Choose Your Own Project Submission

Juan Carlos Cortez Villarreal

Preface

This is my submission for the last part of the Data Science: Capstone! course provided by HarvardX in association with edX.org. The objective is to choose a database, apply datasciense analysis and report the findings.

1. Introduction

Let's stick with recommendation systems. This time we will use information from Last.fm Online Music System. This data was retrieved and formatted by the Information Retrieval group at Universidad Autonoma de Madrid for HetRec 2011 https://grouplens.org/datasets/hetrec-2011/

The reason to choose recommendation systems again and this database in particular is twofold:

- I don't want to drift too much apart from something that could be followed and understood quickly in the context of the course.
- I enjoy music very much, and this database is, in fact, the one that caught my eye the most.

A specific thought got stuck in my mind: In addition to getting an estimation and measuring it, could I get the data to actually suggest me some interesting music?

2. Analysis

The dataset contains information about friendship between users, user-defined tagging, and the listening habits of close to 2k users on artist level.

Other databases for song level prediction are available, for example: https://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html but our database is bigger, newer and has more variables. We'll leave song level prediction for future work.

We will only focus on the play count for our first analysis.

2.1. Data Exploration

The user_artist.dat file contains [userID, artistID] keys with a weight parameter that represent the amount of times the user has played that artist.

The artist.dat file contains the ids, names, URL and image URL of each of the artists associated with the dataset.

```
dl1 <- tempfile()
download.file("https://raw.githubusercontent.com/ciberneuro/dsLastFM/master/data/user_artists.dat", dl1)
plays <- fread(text = readLines(dl1), header = TRUE)
dl2 <- tempfile()
download.file("https://raw.githubusercontent.com/ciberneuro/dsLastFM/master/data/artists.dat", dl2)
artists <- fread(text = readLines(dl2), header = TRUE,
                 quote="", drop=c("url", "pictureURL"),
```

```
                col.names = c("artistID", "artistName"))
rm(dl1, dl2)
artists$artistName <- stri_trans_general(artists$artistName, "Any-Latin")
print(paste("The dataset has", dim(plays)[1], "rows and", dim(plays)[2], "columns."))
```

```
## [1] "The dataset has 92834 rows and 3 columns."
```

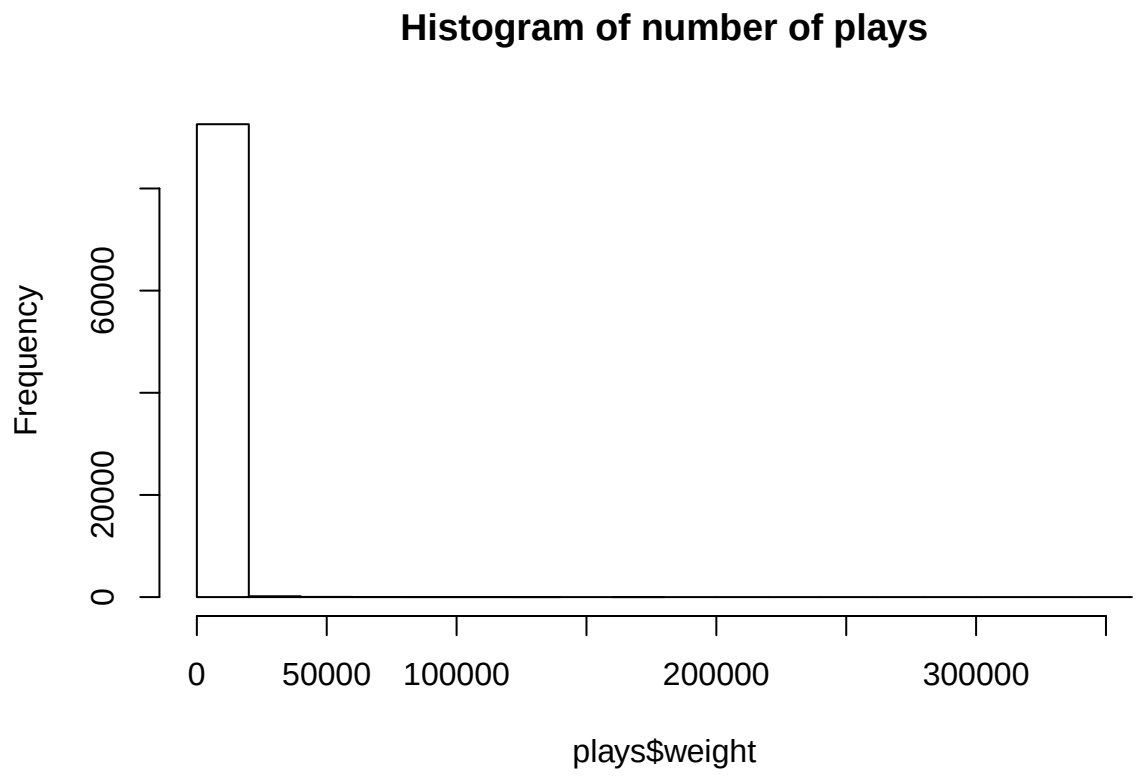
```
print(paste("The columns names are: ", toString(colnames(plays))))
```

```
## [1] "The columns names are:  userID, artistID, weight"
```

2.1.1. Dealing with outliers

We can see a histogram of the number of plays

```
hist(plays$weight, main="Histogram of number of plays")
```




```
writeLines(paste("The plays have an average of", mean(plays$weight),
                "\nwith a median of", median(plays$weight),
                "\nand a maximum of", max(plays$weight)))
```

```
## The plays have an average of 745.243930025637
## with a median of 260
## and a maximum of 352698
```

And notice that the median is less than half the average. This is because we have a very low number of really high values that skew the results.

```
outlier_limit <- boxplot.stats(plays$weight)$stats[5]
writeLines(paste("The statistical process that R uses for plotting,",
                 "\ndefines the outliers as the numbers above", outlier_limit))
```

```
## The statistical process that R uses for plotting,
## defines the outliers as the numbers above 1374
```

There are many ways to deal with outliers. So let's see this as a movie recommendation system, but rather than 1 to 5, let's assume a 1 to 1000 scale. The only significant difference in modeling this way is that in movies a 1 star score can be considered bad, but any number in this case should be considered a positive.

