MovieLens Report

Estanislao Maria Ferrer

7/30/2020

I. Introduction

According to wikipedia "*The Netflix Prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films, i.e. without the users or the films being identified except by numbers assigned for the contest.*"

Based on the 10M version of the MovieLens dataset, the following project will aim towards a similar objective: to predict movie ratings based on historical data, analyzing the relationship between a single dependent variable (movie rating) and different explanatory variables, with a *root mean square error (RMSE)* below 0.8649 stars.

A. Overview

First, we will perform an exploratory analysis, upon which we will try to understand how different variables behave independently from each other; and, in addition to the standard exploratory analysis, we will study user's specific preferences via Principal Component Analysis (PCA).

In second place, we will build the predictive model, optimize it, and evaluate it.

Lastly, the final model will be tested on the validation dataset, reporting the definitive RMSE.

B. Datasets and minor tweaks

According to edX's conditions, two main datasets will be created: a training set named **edx** used to train and optimize models, and an evaluation set named **validation** which will be executed once and only once at the very end of the project to report the RMSE of the final model.

Furthermore, the edx dataset will be splitted into two: train_set and test_set, upon which the predictive model will be trained and evaluated without using the validation set, thus avoiding over-training.

Finally, some minor tweaks are applied to both training, and test sets:

1. 2 more variables are created upon the original ones: **YYY** (year of rating), and **Release**.
2. Since the variable "genres" may include more than one genre for a single movie, all datasets will be modified in order that there is a unique genre-movie combination.

```
# 1. Movies with no genres listed are removed from training set:
edx <- edx %>% filter(genres != "(no genres listed)")

# 2. We compute indexes to split train and test sets (p = 0.25 was arbitrarily chosen):
set.seed(1000, sample.kind="Rounding")
```

```
## Warning in set.seed(1000, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
indexes <- createDataPartition(y = edx$rating, times = 1, p = 0.25, list = FALSE)
train_set <- edx[-indexes,]
temp <- edx[indexes,]

# 3. We follow edx suggestions and codes in order to make sure
#    that userId and movieId in test set are also in train set:
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

# 4. We include new variables on both train_set and test_set:

train_set <- train_set %>%
mutate(YYY = year(as_datetime(timestamp)), # *
Release = as.numeric(str_extract(str_extract(title,"\\([0-9]{4}\\)$"),"[0-9]{4}"))) %>% # **
separate_rows(genres,sep="\\|") # ***

# * Year of review,
# ** Release date for each movie,
# *** Each movie's gender is splitted into multiple ones,
#     for example "Movie X: Adventure|Comedy" is now "Movie X:
#     Adventure" and "Movie X: Comedy" for each review

test_set <- test_set %>% mutate(YYY = year(as_datetime(timestamp)),
MMM = month(as_datetime(timestamp)),
Release = as.numeric(str_extract(str_extract(title,"\\([0-9]{4}\\)$"),"[0-9]{4}"))) %>%
mutate(timespan = YYY-Release) %>%
separate_rows(genres,sep="\\|")

# 5. Finally, we select desired predictors:
train_set <- train_set %>% select(rating,userId,movieId,genres,YYY,Release)
test_set <- test_set %>% select(rating,userId,movieId,genres,YYY,Release)

rm(temp,removed,indexes)
```

C. Description

Reader will find that dataset description it's quite intuitive: each observation represents a movie rating indicating the user who rated the movie, the movie's Id, the genre, the year in which rating happened, and the year in which movie was released.

Furthermore, ratings oscillates between 0.5 stars and 5 starts with an average of 3.52 stars, most ratings have happened between 1995 and 2009, and the dataset includes movies that has been released between 1915 and 2008.

```
## # A tibble: 6 x 6
##   rating userId movieId genres      YYY Release
```

```
##    <dbl>  <int>   <dbl> <chr>     <int>   <dbl>
## 1      5      1     122 Comedy     1996    1992
## 2      4      2    1210 Adventure  1997    1983
## 3      5      4     587 Drama      1996    1990
## 4      3      9    2797 Drama      2006    1988
## 5      5     56     364 Adventure  2006    1994
## 6      4    249    2355 Adventure  2006    1998
```

It is very important to mention that even though most variables are integers, all of them will be treated as categories or factors. This is because no regression techniques such as linear regression or localized regression will be used, facilitating code execution and avoiding complex solutions.

II. Exploratory analysis

A. Univariate analysis

In the first place, each variable will be studied independently, focusing on predictor's effects on rating. The core idea is to study the so called "item to item variability" within each variable.

