Recommendation Engine

anjal hussan

5/2/2018


```
library("SlopeOne")
library("data.table")
library("lazyeval")
library("plotly")
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library("jsonlite")
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("readr")
library("knitr")
```

Introduction

Creating a recommendation engine that displays a list of top recommended music artists and several of their top tracks based on the data that already provided about some favorite artists of a user.

There will be two part code in this project. First part will be offline part. Offline part of the code will consisit of data from a source and it will construct a recommender model when we run first time.

Second part will be online part. Online part can be implemented as part of an interactive system (e.g. a website) that reacts to user input and provides a recommendation of top artists. The online part can be re-run every time there is a new input of favorite artists by the user.

Setup offline

Acquire the input data

Download the dataset with the compiled artist ratings from Last.fm first. This dataset includes multiple files. The main content is the file with the number of times a user has played any track by a particular artist. Also, a database with artist ids and names is provided.

```
### Load Data ----

# Download the ratings dataset
if(!dir.exists("data")){dir.create("data")}

if(!file.exists("data/hetrec2011-lastfm-2k.zip")) {
 download.file(
   url = "http://files.grouplens.org/datasets/hetrec2011/hetrec2011-lastfm-2k.zip",
   destfile = "data/hetrec2011-lastfm-2k.zip",mode='wb', method = "auto")
  unzip("data/hetrec2011-lastfm-2k.zip",exdir = "data")
}

# Load ratings data

ratings = fread("data/user_artists.dat")

names(ratings) = c("user_id", "item_id", "rating")

ratings[, user_id := as.character(user_id)]
ratings[, item_id := as.character(item_id)]
```

```
# Load artist names and genre

artists = fread("data/artists.dat")
artists[,id := as.character(id)]
setkey(artists, id)
```

Exploration: Distributions

Count of ratings per user

```
ratings_per_user = ratings[,.(.N), by = user_id]
summary(ratings_per_user$N)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   50.00   50.00   49.07   50.00   50.00
```

Each user has rated a maximum of 50 artists

Count of ratings per artist

```
# Count of ratings per artist

ratings_per_artist = ratings[,.(.N), by = item_id]
summary(ratings_per_artist$N)
```