So let's just set anything above 1000 as 1000

```
plays$weight[plays$weight>1000] <- 1000
```

The 1000 might seem somewhat arbitrary, but when we apply a cutoff of 1374, we still get outliers of that new dataset. Instead of multiple executions or a more complex formula, let's just simplify it.

Another possible approach would be to normalize based on user. Meaning dividing all numbers by the user's average. However, that seems to punish high number of plays in users with very high numbers by comparison. This particular phase is the outlier handling, the algorithms will normalize in their own way.

2.1.2. About the artists

```
artist_plays <- plays %>%
  group_by(artistID) %>%
  summarize(n=n(), weight = mean(weight)) %>%
  left_join(artists, by="artistID")
artist_plays %>% arrange(desc(weight)) %>% top_n(10,weight)
```

```
## # A tibble: 785 x 4
##    artistID     n weight artistName
##       <int> <int>  <dbl> <chr>
##  1       74     1   1000 Basia
##  2       79     1   1000 Fiction Factory
##  3       83     1   1000 Cock Robin
##  4       87     1   1000 Deacon Blue
##  5       92     1   1000 Vitamin Z
##  6      336     1   1000 Vanessa Petruo
##  7      337     1   1000 Alexander
##  8      368     1   1000 gimjong-gug
##  9      372     1   1000 mongoru800
## 10      513     1   1000 Traff!c
## # ... with 775 more rows
```

When we look at the top listened artist by average weight, we find that there is a really high number of 1 user artists.

We have been taught that the algorithms should be able to account for low values by assigning them a penalty. However, the algorithms don't seem able to cope with such high quantity of these individual cases. Added to the fact that we want to receive suggestions from relatively popular bands, we filter them now.

```
plays <- plays %>%
  semi_join(artist_plays %>% filter(n>=25), by="artistID")
```

We limit the database to artists with 25 or more users because when we observe the data, 25 seems to be a small enough number to generate popular artists.

```
artist_plays <- plays %>%
  group_by(artistID) %>%
  summarize(n=n(), weight = mean(weight)) %>%
  left_join(artists, by="artistID")
artist_plays %>% arrange(desc(weight)) %>% top_n(10,weight)
```

```
## # A tibble: 10 x 4
##    artistID     n weight artistName
##       <int> <int>  <dbl> <chr>
##  1     2102    29   740. xìng tián lái wèi
##  2      289   522   691. Britney Spears
##  3      375    37   665. bāng qíayumi
##  4      292   407   647. Christina Aguilera
##  5      816    77   634. A Day to Remember
##  6      813    42   627. As I Lay Dying
##  7      805    26   624. Parkway Drive
##  8     2044    25   618. Sarah Brightman
##  9      152    71   606. Porcupine Tree
## 10      823    25   603. August Burns Red
```

2.2. Data Exploration of new dataset

The new dataset, filtered and bounded, now has the following characteristics:

```
print(paste("The dataset has", dim(plays)[1], "rows and", dim(plays)[2], "columns."))
```

```
## [1] "The dataset has 49612 rows and 3 columns."
```

```
print(paste("The columns names are: ", toString(colnames(plays))))
```

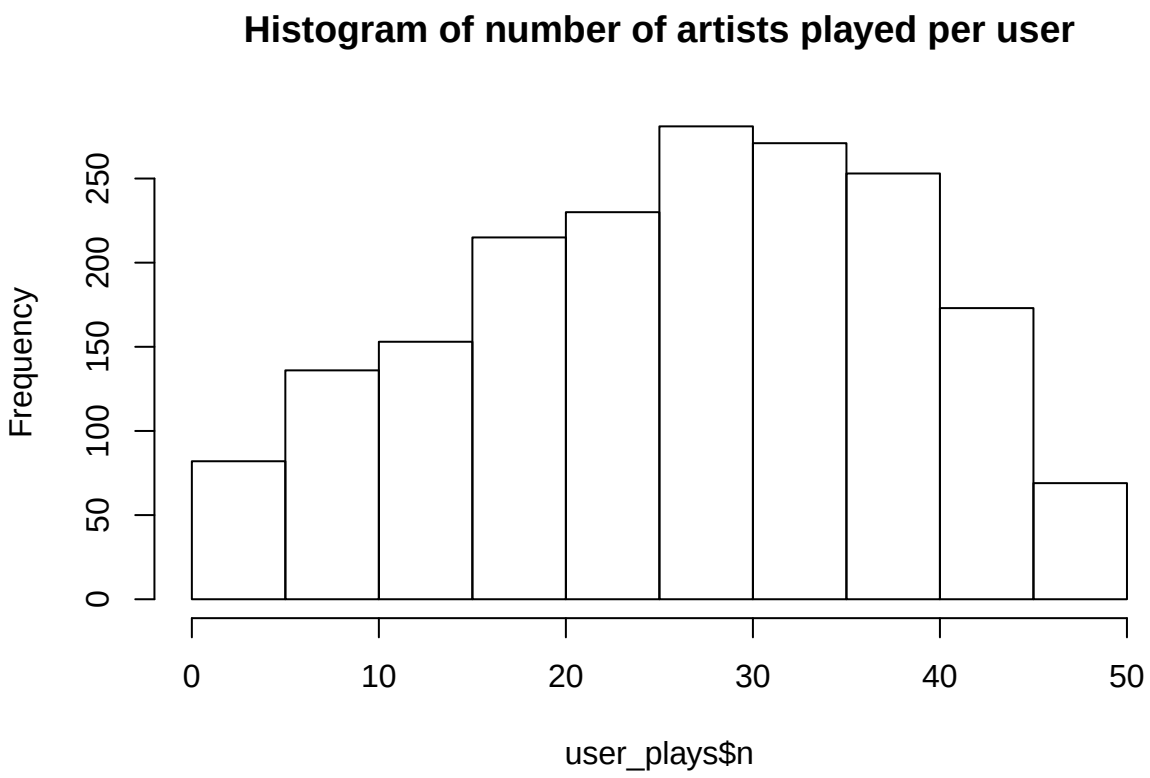
```
## [1] "The columns names are:  userID, artistID, weight"
```

