Reproduction and evaluation of the paper: Evaluating the dynamic properties of recommendation algorithms.

Group 6: Marc Bodner, Raphael Gruber, Hannes Ladurner and Paul Znidaric

2023-04-18

Abstract

The paper "Evaluation of the Dynamic Properties of Recommendation Algorithms" underlines the importance of using dynamic properties for the evaluation of recommender systems. In this report we attempt to reproduce the results presented, and evaluate these results. We use the profile mean squared error and the profile absolute error to measure the accuracy of predictions and how they change if the dataset increases in size and new information arrives. This methodology is used on the UBCF the IBCF and the Naive Bayes. According to our results the number of new entries and the larger the dataset, the more accurate our predictions turns out.

1. Introduction

Recommendation algorithms have become essential in our day-to-day live on e-commerce websites. With the increasing advancements in machine learning techniques and algorithms, evaluating recommendation algorithms have become a complex task. With that in mind, the paper "Evaluating the Dynamic Properties of Recommendation Algorithms" from Robin Burke has provided a new evaluation method for dynamic aspects of said algorithms. It is referred to as the "temporal leave-one-out" approach which shows robustness towards attacks but suffer a high accuracy cost.
In this report we attempt to execute the methodology for our three algorithms (UBCF, IBCF, NB) and compare our results to that of the paper while trying to showcase the similarities and differences between our results and those of the paper.

2. Critical Evaluation of the paper

The paper describes how recommender systems are often evaluated on static data. The authors believe that this evaluation does not suffice, due to information constantly changing and the recommender systems with it. Instead, they emphasize on the importance of dynamic properties, which should consider this information flow. In order to evaluate these dynamic properties, the authors propose using the "leave-one-out" method to assess the performance of the algorithm. This method consists of deleting a dataset entry and then predicting that specific entry by using the algorithm and comparing the results. By using both the mean absolute error and the mean squared error one can evaluate the accuracy of predictions. Despite the paper being very insightful and easy to follow, it seems to lack practical applications and fails to discuss the importance of their findings in the field of e-commerce and content recommendation platforms. Additionally, the paper lacks comparisons of its process or outcomes with other studies. This makes it a little difficult for us to contextualize the results. Furthermore, the authors only briefly mention some limitations, without attempting to describe any sources of bias or alternative explanations of results. In conclusion, the paper convinced us, that the evaluation of recommendation systems should include their dynamic properties and not just their accuracy at a single point in time.

3. Implementation

We implemented 4 algorithms for this project: the ProfileMAE from the paper we aim to recreate and an UBCF, an IBCF and Naive Bayes implementation. To implement the ProfileMAE algorithm outlined in the paper for our purposes we needed to make some adjustments to the pseudo code. Most importantly we had to handle edge cases for the prediction algorithms, but we also enabled the algorithm to switch between prediction algorithms (UBCF, IBCF, NB) and error type (MAE, MSE). Hence we called the algorithm ProfileError. The ProfileError algorithm evaluates prediction algorithms by making a prediction based on all the ratings that previously entered the dataset and calculating the resulting error term. These error terms are then aggregated per profile. We identified three edge cases: 1. The first rating of a user u: This is problematic as this means that in the sub-dataset on which the prediction is based there are no ratings of the user for whom the prediction should be made. 2. The first rating of an item i: This is problematic as this means that in the sub-dataset on which the prediction is based there are no ratings of the item for which the prediction should be made. 3. The timestamp of the jth rating of user u is equal to the timestamp of the 1st rating: This is problematic for the same reason as 1. In all these edge cases, we decided to use the average rating of the item in question as the prediction. If this turns out to be NA, we predict a rating of 3. As for the other algorithms we chose to implement versions of UBCF, IBCF and Naive Bayes outlined in Unit 1: Application for Data Science by Univ. Prof. Dr. Dorner et al.

4. Data

4.1 Dataset

The Paper used the 1M MovieLense Data set which contains 1 million ratings from 6040 Users rating 3883 movies. They furthermore truncated the dataset to delete all entries after the 12/31/2000 which took about 6% of the entries away. For our approach, we used the MovieLense – 100k Dataset, it contains rating data collected through the MovieLense website (movielens.umn.edu) during a seven month period from Steptember 19th, 1997 to April 22nd, 1998. It Contains 100.000 Ratings (1-5 Stars) from 943 users rating 1664 Movies.