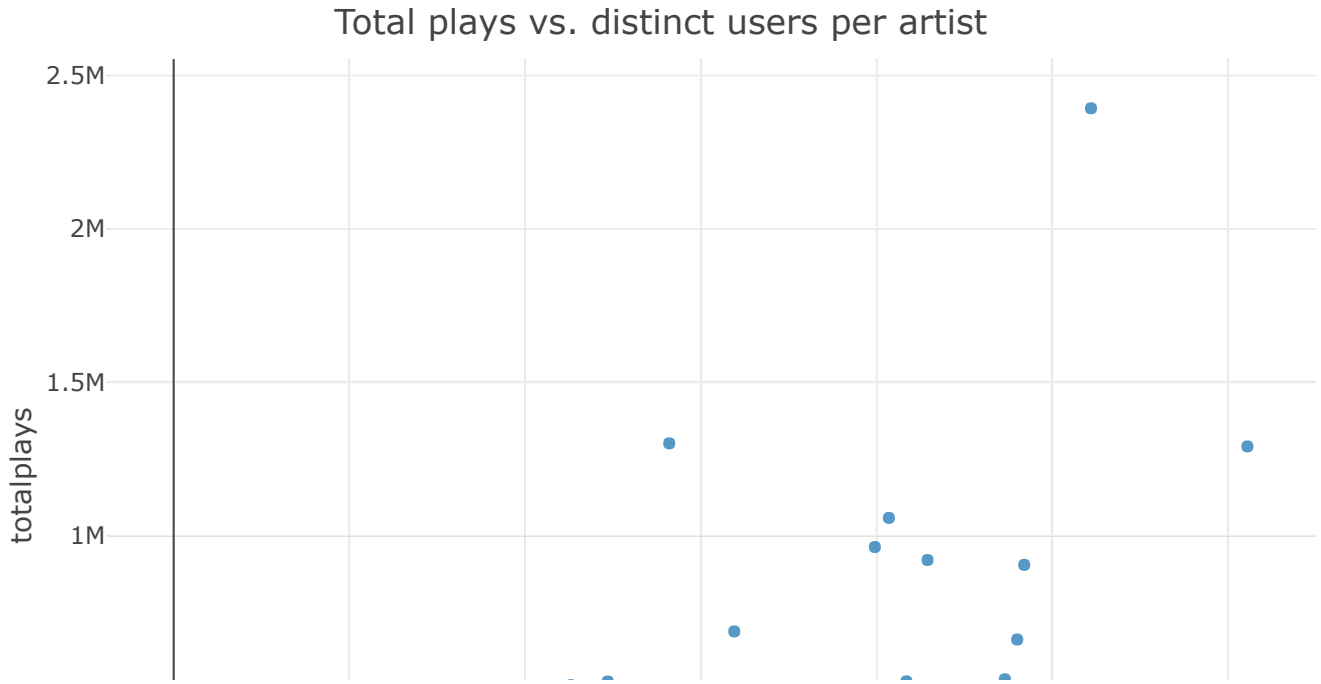
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   5.265   3.000 611.000
```

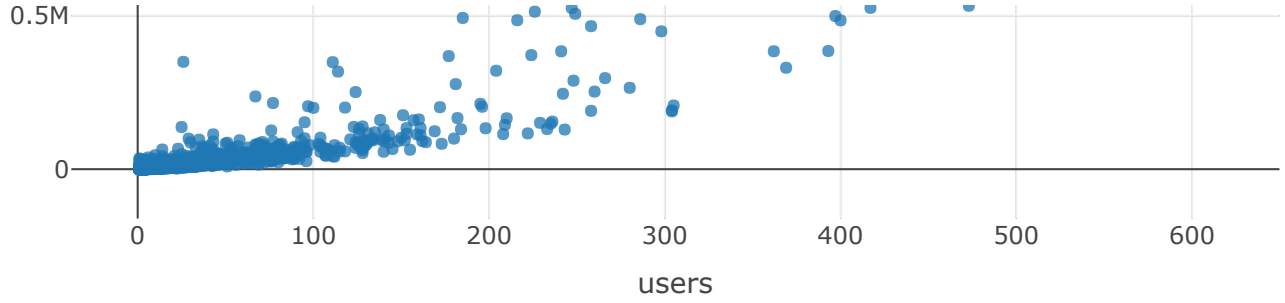
Each artist has received between 1 and 3 ratings, but some have received 611 ratings

Most popular artists by number of plays

```
if (!file.exists("data/popularity.rda")) {
```

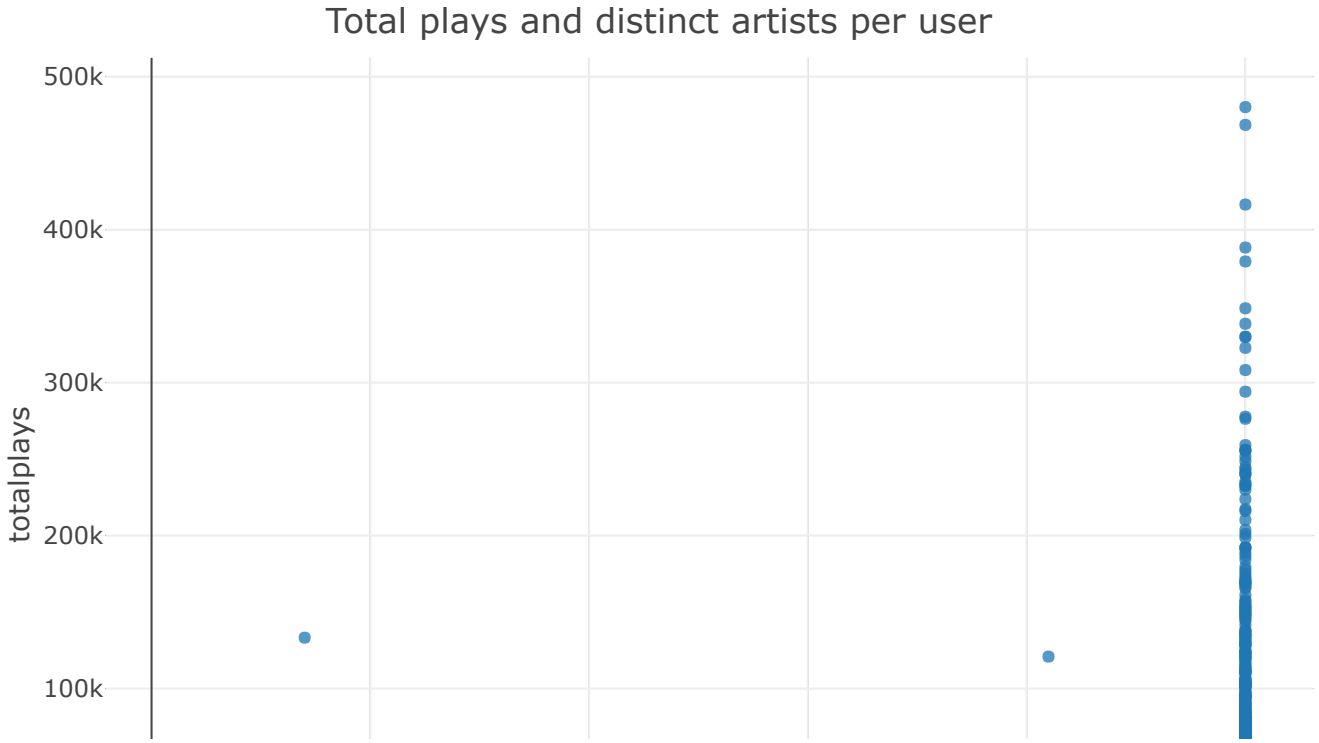
```
# Add artist names to ratings