2.2.1. About the users

```
print(paste("There are", length(unique(plays$userID)), "unique users"))
```

```
## [1] "There are 1863 unique users"
```

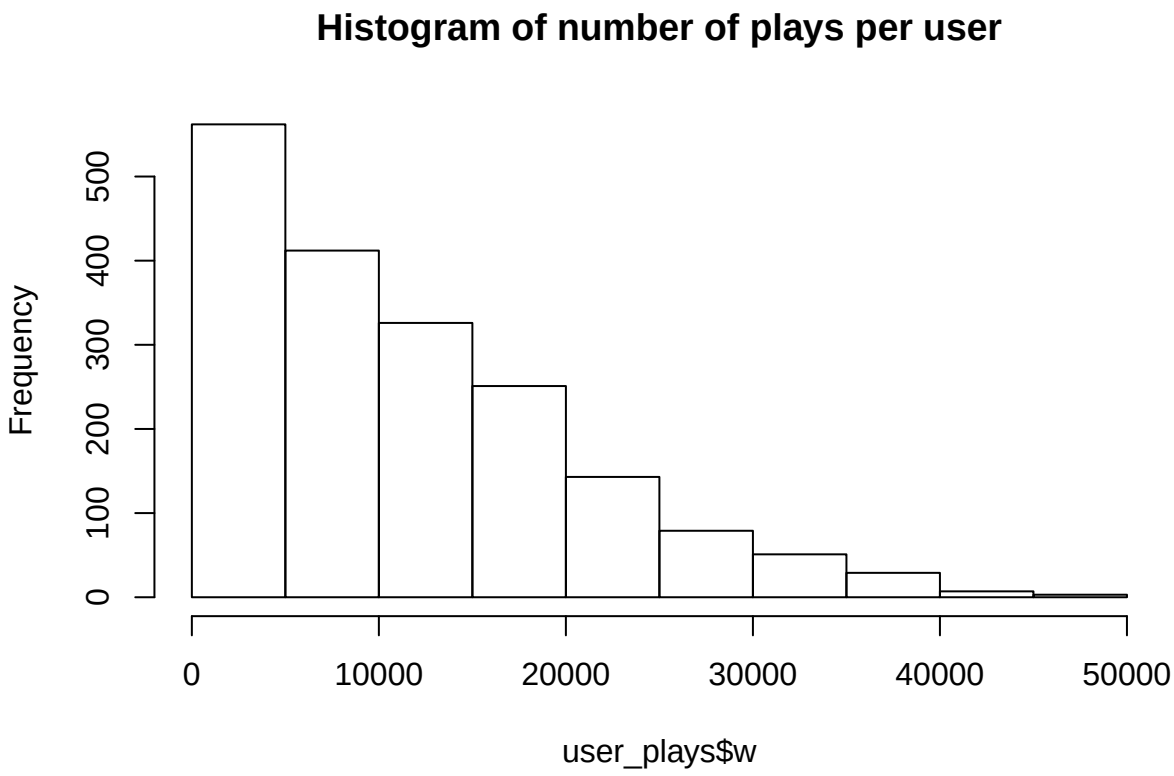
```
user_plays <- plays %>% group_by(userID) %>% summarize(n=n(), w = sum(weight))
hist(user_plays$n, main ="Histogram of number of artists played per user")
```




```
writeLines(paste("The average number of artists played by a user is",
                 mean(user_plays$n)))
```

```
## The average number of artists played by a user is 26.6301663982823
```

```
hist(user_plays$w , main="Histogram of number of plays per user")
```




```
writeLines(paste("Users have played an average of", mean(user_plays$w), "songs,",
            "\nwith a median of", median(user_plays$w),
            "\nand a maximum of", max(user_plays$w)
           ))
```

```
## Users have played an average of 11384.9801395598 songs,
## with a median of 9475
## and a maximum of 47137
```

2.2.2. About the artists

```
print(paste("There are", length(unique(plays$artistID)), "unique artists."))
```

```
## [1] "There are 638 unique artists."
```

```
writeLines(paste("The average artist is listened ", mean(artist_plays$weight),
                 "times\nby", mean(artist_plays$n), "users." ))
```

```
## The average artist is listened  402.819111400784 times
## by 77.7617554858934 users.
```

3. Method

3.1. First idea: Penalized Least Square method

Let's first analyze the Penalized Least Square method. This method is based on the idea that a particular user produces an effect or bias on the score prediction (number of plays in this case) based on the fact that not everyone plays the same amount of music. Also, a particular artist will add an effect or bias based on the fact that they are more liked than others.

We penalize values outside the mean that are estimated by one or too few samples in order to lower their uncertainty. And in the end we have to minimize the following equation:

$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

where
$y_{u,i}$ is our estimation of plays from user u to artist i
μ is the average number of plays of the dataset b_i is the artist effect or bias
b_u is the user effect or bias
λ is the penalty factor

3.1.2. Partition the data

```
set.seed(1)
test_index <- createDataPartition(y = plays$weight, times = 1, p = 0.1, list = FALSE)
trainx <- plays[-test_index,]
temp <- plays[test_index,]

validation <- temp %>%
  semi_join(trainx, by = "artistID") %>%
  semi_join(trainx, by = "userID")

removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userID", "artistID", "weight")
```

```
trainx <- rbind(trainx, removed)
rm(test_index, temp, removed)
```

