them are negligible except the 'None' or the 'no genre listed' genre that is being predicted incorrectly by almost 0.2 stars.

```
for (i in genress) {
  cat (train_set %>% group_by_(c(i)) %>% summarize(mr=mean(residual)) %>% .$mr , i ,'\n') }
```

```
## -0.0007579117 0.01125008 Mystery
## 0.0001079133 -0.005018098 Western
## 0.003608935 -0.009074048 Action
## 0.0005772177 -0.01054752 Animation
## 0.0004353011 -0.00380031 Fantasy
## 3.672613e-06 -0.004043309 IMAX
## -0.0001111692 0.001844144 War
## 0.001234101 -0.001902519 Comedy
## 0.0006926571 -0.002947827 Romance
## 0.001127667 -0.003237199 Thriller
## 0.00151289 -0.008639842 SciFi
## 0.0003811297 -0.007545106 Musical
## 0.0001079133 -0.005018098 Western
## -0.001147178 0.00662649 Crime
## 0.002977864 -0.01106738 Adventure
## -0.0005205151 0.04987321 Documentary
## -0.00647664 0.008425389 Drama
## -0.0004259985 0.005117822 Horror
## 0.001687029 -0.01890883 Children
## -6.102137e-08 0.07322603 None
## -6.0126e-17 FilmNoir
```

So I add the 'None' genre effect.

```
reg <- findBestRegularizer('None' , valPredictions, testPredictions ,
                    train_set, validation, test) #value is 0.5
result <- addEffect('None' , valPredictions, testPredictions ,
                    train_set, validation, test, 0.5)
train_set <- result$train
valPredictions <- result$valPredictions
testPredictions <- result$testPredictions
modelTable <- rbind(modelTable , data.frame(name='None Genre Effect',
                         rmse = RMSE(valPredictions , validation$rating)))
modelTable
```

```
##                      name      rmse
## 1                Baseline 1.0611602
## 2            Movie Effect 0.9449075
## 3             User Effect 0.8673527
## 4 Number of Rating Effect 0.8672420
## 5             Year Effect 0.8672367
## 6       None Genre Effect 0.8672367
```

Predictions on the test set and performance evaluation

Note that I must retrain the model on the full training set, which means adding the cross validation set back into the training set.

```
#Combine the validation and training set into a new 'fulltrain' dataset, and retrain the model on this
#Also create a prediction column, right now with just the average movie rating.
testPredictions <- rep(meanRating , dim(test)[1])
train_set <- train_set %>% select(-predictions, -residual)
fulltrain <- rbind(train_set, validation)
fulltrain$predictions <- meanRating
fulltrain$residual <-  fulltrain$rating - fulltrain$predictions
rm(train_set)

#Follow the same procedure of adding/calculating effects and predictions
#to the full model as I did for the previous training set model
result <- addEffect('movieId' , valPredictions, testPredictions ,
                    fulltrain, validation, test)
fulltrain <- result$train
testPredictions <- result$testPredictions
result <- addEffect('userId' , valPredictions, testPredictions ,
                    fulltrain, validation, test)
```

```
fulltrain <- result$train
testPredictions <- result$testPredictions
result <- addEffect('nratings' , valPredictions, testPredictions ,
                    fulltrain, validation, test, 0.3)
fulltrain <- result$train
testPredictions <- result$testPredictions
result <- addEffect('year' , valPredictions, testPredictions ,
                    fulltrain, validation, test, 0.3)
fulltrain <- result$train
testPredictions <- result$testPredictions
result <- addEffect('None' , valPredictions, testPredictions ,
                    fulltrain, validation, test, 0.5)
fulltrain <- result$train
testPredictions <- result$testPredictions
cat('Final RMSE = ', RMSE(testPredictions  , test$rating))
```

```
## Final RMSE =  0.8652123
```