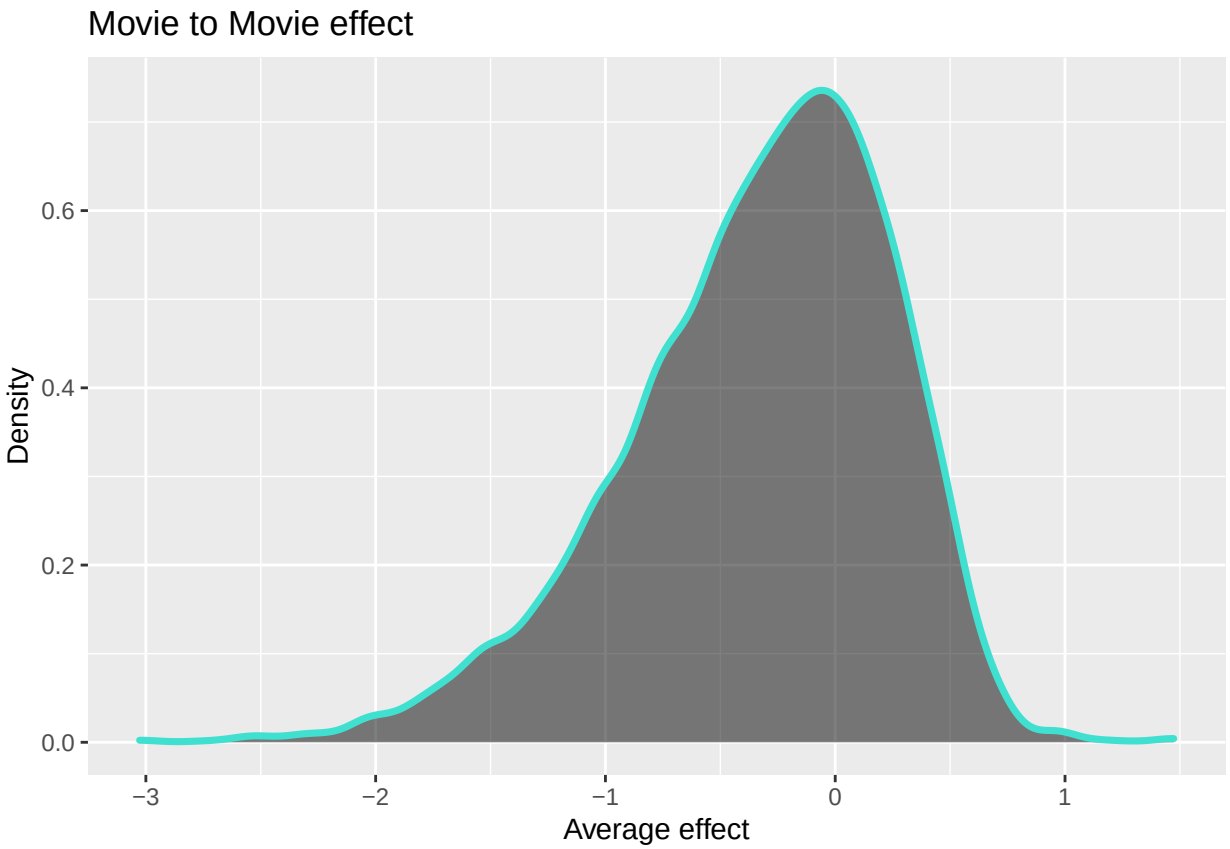
Movie to movie effect

Are there significant differences between a classic like "Pulp Fiction" and a movie like "Besotted"? Let's fin out!

```
Movie_effect <- train_set %>%
  group_by(movieId) %>% # *
  summarize(N = n(), Effect = mean(rating)-alpha) # **

# * Dataset is grouped by each individual item (in this case, movie's ID).
# ** For each item the average rating is computed and the mean rating
#    for the whole dataset (alpha) is removed.
```

Once we have obtained the average effect for each movie, we can study how much this affects our dependent variable. A very intuitive tool we can use is the so called "density plot". The following code will allow for us to visualize the effect of the movie to movie variation:

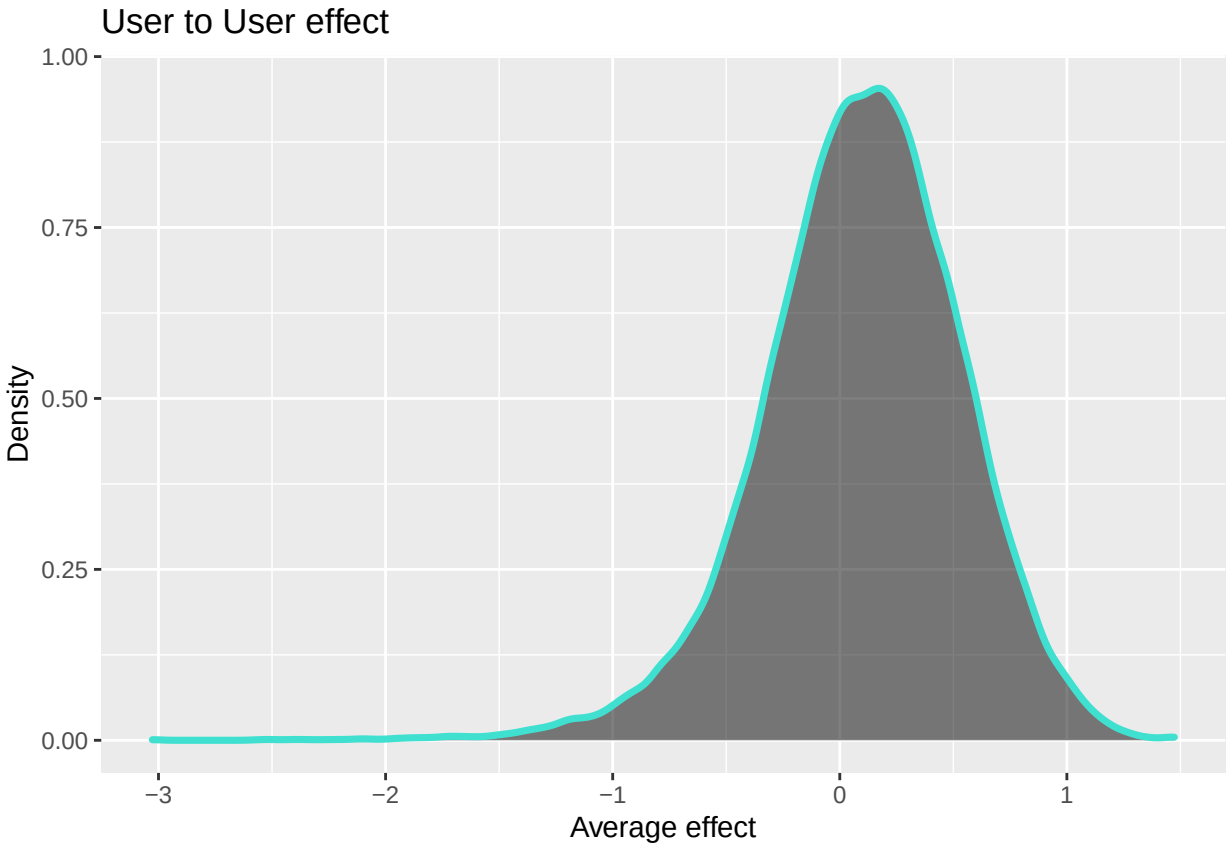
```
Movie_effect %>%
  ggplot(aes(Effect)) +
  geom_density(col="turquoise",fill="black",size=1.25, alpha=0.5) +
  ggtitle("Movie to Movie effect") +
  ylab("Density") +
  xlab("Average effect")
```



The answer to our first question is: **Yes!**. The plot reveals that there are movies that on average are rated 2 or 3 stars **below** the average rating (alpha) and that there are also movies that were rated, on average, 1 star **above**! Reader will also find that the density curve it's quite asymmetric, meaning the proportion of under-rated movies will appear to be quite different to those which are over-rated.