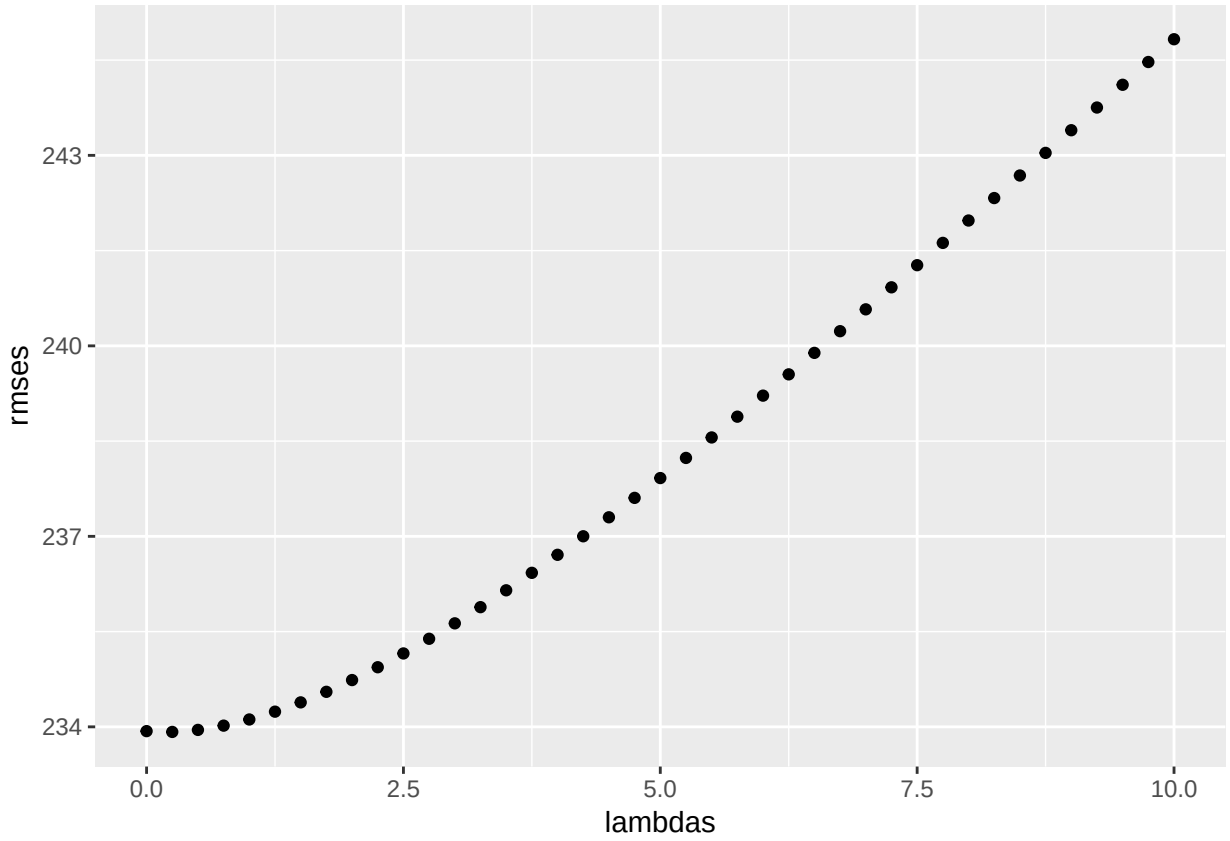
3.1.3. Add extra test data

Our initial premise was to look for interesting suggestions that are in tune with a particular musical taste. In order to do that, we add 4 test records to the training set: a user 9999 that has played Block Party, Incubus, Red Hot Chili Peppers and Arctic Monkeys in equal amount.

```
trainx <- rbind(trainx, data.frame(
  userID = c(9999, 9999, 9999, 9999),
  artistID = c(210, 1116, 220, 207),
  weight = c(1000, 1000, 1000, 1000)
))
```

3.1.4. Optimize the lambda

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(trainx$weight)

  b_i <- trainx %>%
    group_by(artistID) %>%
    summarize(b_i = sum(weight - mu)/(n()+l))

  b_u <- trainx %>%
    left_join(b_i, by="artistID") %>%
    group_by(userID) %>%
    summarize(b_u = sum(weight - b_i - mu)/(n()+l))

  predicted_values <-
    trainx %>%
    left_join(b_i, by = "artistID") %>%
    left_join(b_u, by = "userID") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_values, trainx$weight))
})
qplot(lambdas, rmses)
```




```
print(paste("Lambda", lambdas[which.min(rmses)], "offers the lowest RMSE."))
```

```
## [1] "Lambda 0.25 offers the lowest RMSE."
```

3.1.5. Validate the RMSE

To test our algorithm, we use the RMSE

```
lambda <- 0.25
mu <- mean(trainx$weight)

b_i <- trainx %>%
  group_by(artistID) %>%
  summarize(b_i = sum(weight - mu)/(n()+lambda))

b_u <- trainx %>%
  left_join(b_i, by="artistID") %>%
  group_by(userID) %>%
  summarize(b_u = sum(weight - b_i - mu)/(n()+lambda))

predicted_values <-
  validation %>%
  left_join(b_i, by = "artistID") %>%
  left_join(b_u, by = "userID") %>%
  mutate(pred = mu + b_i + b_u) %>%
```

```
  pull(pred)

validation_RMSE <- RMSE(predicted_values, validation$weight)
print(paste("Our algorithm has a RMSE of", validation_RMSE))
```

```
## [1] "Our algorithm has a RMSE of 246.20279238795"
```

An RMSE of 246 means that our predictions of number of plays typically has an error of 246. This is not necessarily a bad thing since, in practice, what we want is not the number but an order of preference for a particular user.

3.1.5. Making Predictions

We can now predict the number of plays a user will have on the artists and sort it from highest to lowest.

```
suggestions <- merge(data.frame(userID = 9999), artists, all=TRUE) %>%
  left_join(b_i, by = "artistID") %>%
  left_join(b_u, by = "userID") %>%
  semi_join(trainx, by = "artistID") %>%
  mutate(pred = mu + b_i + b_u, artistName = stringr::str_trunc(artistName, 40))
suggestions %>%
  select(artistID, artistName, pred) %>%
  arrange(desc(pred)) %>% top_n(10, pred)
```

```
##    artistID         artistName     pred
## 1      2102  xìng tián lái wèi 1245.954
## 2       375       bāng qíayumi 1222.830
## 3       289     Britney Spears 1218.410
## 4      2044    Sarah Brightman 1203.162
## 5       292 Christina Aguilera 1177.762
## 6      3616          Brand New 1169.534
## 7       813     As I Lay Dying 1163.122
## 8       816  A Day to Remember 1159.712
## 9       152     Porcupine Tree 1137.620
## 10     3324 ān shì nài měi huì 1125.912
```

Finally, we see that although our estimated RSME isn't abysmal, our algorithm would produce the same popular recommendations to every single user. This is because, although we account for user bias, we don't account for tendencies and tastes within groups of people. An easy way to understand the simplistic approach to user bias is to change the b_u and see the results.