artistnames = artists[,.(item_id = id, name)]
ratings1 = artistnames[ratings, on = "item_id"]

# Most popular artists by number of plays

popularity = ratings1[,.(totalplays = sum(rating), users = n_distinct(user_id)),
                   by = .(name,item_id)]

# Store the popularity data
write_rds(popularity,"data/popularity.rda", compress = "gz")} else

  popularity = readRDS("data/popularity.rda")

plot_ly(x = ~users, y = ~ totalplays, data = popularity, type = "scatter", text = ~name, mode ="markers", alpha =
 0.75) %>%
  layout (title="Total plays vs. distinct users per artist")
```

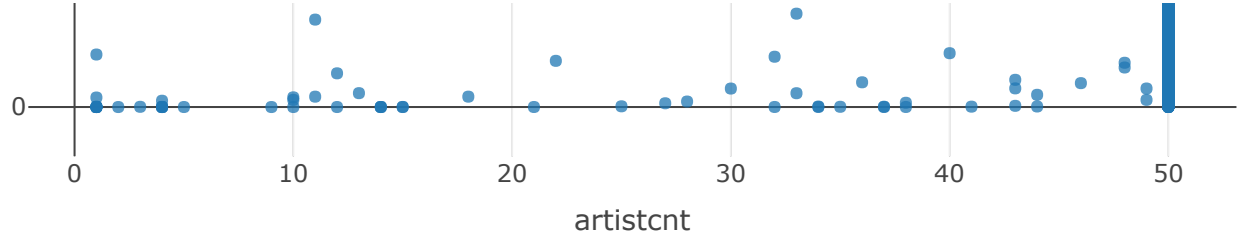




Distribution of user plays

```
useractivity = ratings[,.(totalplays = sum(rating), artistcnt = n_distinct(item_id)),
                        by = "user_id"]