User to user effect

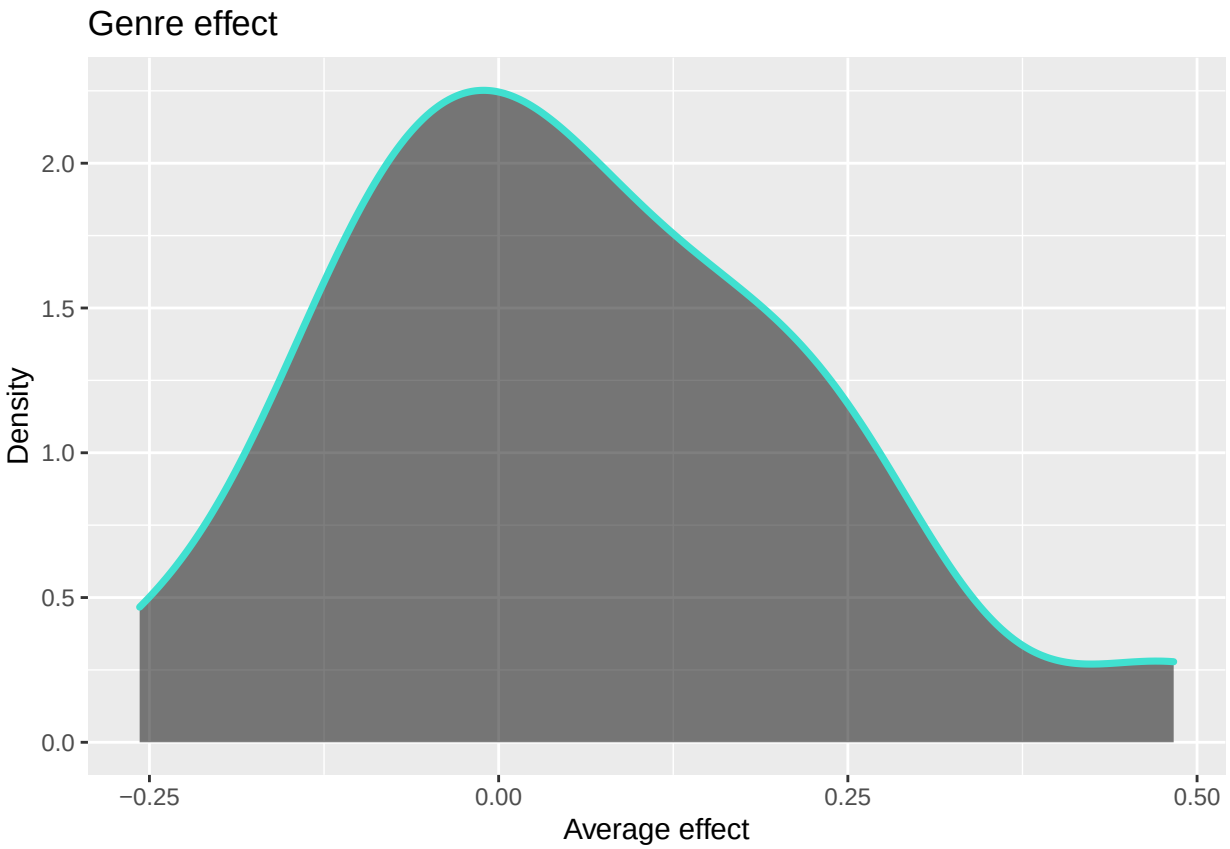
By using a similar approach, we can study the "user to user effect", and make ourselves the exact same question.



We can see that most ratings ranges between -1/1 star below/above average and, unlike the "movie effect", curve appears to follow a symmetric, normal distribution, meaning that the proportion of users that over-rates movies is quite similar to the ones who under-rates.

Genre effect

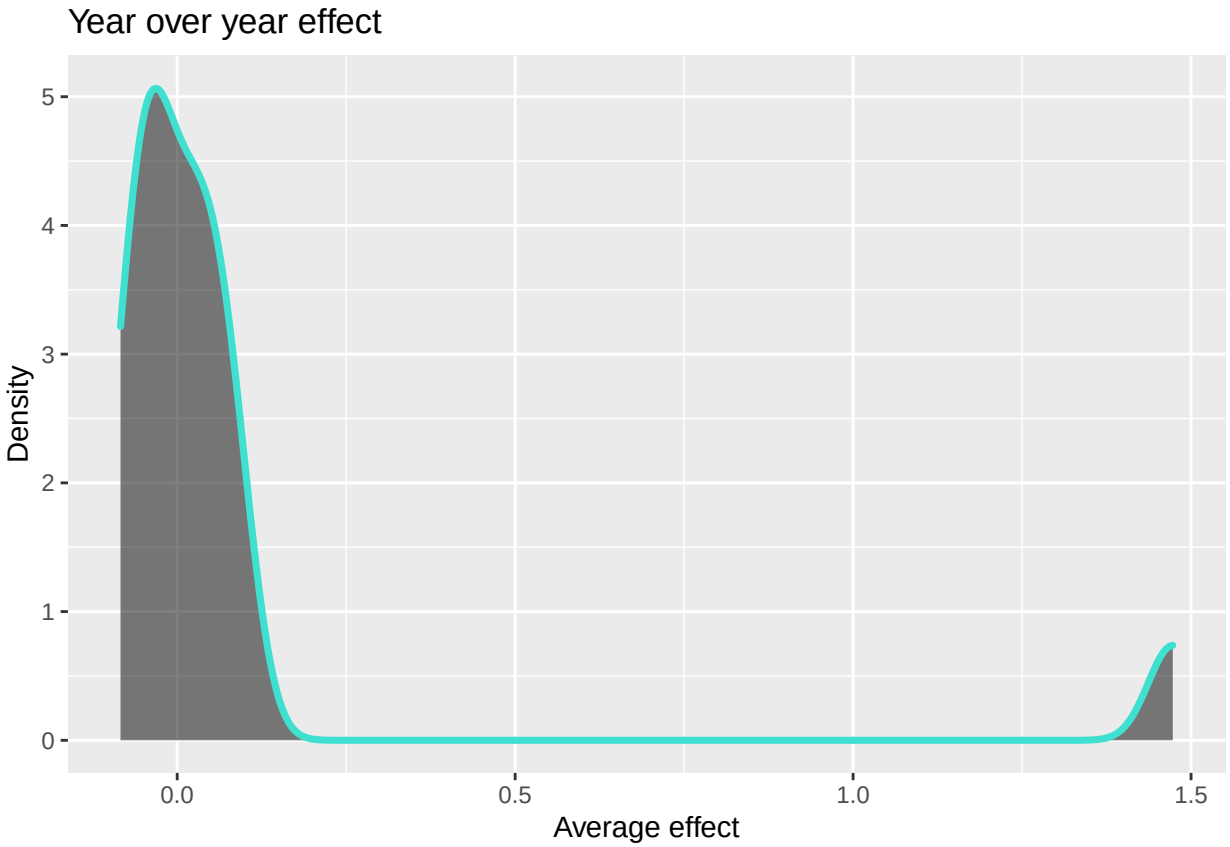
Are adventure movies as popular as romantic ones? Do comedy movies receive better ratings than documentaries? Lets fin out!