```
suggestions <- merge(data.frame(userID = 9999), artists, all=TRUE) %>%
  left_join(b_i, by = "artistID") %>%
  semi_join(trainx, by = "artistID") %>%
  mutate(pred = mu + b_i + 1, artistName = stringr::str_trunc(artistName, 40))
suggestions %>%
  select(artistID, artistName, pred) %>%
  arrange(desc(pred)) %>% top_n(10, pred)
```

```
##    artistID         artistName     pred
## 1      2102  xìng tián lái wèi 718.9775
```

```
## 2       375       bāng qíayumi 695.8537
## 3       289     Britney Spears 691.4341
## 4      2044    Sarah Brightman 676.1859
## 5       292 Christina Aguilera 650.7860
## 6      3616          Brand New 642.5576
## 7       813     As I Lay Dying 636.1457
## 8       816  A Day to Remember 632.7355
## 9       152     Porcupine Tree 610.6436
## 10     3324 ān shì nài měi huì 598.9361
```

In our new test, we change our b_u to 1, and got different prediction values, but the same artists as the difference between them stays proportional.

In order to account for similar user tendencies, we need a different approach.

3.2. Collaborative Filtering

We are working with the assumption that if a user has played music from a specific artist it is because they like it. So the similarities in listening habits allow us to assume similar taste in music. Let's say our data shows that user A listens to music from artists P, Q and R, and user B listens to music from artists P, Q, R and Z. Then it is likely that user A might also enjoy artist Z.

This approach to the recommendation system is called User Based Collaborative Filtering (UBCF). The algorithm's goal is to search the data for similarities to the target user, then calculate the closest matches, these are called nearest neighbors, and use their results to weight the predictions.

R offers easy access to a series of classes and methods to work with UBCF and other algorithms in the form of the recommenderlab package.

3.2.1 Preparing the data

Let's add the users 9998 and 9999 to our plays dataset.

User 9998 likes The Eagles, The Beatles and Led Zepellin. User 9999 likes Bloc Party, Incubus, Red Hot Chili Peppers and Arctic Monkeys.

```
plays <- rbind(plays, data.frame(
  userID = c(9998, 9998, 9998),
  artistID = c(730, 227, 1242),
  weight = c(1000, 1000, 1000)
))
plays <- rbind(plays, data.frame(
  userID = c(9999, 9999, 9999, 9999),
  artistID = c(210, 1116, 220, 207),
  weight = c(1000, 1000, 1000, 1000)
))
```

Note: if you are checking the code, it would be interesting for you to change user 9998 to your particular taste and see the results

In order to work with the Recommenderlab framework, we need to transform the data.frame to a realRatingMatrix class

```
rrm <- as(plays,"realRatingMatrix")
rrm
```

```
## 1865 x 638 rating matrix of class 'realRatingMatrix' with 49619 ratings.
```

3.2.2 Validating the algorithm

Instead of manually setting a train and test sets, the Recommenderlab framework allows us to setup an evaluationScheme and evaluation classes that do the partition and testing.

```
srrm <- rrm[rowCounts(rrm) >= 5,]
es <- evaluationScheme(srrm, method="split", train=0.9, given=3, goodRating=0.001)
ev <- evaluate(es, "UBCF", parameter = list(nn=750), type="ratings",
               n=10, progress=FALSE)
avg(ev)
```

```
##         RMSE      MSE      MAE
## res 306.4374 93903.88 236.0866
```

Our RMSE is worst! Have we failed?

3.2.3 Looking at the results

Let's look at the actual suggestions from the algorithm

```
rec <- Recommender(rrm, method = "UBCF", parameter = list(nn=750))
pre <- predict(rec, rrm["9999"], n=10)
data.frame(artistID = as.integer(as(pre, "list")[[1]])) %>%
  left_join (artists, by="artistID")
```

```
##    artistID                 artistName
## 1       195                Bright Eyes
## 2      1105            Jack's Mannequin
## 3      1510 Black Rebel Motorcycle Club
## 4      1512     The Last Shadow Puppets
## 5      1976              The Black Keys
## 6      2524                  Mando Diao
## 7      3400                         Jet
## 8      3616                   Brand New
## 9      6453           The Academy Is...
## 10     7324           Empire of the Sun
```

In the results section there will be a more detailed explanation, but the conclusion is that these are very good recommendations.

Let's see the suggestions for user 9998

```
pre <- predict(rec, rrm["9998"], n=10)
data.frame(artistID = as.integer(as(pre, "list")[[1]])) %>%
  left_join (artists, by="artistID")
```

```
##    artistID                 artistName
## 1       612               Talking Heads
## 2       733                  John Lennon
## 3      1105            Jack's Mannequin
## 4      1414              Paul McCartney
## 5      1416             George Harrison
```