As mentioned above, the paper used the MovieLense – 1M with 1 million ratings, which would absolutely out scale the processing abilities of our systems. But we wanted to make sure, that our findings are comparable with the ones in the paper, so the smaller Version MovieLense – 100k was chosen for this use. This dataset was retrived from https://grouplens.org/datasets/movielens/100k/ (F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context).

```
##    V1  V2 V3        V4
## 1 196 242  3 881250949
## 2 186 302  3 891717742
## 3  22 377  1 878887116
## 4 244  51  2 880606923
## 5 166 346  1 886397596
```

4.2 Data preprocessing

To make the Data more appealing and useful to us, we first renamed the Columns to: “user_id”, “item_id”, “rating”, “timestamp”. To make the “timestamp” variable workable, we used the POSIXct Format to make it understandable for humans. To handle outliers, we truncated the dataset, by deleting users with <15 movies rated and users with >650 movies rated. Resulting in 98.578 ratings. Resulting in the: “ml_data_trunc” – Data set which we than used to apply our algorithms. You will find more detailed steps at the Bottom of the paper, at the beginning of the code.

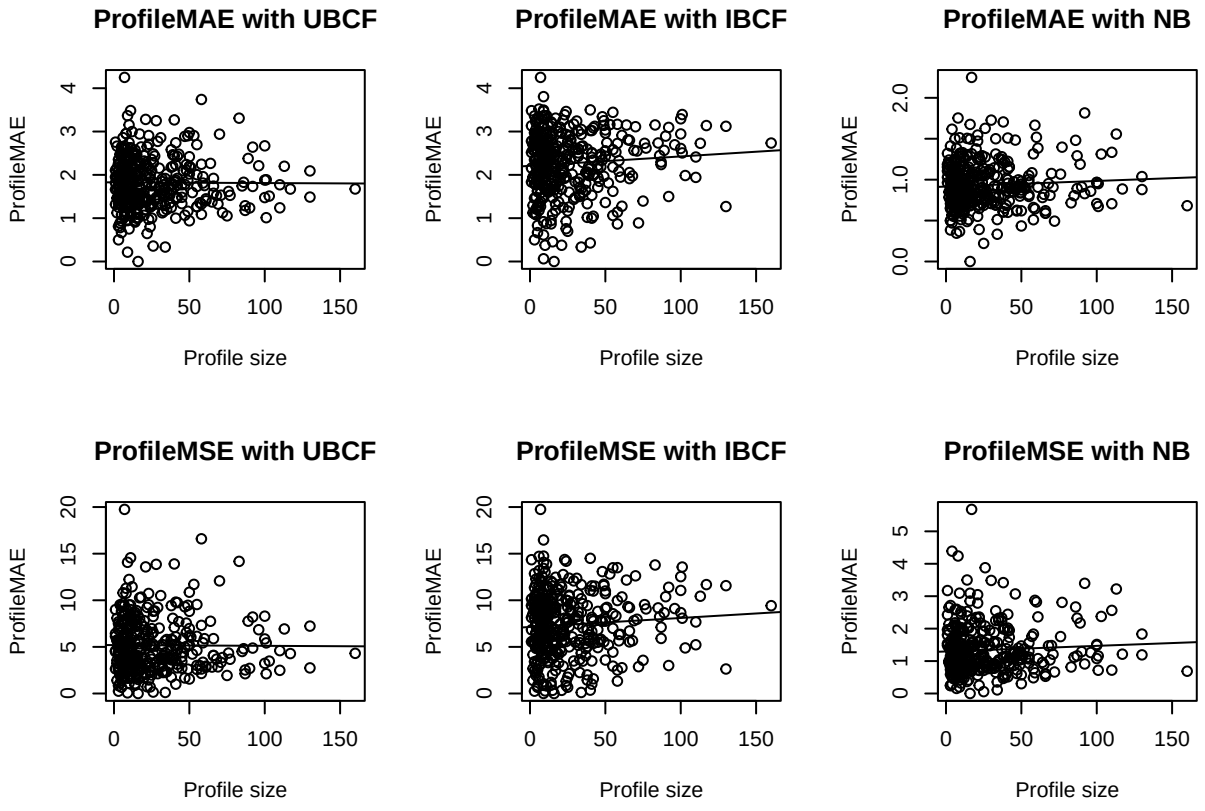
```
## # A tibble: 5 x 4
## # Groups:   user_id [5]
##   user_id item_id rating timestamp
##     <int>   <int>  <int> <dttm>
## 1     196     242      3 1997-12-04 15:55:49
## 2     186     302      3 1998-04-04 20:22:22
## 3      22     377      1 1997-11-07 07:18:36
## 4     244      51      2 1997-11-27 05:02:03
## 5     166     346      1 1998-02-02 05:33:16
```