It seems that there are no significant differences among genres, since genre effect oscillates between -0.25 and 0.5 starts below/above average. This is not surprising, since there are both "fans" and "haters" for each genre! Nevertheless, this curve does not follow a symmetric distribution, so it might result as an interesting predictor.

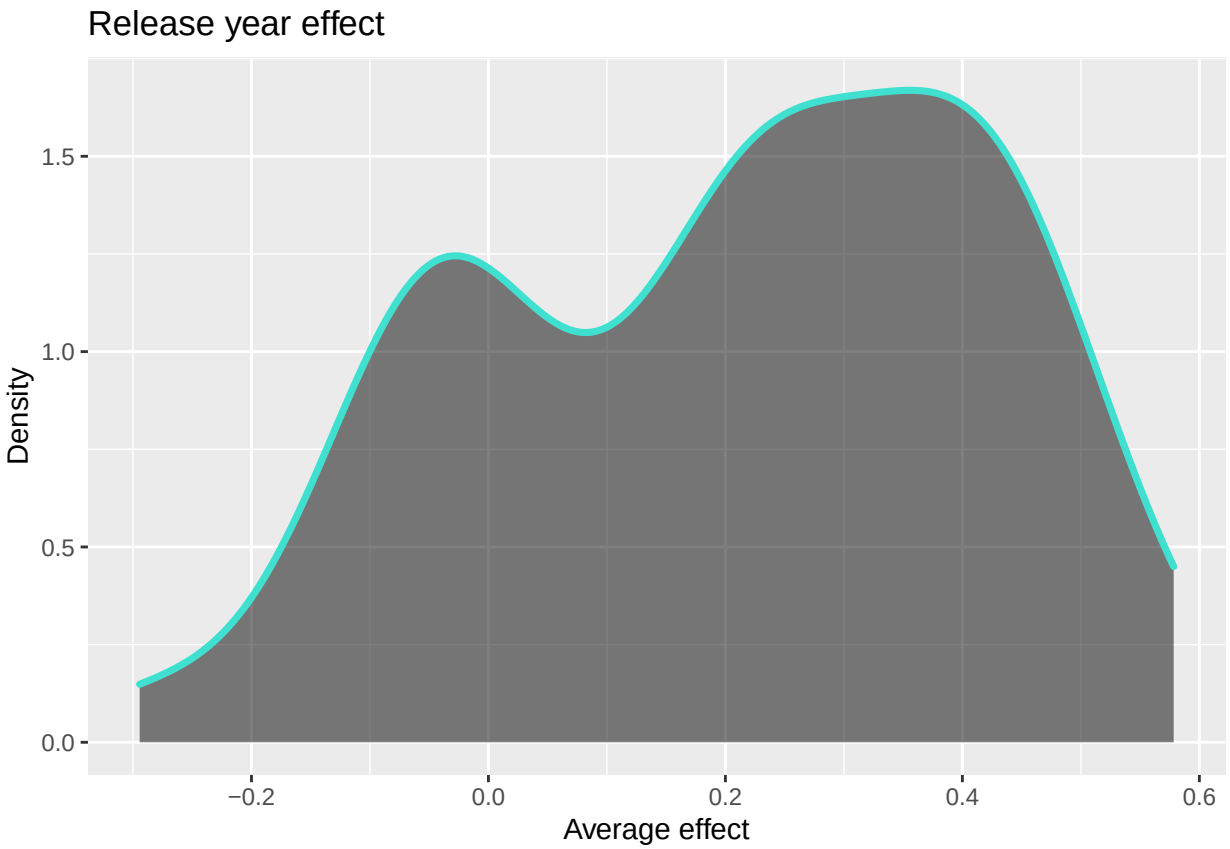
Time effect

Year over Year: Should we expect better ratings during some years and worst ones during others?



Although most data is concentrated on very small values (around 0), there is a small portion of accumulated data towards pretty high values, suggesting that there has been some years in which users have generally over-rated movies.

Year of movie release Is it the same for a movie to be released during 2001/2002 or 2008, when the world was under panic and crisis?



This time, data follows quite an abnormal distribution. What we have here is called a "bi-modal" density curve, and it might provide our predictive model quite useful information, since peaks suggests that most of the data is concentrated both on "below zero" and "above zero" values. It is not "better" or "worst" than any kind of distribution, but bimodal curves represents clusters within observations, and this information will definitely impact in our predictive model.

B. Multivariate analysis: How can we include user's preferences?

Selected predictors may help us achieve our main objective, but they certainly won't help us understand how do people manifests their personal affinities. For instance, we saw that some users are more demanding than others and usually rate movies below average, but we didn't study how do cranky users rate their favorite genres.

One way to do this is to apply clustering algorithms like k-means, but most personal computers don't have the required computing power to do so. Instead, a very popular algorithm called Principal Component Analysis can help us overcome this obstacle: even though it was originally developed to reduce dimensions on large datasets, it can also help us study underlying patterns that simple math may overlook.

By applying PCA on our dataset, we want to understand how do user's preference manifests towards their favorite/most hated genres. Do a Lord of the rings fan reacts the same way towards an adventure movie compared to a romantic one? Let's find out!