```
## 6      1510 Black Rebel Motorcycle Club
## 7      1707              Dead Can Dance
## 8      1755               Chico Buarque
## 9      2139                 Beach House
## 10     3400                         Jet
```

There are some similarities, but the differences account for the user's taste in music.

So what Happened? Why is it that a higher RMSE gives such better results? The RMSE is a value that calculates the typical mistake when performing a prediction, doesn't matter if it's positive or negative, also it doesn't really have to be special for each user, an average could, as we saw, produce a better RMSE. So maybe the RMSE is not a good indicator, or maybe it's incomplete.

Let's look at other statistical measures.

3.2.4 Statistical measures

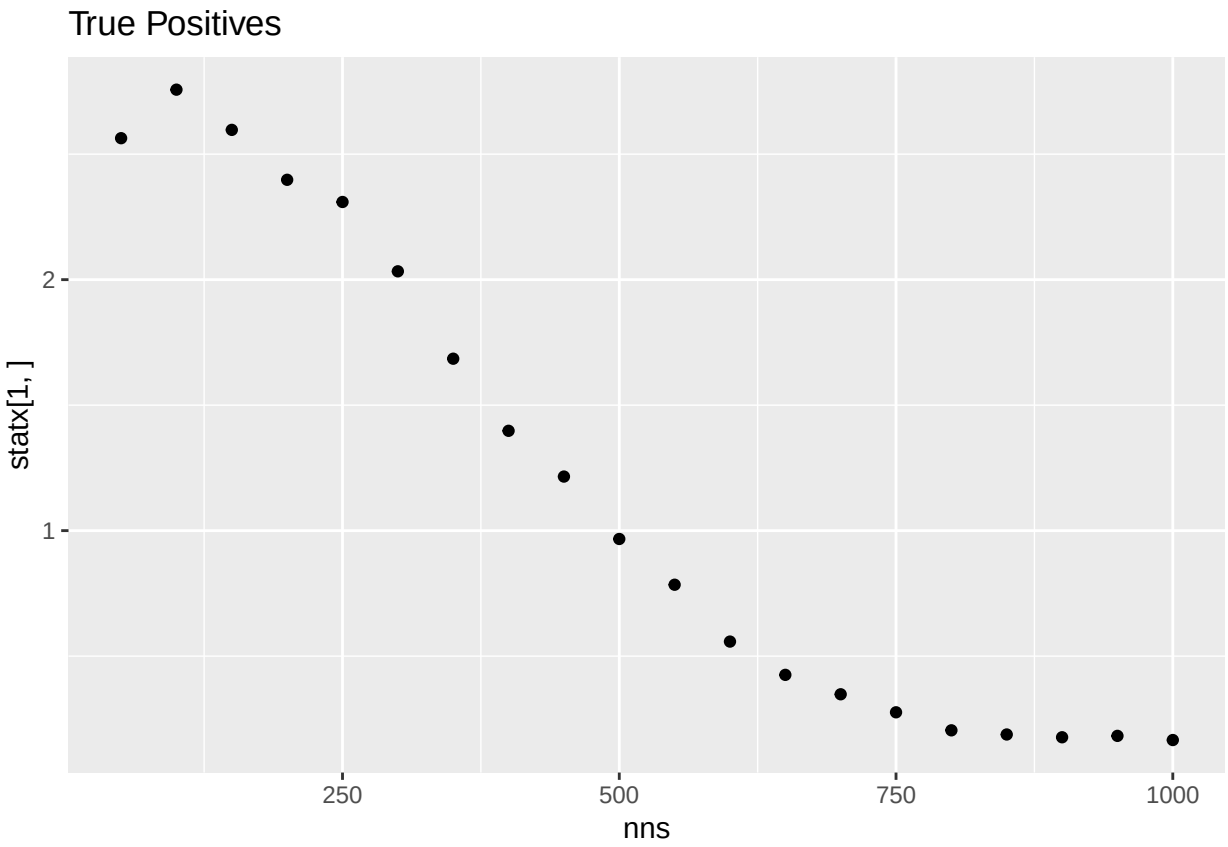
When evaluating an algorithm, other very important statistical measures are the sensitivity and specificity. Those two are based on the four key statistics: True Positive (TP), False Positive (FP), False Negative (FN), True Negative (TN).

By looking at these statistics we can analyze how they are important for our specific goal:

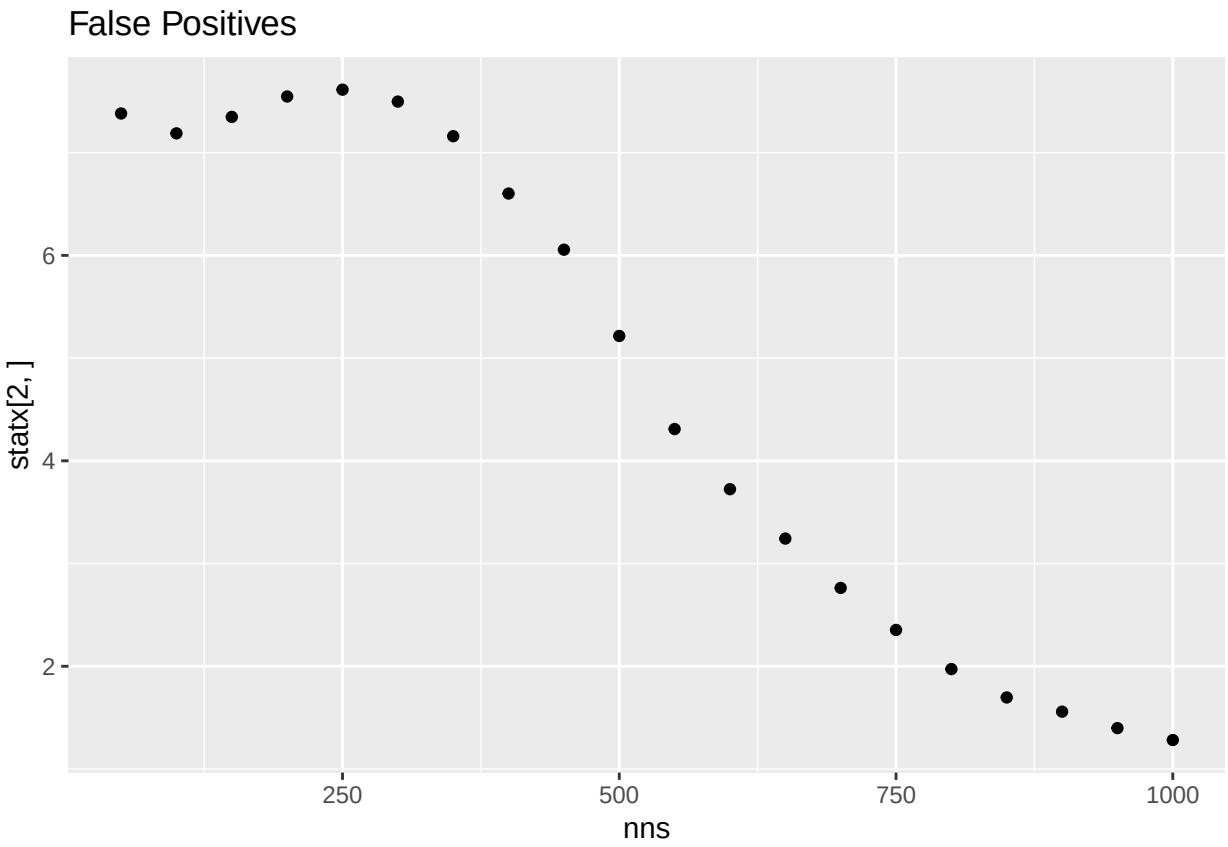