plot_ly(x = ~artistcnt, y = ~ totalplays, data = useractivity, type = "scatter",
        text = ~user_id, mode ="markers", alpha = 0.75) %>%
  layout (title="Total plays and distinct artists per user")
```





Prepare the data for modeling

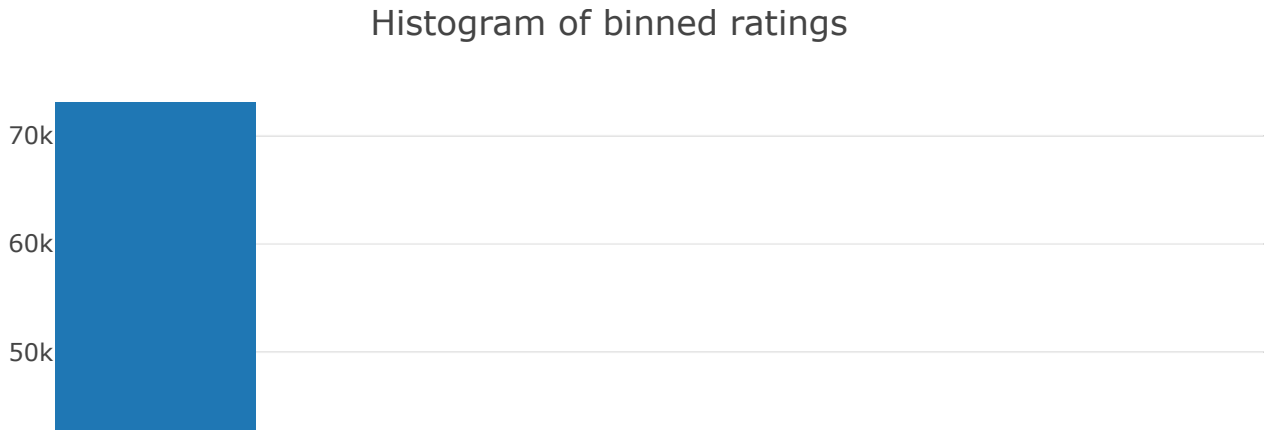
Filter ratings to exclude missing values and users with low number of rated artists

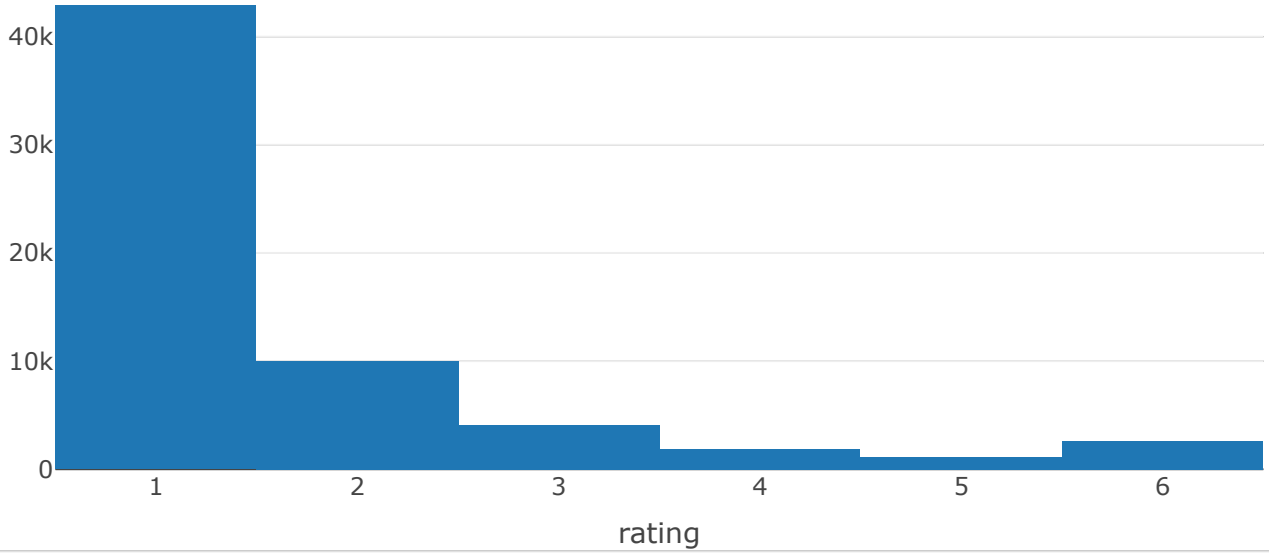
```
ratings = ratings[is.na(rating) ==F,]
ratings = ratings[!(user_id %in% useractivity[artistcnt < 10]$user_id),]
```

Frequency of top bin per artist vs. total plays

As the ratings are just the counts of times an artist was played by the user, they are not comparable between the users. In order to make them comparable, they are assigned to one of six bins according to playing frequency per user.

```
ratings = ratings[, rating := cut(rating,breaks = 6,labels = seq(1:6), include.lowest = T, ordered_result = T), b
y = user_id][,rating:=as.numeric(rating)][order(user_id, -rating)]