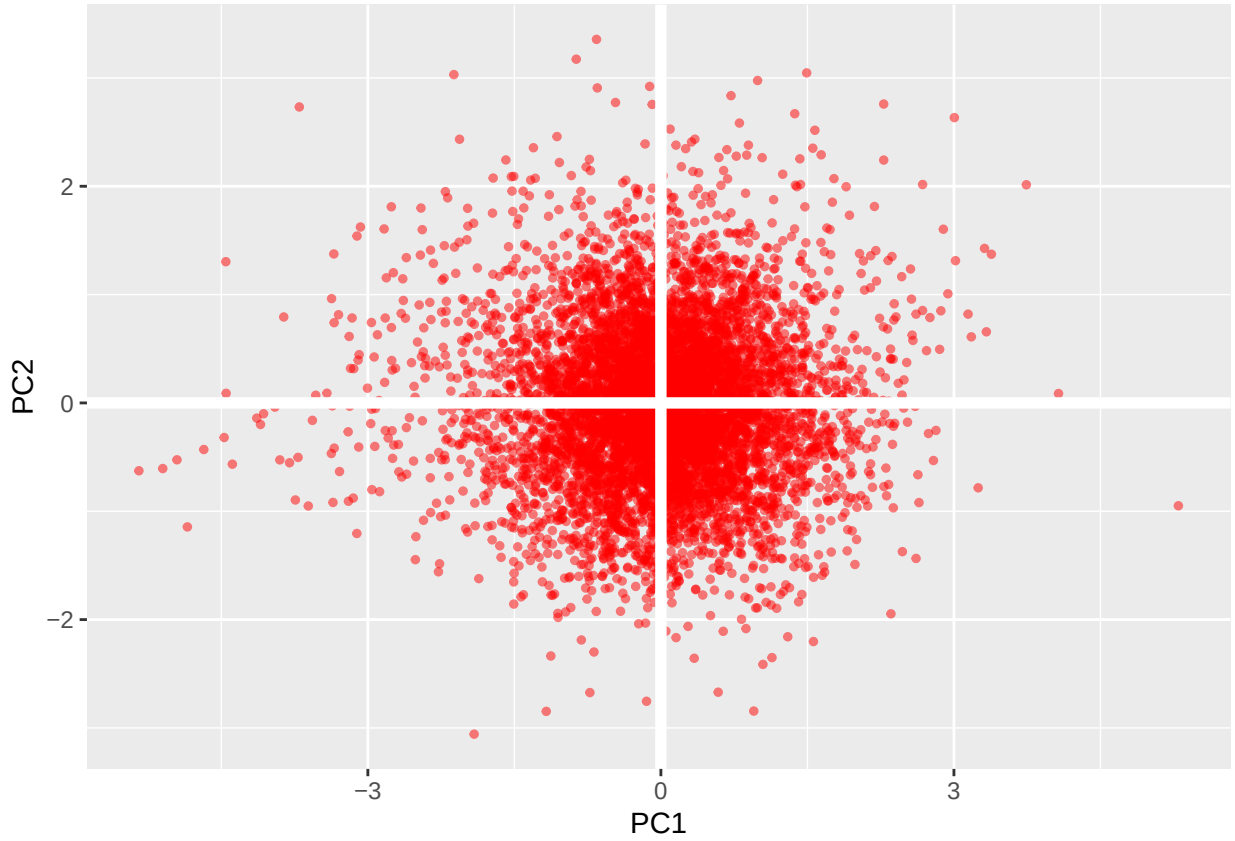
Computing Principal Components:

The following code will guide the reader on the "step by step" of calculating principal components and assigning each user to a different cluster:

```r
# 1. A new dataset is created in which there will be as many rows as different users,
#    and as many columns as different genres.
#    For each user, the average rating for each gender is calculated:
genres_decomp <- train_set %>% group_by(userId,genres) %>% summarize(Effect = mean(rating))
Y <- genres_decomp %>% select(userId,genres,Effect) %>% spread(genres,Effect)

# 2. If a user has never rated a specific genre,
#    we will assume that the movie receives the overall average rating (alpha):
Y[is.na(Y)] <- alpha

# 3. For efficiency purposes a User x Genre matrix is created (UXG),
#    removing userId column and keeping only the different genres.
UXG <- Y[,2:ncol(Y)] %>% as.matrix()
UXG <- sweep(UXG, 1, rowMeans(UXG, na.rm=TRUE))
UXG <- sweep(UXG, 2, colMeans(UXG, na.rm=TRUE))
UXG <- as.data.frame(UXG)

# 4. We perform PCA on our User x Genre matrix,
#    and we store each principal component in a data.frame called "X".
#    For each variable, there is one principal component (PC),
#    and values for each PC oscillates between a negative real number and a positive real number.
#    Being the first PC the one that most represents data structure, it is the most important one.
#    The same might be said about the second one, the third, and so on.
PCA <- prcomp(UXG)
X <- as.data.frame(PCA$x)
```

The reader might be asking him/herself, "how do we use this information?". We will use it to create clusters!

Before we proceed, if we plot two PC's reader will understand that each value oscillates between minus infinity and infinity:



Now that we know how is data distributed, we can make sure users with positive values on one PC will be assigned a category (X) and those with negative values will be assigned another (Y).

For efficiency purposes we will only keep the first 2 principal components, so there are only 4 different groups a user can be assigned to:

1. "X X"
2. "X Y"
3. "Y X"
4. "Y Y"