5. Results and discussion

We attempted to run the algorithm with the entire truncated dataset. However after leaving the algorithm run the over night, it still had not finished calculating the ProfileError with error and prediction parameter set to "MAE" and "UBCF" respectively. As we had an estimated runtime of 45 min, 45 min and 30 min for our UBCF, IBCF and Naive Bayes implementations, the time allotted for these calculations was not sufficient in hinsight. These estimates were reached by extrapolating runtime of the algorithms based on sub-datasets with 1000 observations.Based on these circumstances, we decided to truncate the dataset. Thus, we created a smaller version of our dataset, consisting of the first 10000 entries.

```
ml_data_trunc_10k <- ml_data_trunc[1:10000,]
```

With this sample size, the runtime of the algorithms was almost perfectly in line with our estimations (around 4 hours). This indicates we underestimated the time and potentially space complexity of the algorithms we implemented by a factor of 10. This analysis yielded the following results:



As we can see in the graphs the regression lines remain stable across profile size, although the data spreads

less on average as profile size increases. This is interesting, because the in the paper we aim to recreate, the authors observed a different trend, namely that ProfileMAE decreases steadily up to a profile size of around 300, while spread in the data increases as profile size increases. We assume this difference is attributable to two factors: The difference in data and prediction algorithm implementation used. the latter seems to be the main driver of the effect as our data spreads more in general and the intercept of our regression line seems to be consistently larger as well. As the regression lines are almost constant we believe the best way to compare algorithms is to compare their regression intercepts:

```
##  MAE UBCF  MAE IBCF    MAE NB  MSE UBCF  MSE IBCF    MSE NB
## 1.8266181 2.2082813 0.9152103 5.1936562 7.1358883 1.2960378
```

Based on these intercepts we can see that the Naive Bayes predictor performed best based on both of our error calculation methods. We ran the same algorithm with a sub-dataset of 30.000 entries. As shown below the performance of UBCF and IBCF improved dramatically both in terms of MAE and MSE. Naive Bayes performed very similarly to the run with 10.000 entries, which leads us to believe its performance is not as predicated on the amount of observations compared to UBCF and IBCF.

```
##  MAE UBCF  MAE IBCF    MAE NB  MSE UBCF  MSE IBCF    MSE NB
## 1.2080222 1.6999053 0.9136347 2.4904201 4.7058421 1.2612606
```

Overall, Naive Bayes still performes best compared to the other two algorithms. However, UBCF and IBCF made significant improvements when sample size was increased to 30.000

6. Limitations

We discovered two massive limitations that our way of recreating the paper came with. The first one is a runtime issue and second one is the difficult recreation of the paper method. The runtime issues result from two key factors. The first one lies within the chosen dataset. With the number of entries of around 100.000 it already requires some computing time. Additionally, we compute for three algorithms for two measures that add quite a lot to the runtime. The resulting formula for our runtime equals to O(U) * O(NlogN) * O(prediction algorithm) where N corresponds to the number of ratings in a user's profile. We had a whole day scheduled for running the algorithms but despite the code running over night to get our results, it could not finish. The recreating was difficult as due to the specific method of the paper. Some algorithms packages were not usable as it would not have given us the desired output. Therefore, the code for the algorithms needed to be made from scratch, which were not the focus of our project. This added a whole different layer of complexity because a deeper understanding of the algorithms was needed to code them correctly. Though the code was tested for path completeness, we neither had the time nor the resources to optimize it with regards to runtime, which is most likely contributing to the inability to scale the functions to the ML100k in the alloted time.