toprating = max(ratings$rating, na.rm = T)
plot_ly(x = ~rating, data = ratings, type = "histogram") %>% layout(title="Histogram of binned ratings")
```

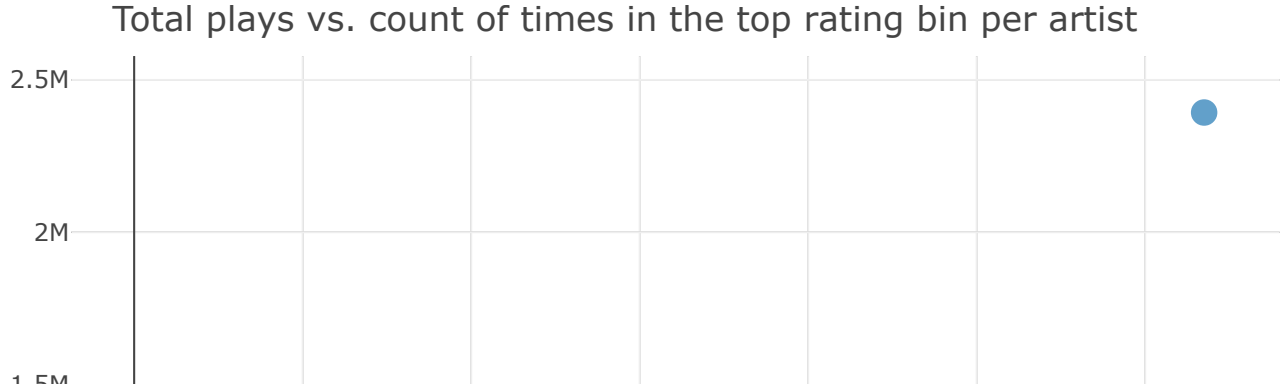


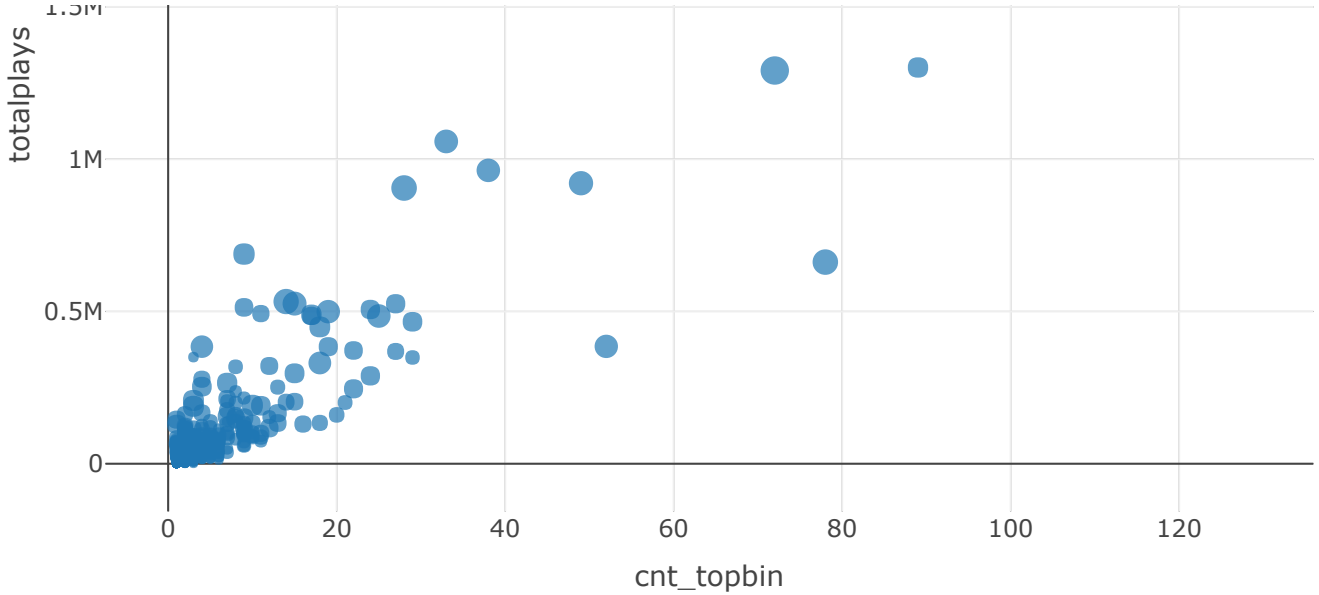



```
setkey(ratings, user_id, item_id)
```

As expected, most ratings fall into the lower bins. Now we can inspect how various artists score in terms of the relative populatity (being in the top bin by the number of plays) among users.