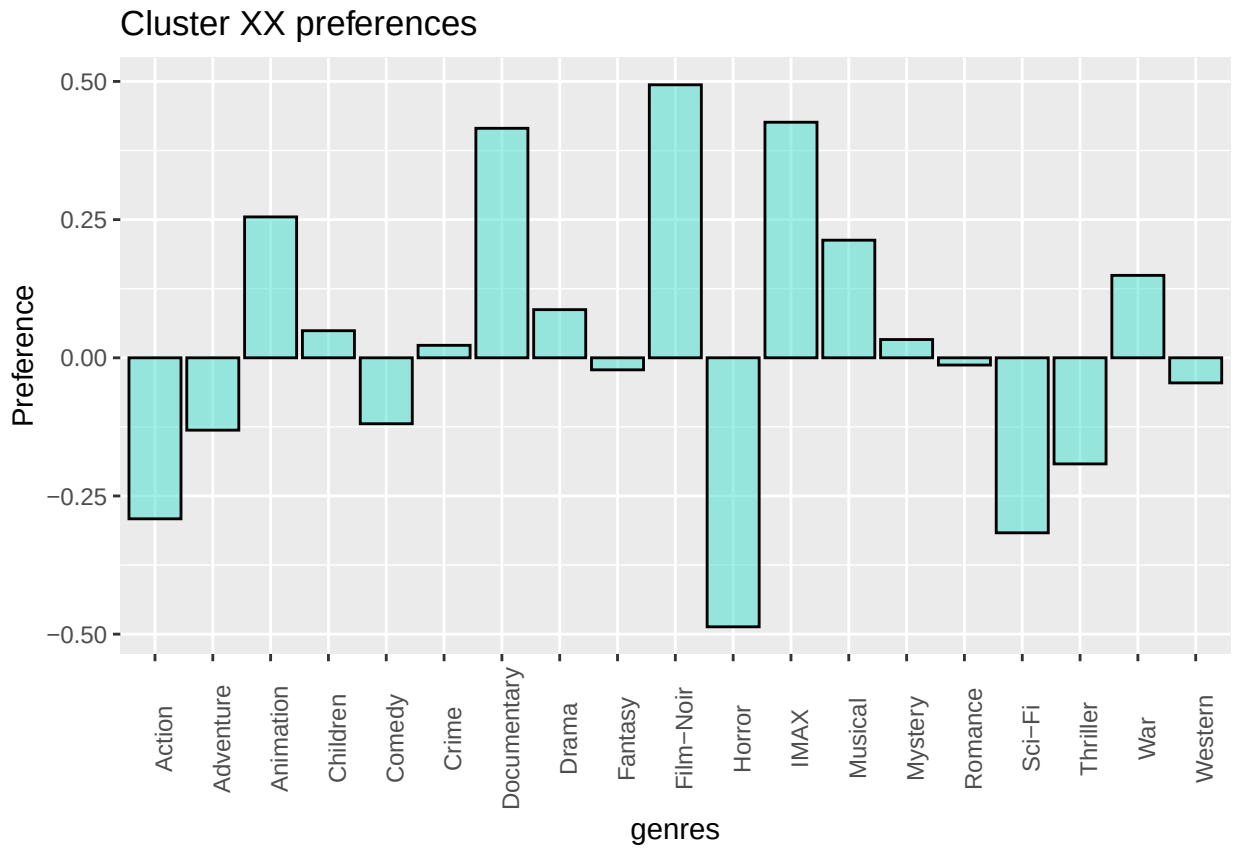
Once each user is assigned to a specific cluster we proceed to modify our training set adding two new variables: **userType** indicating the cluster to which each user belongs to, and **event**, indicating what happened on each movie rating (for example, an "X X" user rated an Adventure movie, or and "X Y" user rated a Comedy movie).

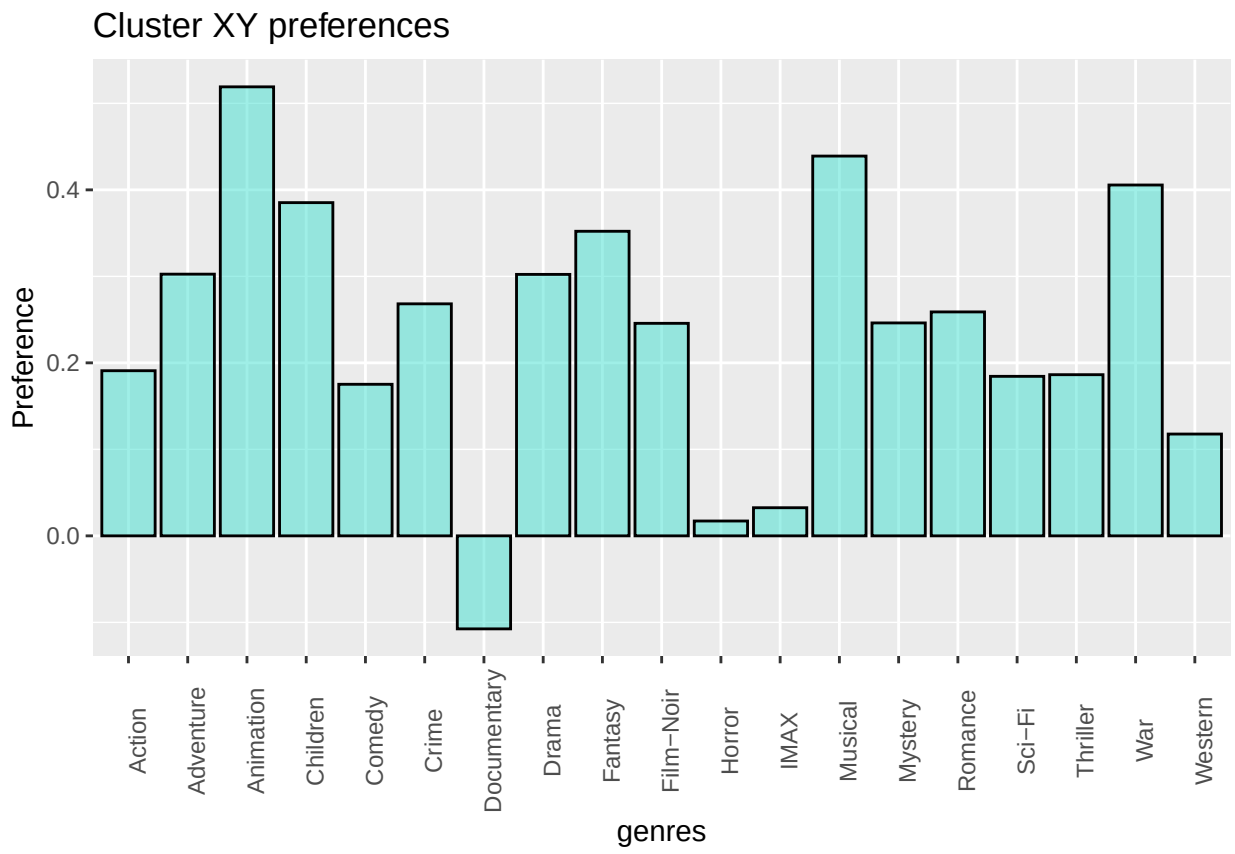
```
# 1. Creating categories:
Categories <- data.frame(userId = Y$userId,
                         Cat1 = ifelse(X$PC1>0,"X","Y"),
                         Cat2 = ifelse(X$PC2>0,"X","Y"))

Categories <- Categories %>% mutate(userType = paste(Cat1,Cat2)) %>% select(userId,userType)

# 2. Including userType variable and creating "event":
train_set <- train_set %>%
  left_join(Categories, by = "userId") %>%
```

```
  mutate(Event = paste(genres,userType))

test_set <- test_set %>%
  left_join(Categories, by = "userId") %>%
  mutate(Event = paste(genres,userType))
```

What is PCA telling us?