- True Positives: a true positive is a correct suggestion, we should maximize this.
- False Positives: a false positive might lead to a bad suggestion, but if you think about it, because of people's music listening habits, there might be a lot of good suggestions that would read as false positives, since they are not in their playlists.
- False Negatives: a false negative would deny us of a meaningful suggestion, we should minimize this.
- True Negatives: A true negative allows us to discard a suggestion as useful, without them every artist would be suggested. However, we cannot maximize this since it would also increase the false negatives, denying the user from any suggestion.

In a somewhat related note. Something that may have gone unnoticed if the reader is not paying close attention to the code is that there is a hard coded parameter called nn, set at 750. Let's run a series of nn's and see our statistics.

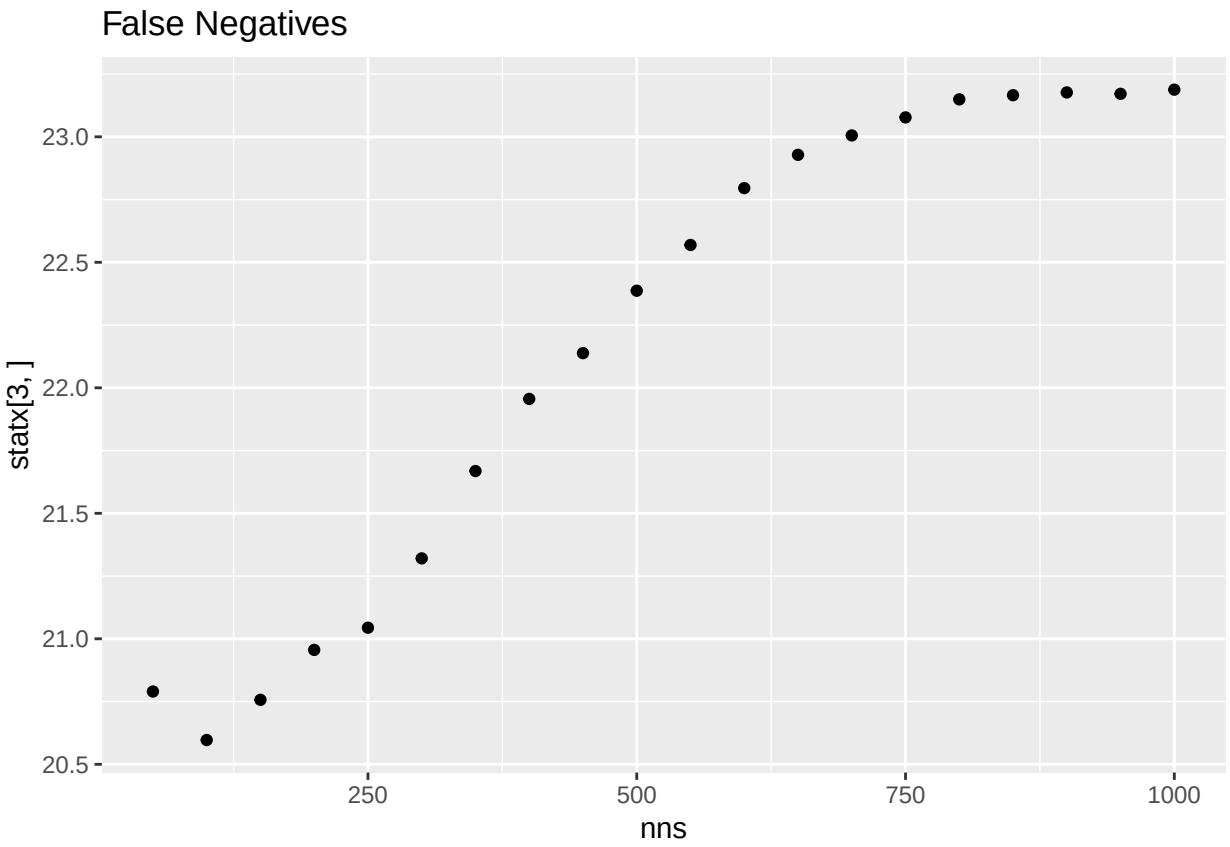
```
nns <- seq(50, 1000, 50)
statx <- sapply(nns, function(nnx){
  ev <- evaluate(es, "UBCF", parameter = list(nn=nnx), n=10,
                 progress = FALSE)
  return(avg(ev))
})
qplot(nns, statx[1,], main="True Positives") #TP
```



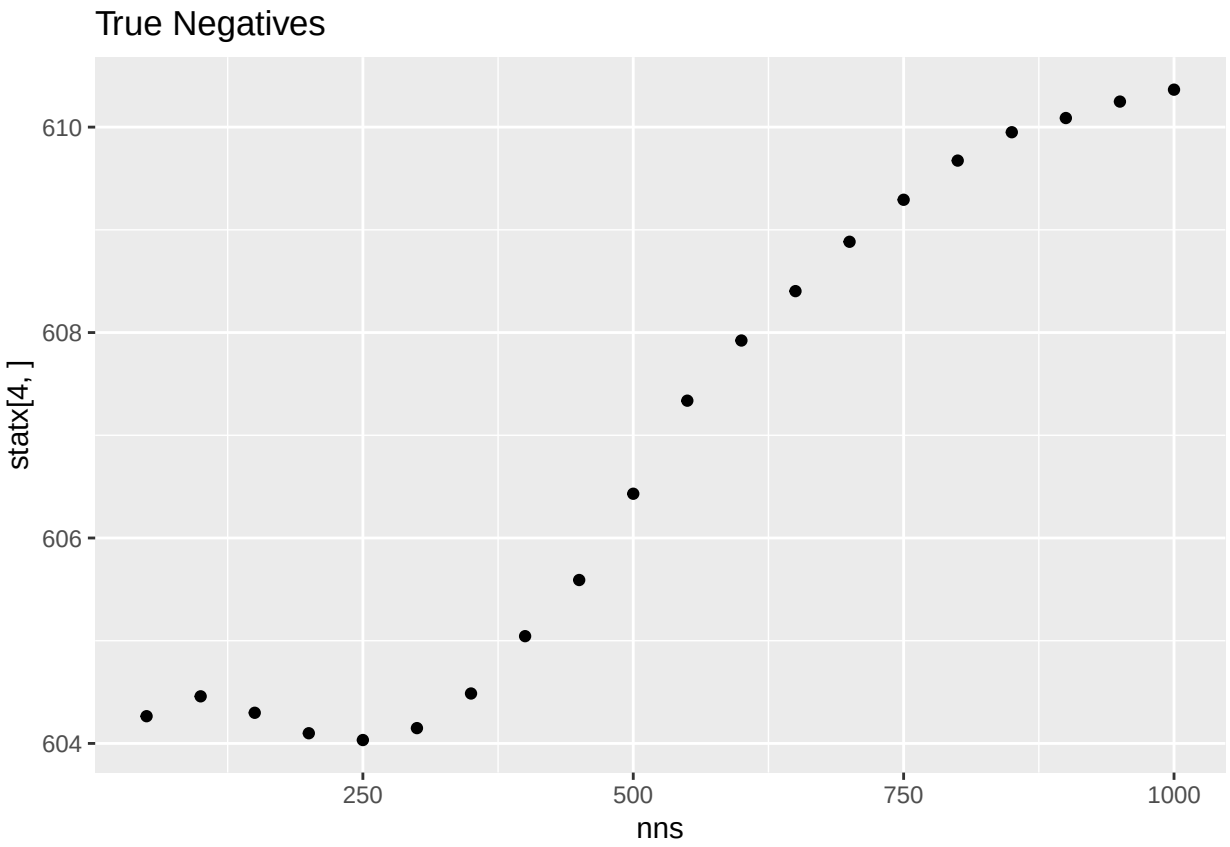