```
topscores = ratings[rating %in% c(5,6), .(cnt_topbin = .N), by = .(item_id)]
topscores = popularity[topscores, on = .(item_id)]

plot_ly(x = ~cnt_topbin, y =~totalplays, size = ~users, data = topscores, text = ~name,
        type = "scatter") %>% layout(title="Total plays vs. count of times in the top rating bin per artist")
```





We see that in this sample, Britney Spears is the absolute leader both in terms of relative popularity, and by the total number of plays. This is likely to influence the recommendations.

Build the Slope One model

We use the binned ratings to build the slope one model which applies its own normalization step. The generated model contains the weighted average deviance (difference in rating) for each pair of items (artists). As the input dataset is fairly small, a large number of such pairs only come from a small number of users, making the value of the inferred average deviance extremely unstable and sensitive to individual user preferences.
In order to build a generalizable model, all entries in the model table that come from less than 23 users (support is less than 23), are discarded. This cutoff value has been defined in model iterations. Only the shortened model is stored and used later for predictions.

```
if (!file.exists("data/slopeOneModel.rda")) {
# Building Slope One model:

start = proc.time()
ratings_norm = normalize_ratings(ratings)
model = build_slopeone(ratings_norm$ratings)
finish = proc.time() - start
```

```
# Reduce the model to only stable ratings with support over 25

model_short = model[support>=23,]

# Store the model
write_rds(model_short,"data/slopeOneModel.rda",compress = "xz")} else

  model_short = readRDS("data/slopeOneModel.rda")
  model_short = data.table(model_short)
```

In order to provide predictions, a separate dataset should be generated and stored: a list of possible items. This list is a subset of all artists rated by at least 6 users.

Generate targets - only the artists that were rated by at least 20 users will be used for rating prediction.

```
if (!file.exists("data/targets.rda")) {
# Create a dataset of items listened to by at least X users

targets = ratings[,.N, by=item_id][N>=20,]

targets = targets[,.(item_id)]
targets = unique(targets)

# Store

  write_rds(targets,"data/targets.rda",compress = "gz")

  } else

    targets = readRDS("data/targets.rda")
    targets = data.table(targets)
```

Implement Online part

This part can be implemented as part of an interactive system (e.g. a website) that reacts to user input and provides a recommendation of top artists. An exemplary user input is provided below in order to demonstrate the function of the recommender system.

Predictions for a new user

Let us assume that a new user has provided the following list of favorite artists: “Madonna”, “Lady Gaga”, “Rihanna”, “Bruno Mars”.

The input is stored in the `userinput` variable, as shown in the code snippet below. Only the contents of this variable should be adjusted to provide new recommendations.

We process this user input

```
# New user inputs a list of artist names, some other sets of inputs provided for testing

userinput = c("Madonna","Lady Gaga", "Rihanna", "Bruno Mars")
# userinput = c("Slipknot","In Flames")
# userinput = c("Oasis", "Blur", "Garbage")


userinput = tolower(userinput)

# Get artist ids from last.fm db for the artists them

userartists = artists[tolower(name) %in% userinput,.(id,name)]
userartists_id = userartists$id
```

Construct model input for the user and generate prediction for all targets. This takes some time, as a prediction function is applied to every artist in the list of targets in order to generate an ordered list of top rated artists.

```
# As these are the favorite artists, assign top rating to each of them

ratings_norm = normalize_ratings(ratings)
newuser_rating = data.frame(user_id = as.character(rep(9999, length(userartists_id))),
                     item_id = as.character(userartists_id),
                     rating = 6)
newuser_rating = data.table(newuser_rating)

targets$user_id = as.character(9999)

start = proc.time()
```

```
predict_new = predict_slopeone(model = model_short,
                               targets = targets,
                               ratings = newuser_rating)
finish = proc.time() - start

predict_new = unnormalize_ratings(normalized = ratings_norm,ratings = predict_new)

# Generate top N predictions

top_N = function (predictions, n) {
  out = predictions[order(-predicted_rating)][1:n,]
  out = merge(out, artists, by.x = "item_id", by.y = "id")
  out[,pictureURL := NULL]
  #Exclude accidental repetition of users own input in the prediction.
  out = out[!(item_id %in% newuser_rating$item_id),]

  out[order(-predicted_rating)]