So far we have calculated all principal components on a User x Genre data.frame, we have also assigned each user to a specific cluster, and we have created two more variables on our train_set and test_set. We must, off course, explore how do different clusters behave:





We can see, for instance, that "X X" cluster is mainly composed by cranky users who enjoys Film-noir movies and Documentaries, but clearly hates Horror movies. On the other hand, "X Y" users usually rates above average except for documentaries, but definitely have some genre preferences, like Animation and Musicals.

III. Methodology: Building our predictive model

I would like to appreciate the effort the reader has made in order to get to this point. Overall we have studied each predictor individually, chosen the most important ones, and we have created 4 different user clusters that manifests their preferences quite differently.

Now we will build our predictive model, and optimize it by applying regularization. Once we are done with that, we will evaluate it on the test_set, and check if it can help us achieve our mail goal: predict movie ratings with a root mean squared error below 0.8649. Finally, we will use the optimal model to train it on the whole "edx" dataset.

A. First stage: No regularization

We will first create a model with no regularization at all:

```
# The training set is grouped by a specific item (in this case, movieId),
# and for each movie the average rating is calculated.
# The overall average rating (alpha) is subtracted from the result.
b_movie <- train_set %>% group_by(movieId) %>% summarize(b_movie = mean(rating-alpha))
```

```
# We create a new dataset called "Effects" which will just include the desired predictors,
# and each variable's effect:
Effects <- train_set %>%
  left_join(b_movie, by = "movieId") %>%
  select(rating,YYY,Release,Event,genres,userId,b_movie)

# User's effect:
b_user <- Effects %>% group_by(userId) %>% summarize(b_user = mean((rating-alpha)-b_movie))
Effects <- Effects %>% left_join(b_user, by = "userId")

# Year over year effect:
b_yoy <- Effects %>% group_by(YYY) %>% summarize(b_yoy = mean((rating-alpha)-b_movie-b_user))
Effects <- Effects %>% left_join(b_yoy, by = "YYY")

# Release year effect:
b_year <- Effects %>% group_by(Release) %>%
  summarize(b_year = mean((rating-alpha)-b_movie-b_user-b_yoy))
Effects <- Effects %>% left_join(b_year, by = "Release")

# Event effect (userType - genre combination):
b_event <- Effects %>% group_by(Event) %>%
  summarize(b_event = mean((rating-alpha)-b_movie-b_user-b_yoy-b_year))
Effects <- Effects %>% left_join(b_event, by = "Event")

# Genre effect:
b_genre <- Effects %>% group_by(genres) %>%
  summarize(b_genre = mean((rating-alpha)-b_movie-b_user-b_yoy-b_year-b_event))
Effects <- Effects %>% left_join(b_genre, by = "genres")

# By adding different effects, we calculate the predicted rating,
# and we keep the observed rating (rating) and the predicted one (y_hat)
Effects <- Effects %>% select(rating,b_movie,b_user,b_yoy,b_year,b_event,b_genre) %>%
  mutate(y_hat = alpha + (b_movie+b_event+b_genre+b_user+b_yoy+b_year)) %>%
  select(rating,y_hat)
```

Lastly, the RMSE is calculated on the training set:

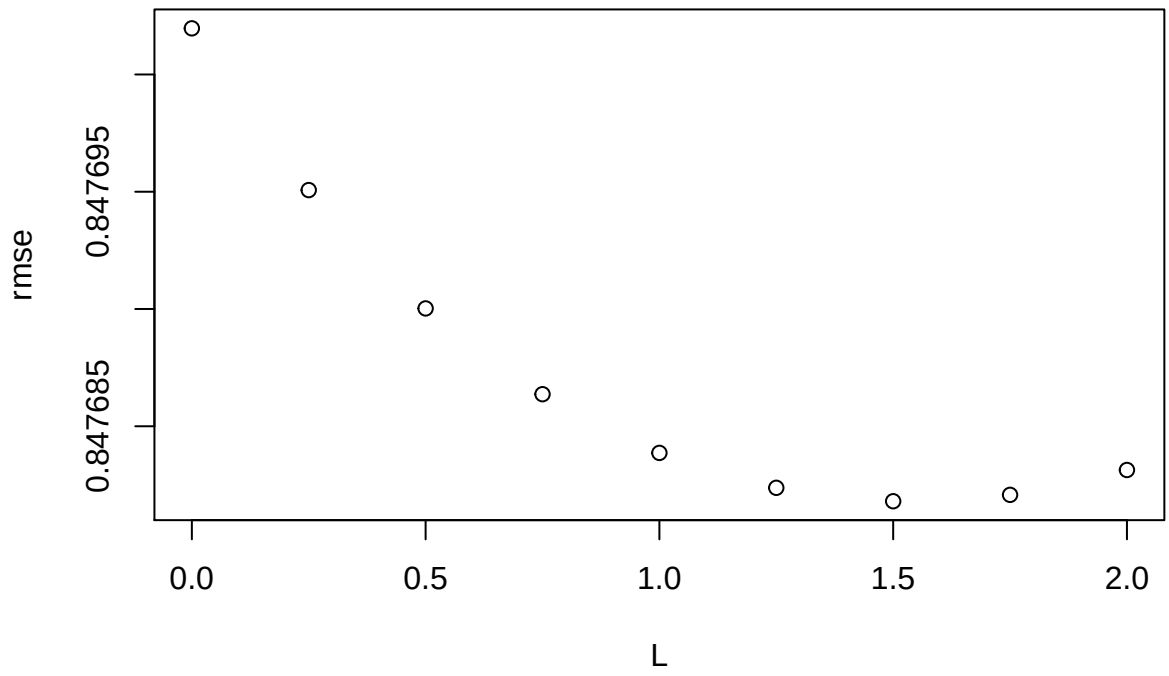
```
RMSE(Effects$y_hat,Effects$rating)
```

```
## [1] 0.847702
```

B. Second stage: Regularization

The RMSE we have obtained it's quite decent, but we can definitely improve it!

In order to do so, we will use the "lambda technique": When calculating the average effect of a particular variable, we will add an extra double to the denominator that will penalize large estimates resulting from small grouping samples. Lambda will assume decimal values from 0 to 2, and we will select the one that minimizes the RMSE value.



Now we have a specific value upon which we can re-train our model.