7. Conclusion

The evaluation of dynamic properties is an important part when evaluating recommender systems and should be implemented whenever possible. The paper does a great job by highlighting its importance and providing an algorithm that assist with this evaluation. We did our best to recreate the paper however once we visualized the data, we noticed some large differences. First of all, the spread of our Profile MAE was a lot larger, with values between 0 and 4, while the Profile MAE of the paper described values between 0 and 1. Second of all in our Dataset, the more entries we have and the more time passes, the more accurate our recommenders seem to get! In the paper time seems appears to have the opposite effect - as more time passes, the less accurate their predictions seem. Overall we found this approach very interesting and have decided that one would need further research to fully test and understand the Profile MAE. Additionally newer and better algorithms can and will be developed over time.

8. References

Unit 1: Application for Data Science by Univ. Prof. Dr. Dorner et al.

Dataset: MovieLense 100k, url: https://grouplens.org/datasets/movielens/100k/

Robin Burke. 2010. Evaluating the dynamic properties of recommendation algorithms. In Proceedings of the fourth ACM conference on Recommender systems (RecSys '10). Association for Computing Machinery, New York, NY, USA, 225–228. https://doi.org/10.1145/1864708.1864753

9. Appendix

R code of the algorithms implemented:

```
if (!require("tidyverse")) install.packages("tidyverse"); library(tidyverse)
if (!require("lubridate")) install.packages("lubridate"); library(lubridate)
if (!require("naivebayes")) install.packages("naivebayes"); library(naivebayes)
if (!require("e1071")) install.packages("e1071"); library(e1071)

# set work directory
setwd('your wd here')

# clean the data
# F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets:
# History and Context. ACM Transactions on Interactive Intelligent
# Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages.
# DOI=http://dx.doi.org/10.1145/2827872
ml_data <- read.delim('location of ML100k data here', header=FALSE)
#set column names
colnames(ml_data) <- c('user_id', 'item_id', 'rating', 'timestamp')
#set timestamps to human readable format
ml_data[4] <- as.POSIXct("1970-01-01 00:00:00", format = "%Y-%m-%d %H:%M:%S") + unlist(ml_data[4])

#replicating the truncation of the dataset described in the paper
cutoff <- as.POSIXct("2000-12-31 00:00:00", format = "%Y-%m-%d %H:%M:%S")
ml_data_trunc <- ml_data[ml_data$timestamp < cutoff,]

#initiate the lower and upper bounds of reviews per profile from the paper
lower_bound <- 15
upper_bound <- 650
ml_data_trunc <- ml_data_trunc %>% group_by(user_id) %>% filter(n() > lower_bound)
ml_data_trunc <- ml_data_trunc %>% filter(n() < upper_bound)


#implement UBCF, IBCF and Naive Bayes to get predictions for specific user-item pairs

#implement UBCF, IBCF and Naive Bayes to get predictions for specific user-item pairs

Naive_Bayes <- function(data, user_id, item_id, k = 50) {

  user_id <- as.integer(user_id)
  item_id <- as.integer(item_id)
  k <- as.integer(k)
```

```
  # check if the input arguments are valid
  m <- match.arg(colnames(data), c('user_id', 'item_id', 'rating'), several.ok = TRUE)
  stopifnot("Columnnames are not fully matching  c('user_id', 'item_id', 'rating')" = length(m) == 3)
  t <- match.arg(sapply(ml_data_trunc, typeof), c('integer', 'integer', 'integer'), several.ok = TRUE)
  stopifnot("Columntypes are not fully matching  c('integer', 'integer', 'integer')" = length(t) == 3)
  stopifnot("user_id argument is not of type 'integer'" = typeof(user_id) == 'integer')
  stopifnot("item_id argument is not of type 'integer'" = typeof(item_id) == 'integer')
  stopifnot("k argument is not of type 'integer'" = typeof(k) == 'integer')

  # Create a new model everytime the function is called
  nbModel <- naiveBayes(rating ~ user_id + item_id, data = data)

  # Create a data frame with the user and item IDs in order to pass it through the predict function
  newdata <- data.frame(user_id = user_id, item_id = item_id)