}

top_artists = top_N(predict_new,10)
```

Prediction finished in 3.418 seconds.

The output of the top-N function looks as follows:

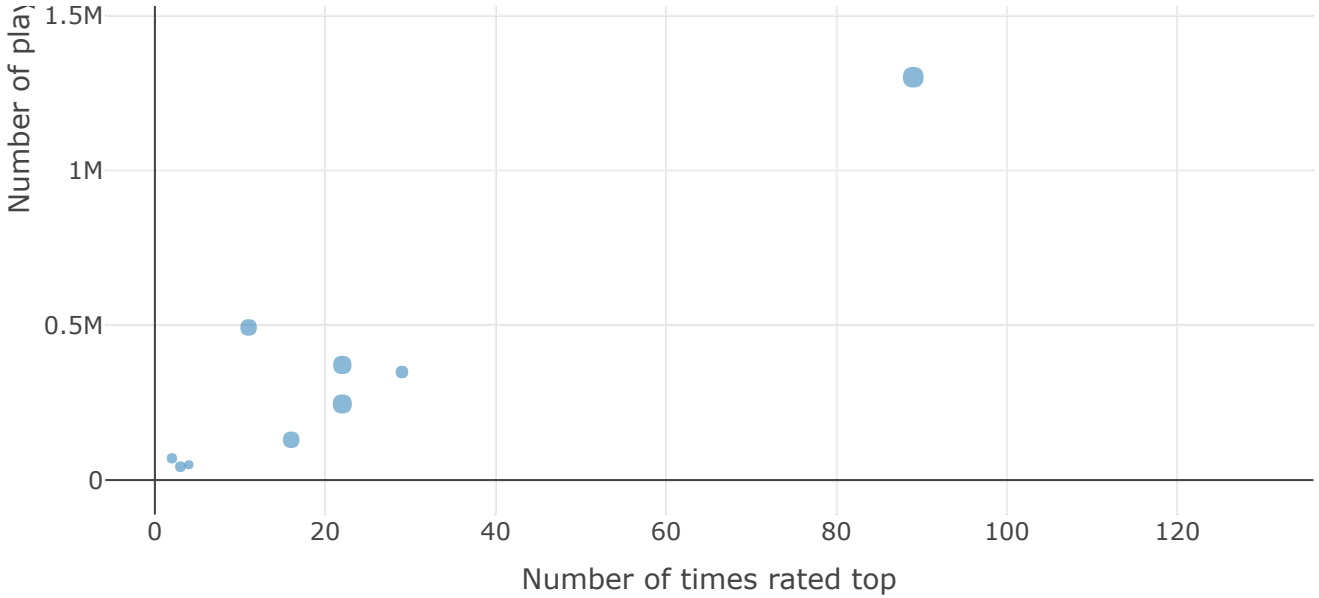
```
#load('stringi')
kable(top_artists)
```

item_id	user_id	predicted_rating	name	url
72	9999	10.900850	Depeche Mode	http://www.last.fm/music/Depeche+Mode
51	9999	10.593875	Duran Duran	http://www.last.fm/music/Duran+Duran
289	9999	10.229518	Britney Spears	http://www.last.fm/music/Britney+Spears

item_id	user_id	predicted_rating	name	url
173	9999	9.700104	Placebo	http://www.last.fm/music/Placebo
707	9999	9.674790	Metallica	http://www.last.fm/music/Metallica
961	9999	9.538920	Tori Amos	http://www.last.fm/music/Tori+Amos
424	9999	9.510748	The Strokes	http://www.last.fm/music/The+Strokes
511	9999	9.474980	U2	http://www.last.fm/music/U2
951	9999	9.449635	Bon Jovi	http://www.last.fm/music/Bon+Jovi
683	9999	9.448403	John Mayer	http://www.last.fm/music/John+Mayer

Display a chart of listeners and plays for the recommended artists

```
chart_subset =  top_artists$item_id

user_popularity = topscores[item_id %in% chart_subset,]

plot_ly(x = ~cnt_topbin, y = ~ totalplays, data = user_popularity, type = "scatter",
        text = ~name, mode ="markers", alpha = 0.75, size = ~users) %>%
  layout (title="Total plays vs. relative popularity for artists in your recommendation",
          xaxis=list(title="Number of times rated top"),
          yaxis=list(title="Number of plays"))
```



The predicted ratings are heavily influenced by the most popular artists in the sample. However, a few relevant recommendations should also appear among the results.

Get top tracks for the top 3 artists

Now we can use the output of the top recommended artists to provide a list of top tracks by these artists. We fetch the list of top tracks from Last.fm via the TopTracks API.