C. Third stage: Evaluating on test set

Now it is time to determine whether the model is finished or not, but before we execute any evaluation, we must tweak our datasets:

```
# 1. First we add recalculated effects to our testing set
#    (each "b_effect" has been recalculated with optimal lambda)
beta_test <- test_set %>%
  left_join(b_movie, by = "movieId") %>%
  left_join(b_genre, by = "genres") %>%
  left_join(b_event, by = "Event") %>%
  left_join(b_user, by = "userId") %>%
  left_join(b_yoy, by = "YYY") %>%
  left_join(b_year, by = "Release") %>%
  select(rating,b_movie,b_genre,b_event,b_user,b_yoy,b_year)

# 2. In case there are events or rating years that are not considered by the train_set,
#    we fill the NA values with the average effect of each predictor:
beta_test$b_yoy[which(is.na(beta_test$b_yoy)=="TRUE")] <- mean(b_yoy$b_yoy)
beta_test$b_event[which(is.na(beta_test$b_event)=="TRUE")] <- mean(b_event$b_event)

# 3. Predicted ratings are calculated:
beta_test <- beta_test %>%
```

```
  mutate(y_hat = alpha+b_movie+b_genre+b_event+b_user+b_yoy+b_year) %>%
  select(rating,y_hat)

# 4. RMSE:
RMSE(beta_test$rating,beta_test$y_hat)
```

```
## [1] 0.8637402
```

IV. Results

We have seen that with regularization, RMSE on training set can be as low as **0.847702** and, when predicting with our testing dataset, result can also be quite satisfactory.

Now the final evaluation will be performed. We will first train our model on the entire edx dataset and with the lambda value that minimizes error, and add new variables to validation dataset.

```
# 1. Binding train_set and test_set (obtaining edx dataset but with new variables):
final_train_set <- rbind(train_set,test_set)
rm(train_set,test_set)

# 2. Including new variables into validation dataset
validation <- validation %>%
mutate(YYY = year(as_datetime(timestamp)),
Release = as.numeric(str_extract(str_extract(title,"\\([0-9]{4}\\)$"),"[0-9]{4}"))) %>%
separate_rows(genres,sep="\\|") %>%
left_join(Categories, by = "userId") %>%
mutate(Event = paste(genres,userType)) %>%
select(rating,movieId,YYY,Release,userId,Event,genres)

rm(Categories)
gc()

# 3. Training model with L value (representing the optimal lambda):
  b_movie <- final_train_set %>%
    group_by(movieId) %>% summarize(b_movie = sum(rating-alpha)/(n()+L))

  Effects <- final_train_set %>%
    left_join(b_movie, by = "movieId") %>%
    select(rating,YYY,Release,Event,genres,userId,b_movie)

  b_user <- Effects %>% group_by(userId) %>%
    summarize(b_user = sum((rating-alpha)-b_movie)/(n()+L))

  Effects <- Effects %>% left_join(b_user, by = "userId")

  b_yoy <- Effects %>% group_by(YYY) %>%
    summarize(b_yoy = sum((rating-alpha)-b_movie-b_user)/(n()+L))

  Effects <- Effects %>% left_join(b_yoy, by = "YYY")

  b_year <- Effects %>% group_by(Release) %>%
    summarize(b_year = sum((rating-alpha)-b_movie-b_user-b_yoy)/(n()+L))
```

```
  Effects <- Effects %>% left_join(b_year, by = "Release")

  b_event <- Effects %>% group_by(Event) %>%
    summarize(b_event = sum((rating-alpha)-b_movie-b_user-b_yoy-b_year)/(n()+L))

  Effects <- Effects %>% left_join(b_event, by = "Event")

  b_genre <- Effects %>% group_by(genres) %>%
    summarize(b_genre = sum((rating-alpha)-b_movie-b_user-b_yoy-b_year-b_event)/(n()+L))

rm(Effects,final_train_set)

# 4. Adding new variables to validation dataset:
validation <- validation %>%
  left_join(b_movie, by = "movieId") %>%
  left_join(b_genre, by = "genres") %>%
  left_join(b_event, by = "Event") %>%
  left_join(b_user, by = "userId") %>%
  left_join(b_yoy, by = "YYY") %>%
  left_join(b_year, by = "Release") %>%
  select(rating,b_movie,b_genre,b_event,b_user,b_yoy,b_year)

# 5. Replacing NA's originated on non-contemplated values on final_train_set
#    with predictor's average effect:
validation$b_yoy[which(is.na(validation$b_yoy)=="TRUE")] <- mean(b_yoy$b_yoy)
validation$b_event[which(is.na(validation$b_event)=="TRUE")] <- mean(b_event$b_event)

# 6. We calculate predictions and select both observed and predicted ratings:
validation <- validation %>%
  mutate(y_hat = alpha+b_movie+b_genre+b_event+b_user+b_yoy+b_year) %>%
  select(rating,y_hat)
```

Once we are done, we calculate the RMSE value on validation dataset, concluding with our project:

```
RMSE(validation$rating,validation$y_hat)
```

```
## [1] 0.861462
```

V. Final thoughts

A. Conclusions

Reader have seen that variables behave quite differently, and this represents a very useful source of information that help us predict with better accuracy. After analyzing each predictor individually, and clustering users within different groups with different preferences, we have managed to build a predictive model that help us achieve satisfactory results.

First, we created a model from scratch and evaluated it on our training set. After doing so, we applied the "lambda" technique in order to achieve a better performance and managed to pull satisfactory results when assessing model's performance on the test_set. Finally, predictions were made on the validation set, achieveing our main goal with a RMSE of **0.8615**

B. Limitations

The main limitation has undoubtedly been computer power, since it has forced me to use only a few variables and prevented from using more complex algorithms. It has also forced me to use only 2 principal components for clustering observations, while using 3 or 4 PCs could have provided much more accuracy.

Another big limitation is that in order to follow an innovative approach I only used the user-genre combination, while there may be others with much more predictive power. Once again, it has been impossible to execute PCA on other combinations since there are approximately just 20 different genders while one might find hundreds of movies, for example.

C. Future work

Future work will involve performing much more complex machine learning algorithms in order to achieve a lower error. I also encourage the reader to try this model by using more principal components so the algorithm can capture much more variability when explaining user's preference towards different genres. Finally, clustering observations with other variables, such as a user-movie or year-genre combination may also help with additional accuracy.

Thank you, Estanislao Maria Ferrer.