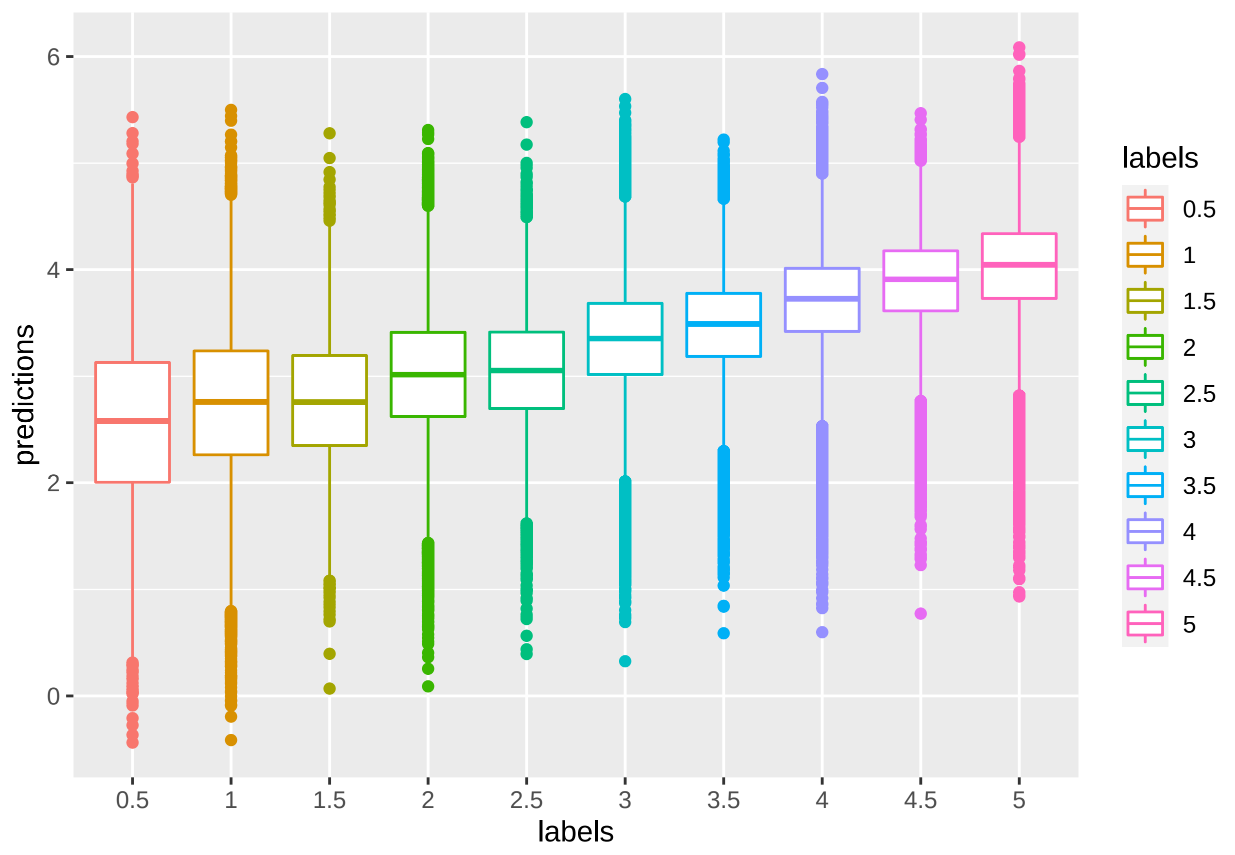
```
modelTable <- rbind(modelTable , data.frame(name='Final Test RMSE',
                         rmse = RMSE(testPredictions , test$rating)))
modelTable
```

```
##                      name      rmse
## 1                Baseline 1.0611602
## 2            Movie Effect 0.9449075
## 3             User Effect 0.8673527
## 4 Number of Rating Effect 0.8672420
## 5             Year Effect 0.8672367
## 6       None Genre Effect 0.8672367
## 7         Final Test RMSE 0.8652123
```

The final test RMSE is 0.8652.