```
lastfm_gettracks = function(inputartist) {

# Construct a string input recognized by the API

userqueryapi = inputartist

# Construct a string input recognized by the API
userqueryapi = tolower(userqueryapi)
userqueryapi = URLencode(userqueryapi)

api_param = function(tag, value) {
  paste(tag,value, sep = "=")
```

```
}

#http://ws.audioscrobbler.com/2.0/?method=artist.gettoptracks&artist=cher&api_key=bacaeacee2e79419fba13b5b2cc411c5&format=json

apikey = "bacaeacee2e79419fba13b5b2cc411c5"
baseurl = "http://ws.audioscrobbler.com/2.0/?"

api_param_string = paste(
  api_param("method", "artist.gettoptracks"),
  api_param("artist", userqueryapi),
  api_param("api_key", apikey),
  api_param("format", "json"),
  api_param("autocorrect", 1),
  api_param("limit", 5),sep = "&")
request_url = paste0(baseurl,api_param_string)

# See http://stackoverflow.com/questions/33200790/json-parsing-error-invalid-character
raw_output = readLines(request_url, warn = FALSE)

# Get the results DF and metadata out of the request JSON output

parsed_output = fromJSON(raw_output, simplifyDataFrame = T, flatten = T)

api_out = parsed_output$toptracks$track[,c("name","url")]
api_out$artist = as.character(inputartist)
api_out = api_out[,c("artist","name","url")]
names(api_out) = c("Artist", "Song name", "Link")

artistimg = parsed_output$toptracks$track$image[[1]][1]
artist_image_url = artistimg[nrow(artistimg),]

output = list(api_out, artist_image_url)

}
```

Here are the top tracks by the top 3 recommended artists

```
artist1 = lastfm_gettracks(top_artists[1,"name"])
artist2 = lastfm_gettracks(top_artists[2,"name"])
artist3 = lastfm_gettracks(top_artists[3,"name"])
```

```
kable(artist1[1])
```

Artist	Song name	Link
Depeche Mode	Enjoy the Silence	https://www.last.fm/music/Depeche+Mode/_/Enjoy+the+Silence
Depeche Mode	Personal Jesus	https://www.last.fm/music/Depeche+Mode/_/Personal+Jesus
Depeche Mode	Precious	https://www.last.fm/music/Depeche+Mode/_/Precious
Depeche Mode	Just Can’t Get Enough	https://www.last.fm/music/Depeche+Mode/_/Just+Can%27t+Get+Enough
Depeche Mode	Strangelove	https://www.last.fm/music/Depeche+Mode/_/Strangelove

```
kable(artist2[1])
```

Artist	Song name	Link
Duran Duran	Hungry Like the Wolf	https://www.last.fm/music/Duran+Duran/_/Hungry+Like+the+Wolf
Duran Duran	Ordinary World	https://www.last.fm/music/Duran+Duran/_/Ordinary+World
Duran Duran	Girls on Film	https://www.last.fm/music/Duran+Duran/_/Girls+on+Film
Duran Duran	Come Undone	https://www.last.fm/music/Duran+Duran/_/Come+Undone
Duran Duran	Rio	https://www.last.fm/music/Duran+Duran/_/Rio

```
kable(artist3[1])
```

Artist	Song name	Link

Artist	Song name	Link
Britney Spears	Toxic	https://www.last.fm/music/Britney+Spears/_/Toxic
Britney Spears	Womanizer	https://www.last.fm/music/Britney+Spears/_/Womanizer
Britney Spears	…Baby One More Time	https://www.last.fm/music/Britney+Spears/_/…Baby+One+More+Time
Britney Spears	Piece of Me	https://www.last.fm/music/Britney+Spears/_/Piece+of+Me
Britney Spears	Gimme More	https://www.last.fm/music/Britney+Spears/_/Gimme+More

Reference

http://www.guidetodatamining.com/assets/guideChapters/DataMining-ch3.pdf

https://github.com/tarashnot/SlopeOne

https://rpubs.com/tarashnot/recommender_comparison

Last.fm website, http://www.lastfm.com

http://files.grouplens.org/datasets/hetrec2011/hetrec2011-lastfm-2k.zip Collected by Ignacio Fernández-Tobías with the collaboration of Iván Cantador and Alejandro Bellogín, Universidad Autonoma de Madrid

http://ir.ii.uam.es

http://www.last.fm/api/show/artist.getTopTags