  # We us our Naive Bayes Model to predict the rating of our Dataframe consisting of our user and item
  predictedRating <- predict(nbModel, newdata = newdata, type = "raw")

  # The predict function returns a dataframe with probabilities, so we try to turn that into a singular
  predicted_rating <- sum(predictedRating[1, ] * c(1, 2, 3, 4, 5))

  # Return the predicted value
  return(predicted_rating)
}


UBCF <- function(data, user_id, item_id, k = 50){
  # returns a prediction for a user item pair using UBCF
  # data needs to be a list with 3 columns named user_id, item_id and rating
  # these columns all need to have entries of type integer

  # try to coerce user_id, item_id and k to integers
  user_id <- as.integer(user_id)
  item_id <- as.integer(item_id)
  k <- as.integer(k)

  # check if the input arguments are valid
  m <- match.arg(colnames(data), c('user_id', 'item_id', 'rating'), several.ok = TRUE)
  stopifnot("Columnnames are not fully matching  c('user_id', 'item_id', 'rating')" = length(m) == 3)
  t <- match.arg(sapply(ml_data_trunc, typeof), c('integer', 'integer', 'integer'), several.ok = TRUE)
  stopifnot("Columntypes are not fully matching  c('integer', 'integer', 'integer')" = length(t) == 3)
  stopifnot("user_id argument is not of type 'integer'" = typeof(user_id) == 'integer')
  stopifnot("item_id argument is not of type 'integer'" = typeof(item_id) == 'integer')
  stopifnot("k argument is not of type 'integer'" = typeof(k) == 'integer')

  # format data and find relevant users
  data_matrix <- xtabs(rating ~ user_id + item_id, data = data)
  relevant_users <- names(which(data_matrix[, as.character(item_id)]>0))

  if(length(relevant_users) > 0){
    similarities <- cor(t(data_matrix))[as.character(user_id), relevant_users]
  }
  else{return(sum(data_matrix[as.character(user_id),][data_matrix[as.character(user_id),] > 0]) / lengt
```

```
  if(length(similarities) < k){
    similarities_ordered <- (similarities[order(similarities,decreasing= T)])[1:length(similarities)]
  }
  else{similarities_ordered <- (similarities[order(similarities,decreasing= T)])[1:k]}

  if(length(similarities) > 1){
    k_relevant_users <- names(similarities_ordered)
    relevant_means <- rowMeans(data_matrix[k_relevant_users,])
  }
  else{
    k_relevant_users <- relevant_users
    relevant_means <- mean(data_matrix[k_relevant_users,][data_matrix[k_relevant_users,] > 0])
  }


  # weight the ratings of the k most similar users
  relevant_ratings <- data_matrix[k_relevant_users, as.character(item_id)]
  prediction <- mean(data_matrix[as.character(user_id),]) + (similarities_ordered %*% (relevant_ratings

  if(prediction < 0){
    return(0)
  }
  else{return(prediction)}
}


adjusted_cosine_similarity <- function(data_matrix, item_id) {
  # Calculate the mean ratings for each user
  user_means <- rowMeans(data_matrix, na.rm = TRUE)

  # Center the ratings for each item by subtracting the mean rating for each user
  centered_data_mat <- t(scale(t(data_matrix), scale = FALSE))

  # Compute the adjusted cosine similarity between item_id and every other item
  similarities <- numeric(length = ncol(data_matrix))
  for (i in 1:ncol(data_matrix)) {
    if (i != item_id) {
      numerator <- sum(centered_data_mat[, as.character(item_id)] * centered_data_mat[, i], na.rm = TRU
      denominator <- sqrt(sum(centered_data_mat[, as.character(item_id)]^2, na.rm = TRUE)) * sqrt(sum(c
      similarities[i] <- numerator / denominator
    }
    else{
      similarities[i] <- 1
    }
  }

  # Return the similarities
  names(similarities) <- colnames(data_matrix)
  return(similarities)
}

IBCF <- function(data, user_id, item_id, k = 50){
  # returns a prediction for a user item pair using IBCF
```

```
  # data needs to be a list with 3 columns named user_id, item_id and rating
  # these columns all need to have entries of type integer

  # try to coerce user_id, item_id and k to integers
  user_id <- as.integer(user_id)
  item_id <- as.integer(item_id)
  k <- as.integer(k)

  # check if the input arguments are valid
  m <- match.arg(colnames(data), c('user_id', 'item_id', 'rating'), several.ok = TRUE)