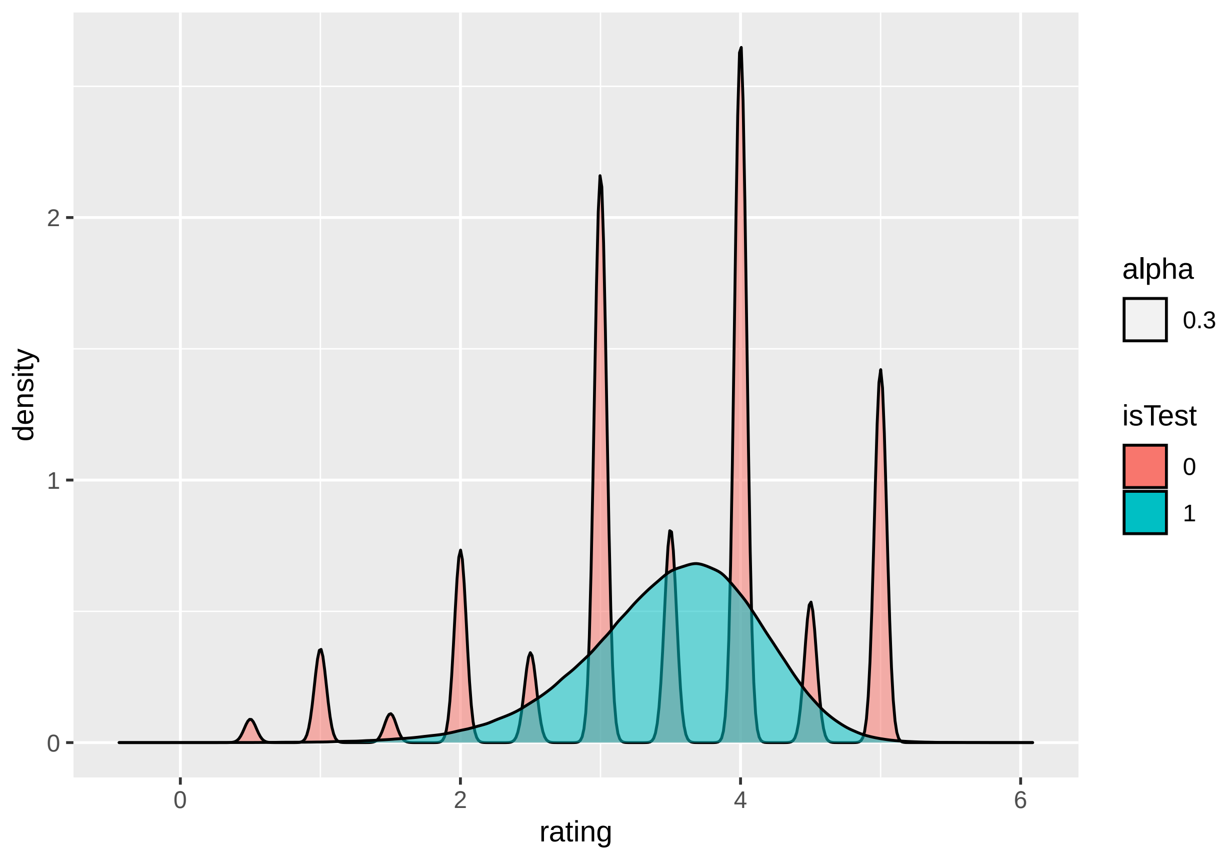
I think that this boxplot of predictions and actual summarizes the model well. While it is true that on average the model is able to differentiate between low ratings and high ratings (the median predicted ratings increases with real ratings). However the model is not able to differentiate between low ratings and high rating movies by a significant margin (actual 5 star ratings starts differ from actual 0.5 star ratings by just 1.5 stars in the model). Furthermore because inter quartile range increases with decreasing movie ratings, the model seems to confuse many low rated movies with high ratings. The model making poor predictions on low star movies is not surprising because there were many more 3-5 star examples then low star examples in the training set.

```
data.frame(predictions = testPredictions , labels=as.factor(test$rating)) %>% ggplot(aes(labels,predict
```



This next density plot of predictions and labels gives a great summary of the model. * The model has not learned a lot. It predicts the average rating of all movies, and is able to differentiate between low ratings and high ratings with limited degree, which is seen in the density plot that is concentrated around the mean. Because the model mainly predicts the mean rating 3.5 stars, ratings far away from the mean, such as low star ratings (less than 1.5 stars) are virtually absent from the model predictions.

```
rbind(data.frame(rating= testPredictions,isTest=1),data.frame(rating=test$rating, isTest=0)) %>% mutate
```



```
cat('difference between 5 stars and 0.5 stars',median(testPredictions[test$rating==5]) - median(testPre
```

```
## difference between 5 stars and 0.5 stars 1.466049
```

```
cat('mean of residual', mean(abs(testPredictions - test$rating)), 'standard deviation of the residual',
```

```
## mean of residual 0.6686018 standard deviation of the residual 0.5491486
```

Conclusion

The goal of this project was to create a model that accurately predicts movie ratings based off of the userId, movieId, timestamp, and genres. The model approach taken was repeated -feature selection and residual reduction. The final model selected userId, movieId, number of ratings, year of timestamp, and the 'no genre listed' genre listed as features for the effect. Regularization was used with parameters determined from cross validation. However the model was unable to predict extreme ratings like 5.0 or 0.5 stars, hence the root mean square error (the metric chosen) was high.

Limitations and Future Work

What the model fundamentally needs is the ability to predict very low and very high rated movies well. The fundamental reason for this problem is not that the training set does not contain enough extreme examples, or that we did not select the right features or the right regularizers, but rather the model is systemetically unable to make extreme predictions. The model is underfitting the data. Because we repeatedly added the effects based on the residual, which had a mean 0.6 stars, each feature could not have changed the predictions significantly. The model was not able to 'break the baseline models predictions', and predictions instead stayed near mean.

A more complex model that can for example, capture that half star ratings more common then full star ratings, is needed. Another thing the model needs to learn is the various clusters that we know exist among audiences and the movies they watch (which is why the user and movie effect decreased the models error the most). We need hard decision boundaries to capture the various groups of users/movies. Furthermore we need a model that performs classification of ratings instead of regression of ratings, as there are only 9 possible ratings. Some models I propose:

- Random Forests, XGBoost: and other tree based models, because they have a hard decision boundary and are good at clustering/separating groups of data.
- Matrix Factorization: a movie-rating matrix contains many empty cells (users do not rate all movies) there will be many empty cells in the matrix, which makes the problem a candidate for matrix factorization.
- Support Vector Machines: a very effective classification algorithm.
- Clustering paired with any classification algorithm: because the move recommendation task contains clusters, we can first use any clustering algorithm (like k-means) to identify the clusters within the data, and then fit a classification model to each of clusters separately.

However all of these would require significantly more computational resources, which is why I did not use them.