  stopifnot("Columnnames are not fully matching  c('user_id', 'item_id', 'rating')" = length(m) == 3)
  t <- match.arg(sapply(ml_data_trunc, typeof), c('integer', 'integer', 'integer'), several.ok = TRUE)
  stopifnot("Columntypes are not fully matching  c('integer', 'integer', 'integer')" = length(t) == 3)
  stopifnot("user_id argument is not of type 'integer'" = typeof(user_id) == 'integer')
  stopifnot("item_id argument is not of type 'integer'" = typeof(item_id) == 'integer')
  stopifnot("k argument is not of type 'integer'" = typeof(k) == 'integer')

  # format data and find relevant items
  data_matrix <- xtabs(rating ~ user_id + item_id, data = data)
  relevant_items <- names(which(data_matrix[as.character(user_id), ]>0))

  # find similarities and filter out k most similar items
  if(length(relevant_items) > 0){
    similarities <- adjusted_cosine_similarity(data_matrix, item_id)[relevant_items]
  }
  else{return(sum(data_matrix[,as.character(item_id)][data_matrix[,as.character(item_id)] > 0]) / lengt
  if(length(similarities) < k){
    similarities_ordered <- (similarities[order(similarities,decreasing= T)])[1:length(similarities)]
  }
  else{similarities_ordered <- (similarities[order(similarities,decreasing= T)])[1:k]}

  if(length(similarities) > 1){
    k_relevant_items <- names(similarities_ordered)
  }
  else{
    k_relevant_items <- relevant_items
  }

  prediction <- sum(similarities_ordered * data_matrix[as.character(user_id), k_relevant_items]) / sum(

  if(prediction < 0){
    return(0)
  }
  else{return(prediction)}
}


#implement ProfileMAE algorithm
ProfileError <- function(data, error = c('MAE', 'MSE'), prediction = c('UBCF', 'IBCF', 'Naive_Bayes')){
  # data needs to be a list with 4 columns and the columnnames user_id, item_id, rating and timestamp
  # these columns need to have entries of type integer, integer, integer and double respectively
  # one can choose the profile error measure, which can be set to either MAE or MSE (default being MAE)
```

```r
#check if the required packages are loaded and load them if not
install.packages(setdiff(c('tidyverse', 'naivebayes'), rownames(installed.packages())))


#check if the input arguments are valid
stopifnot("Input data with incorrect column numbers: need 4" = ncol(data) == 4)
m <- match.arg(colnames(data), c('user_id', 'item_id', 'rating', 'timestamp'), several.ok = TRUE)
stopifnot("Columnnames are not fully matching  c('user_id', 'item_id', 'rating', 'timestamp')" = leng
t <- match.arg(sapply(ml_data_trunc, typeof), c('integer', 'integer', 'integer', 'double'), several.o
stopifnot("Columntypes are not fully matching  c('integer', 'integer', 'integer', 'double')" = length
match.arg(error)
match.arg(prediction)


user_ids <- unique(data$user_id) #retrieve unique user ids
c <- numeric(length(user_ids)) #initialize vector to count the number of ratings per user
eps <- numeric(length(user_ids)) #initialize vector to sum error terms for each user

for (u in user_ids) {
  p <- data[data$user_id == u,]
  ps <- p[order(p$timestamp),]
  u_index <- match(u, user_ids)


  for (j in 1:length(ps$user_id)) {
    e <- ps[j,]
    data_j <- data[data$timestamp < e$timestamp,][,1:3]

    # predict the rating via average rating of the item in question when the user u has no previous r
    if (j == 1 | e$item_id %in% data_j$item_id == FALSE | ps[1,]$timestamp == ps[j,]$timestamp){
      q <- colMeans(data_j[data_j$item_id == e$item_id,1:3])[3]
      if(is.na(q)){
        q <- 3
      }
    }

    else{
      if(prediction == 'UBCF'){
        q <- UBCF(data_j, u, e$item_id)
      }
      if(prediction == 'IBCF'){
        q <- IBCF(data_j, u, e$item_id)
      }
      if(prediction == 'Naive_Bayes'){
        q <- Naive_Bayes(data_j,u, e$item_id)
      }
    }

    if(error == 'MAE'){
      eps_j <- abs(e$rating - q)
    }
    else{
      eps_j <- (e$rating - q)^2
```

```
      }


      eps[u_index] <- eps[u_index] + eps_j
      c[u_index] <- c[u_index] + 1
    }
  }
  res <- cbind(user_ids, eps/c)
  if(error == 'MAE'){
    colnames(res) <- c('user_id', 'ProfileMAE')
  }
  else{
    colnames(res) <- c('user_id', 'ProfileMSE')
  }

  return(res)
}
```