```
qplot(nns, statx[2,], main="False Positives") #FP
```



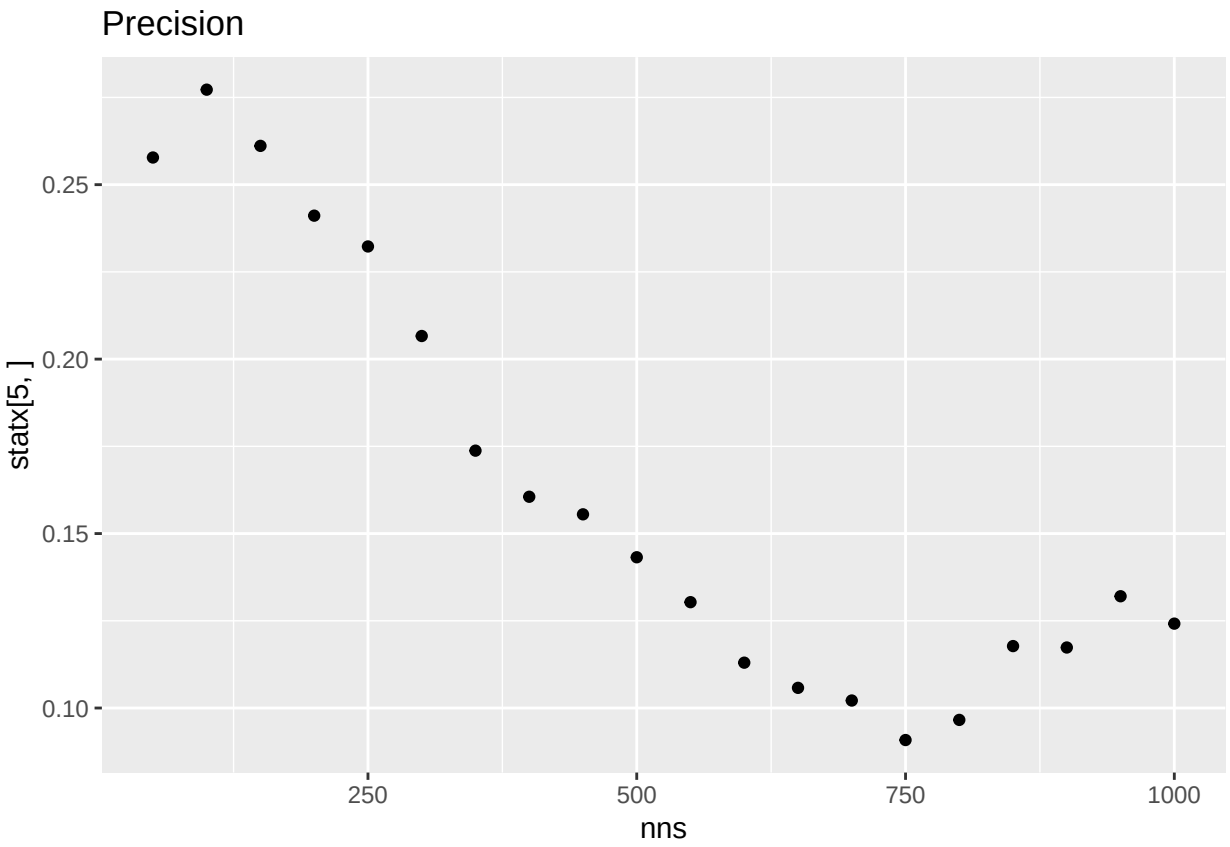

```
qplot(nns, statx[3,], main="False Negatives") #FN
```



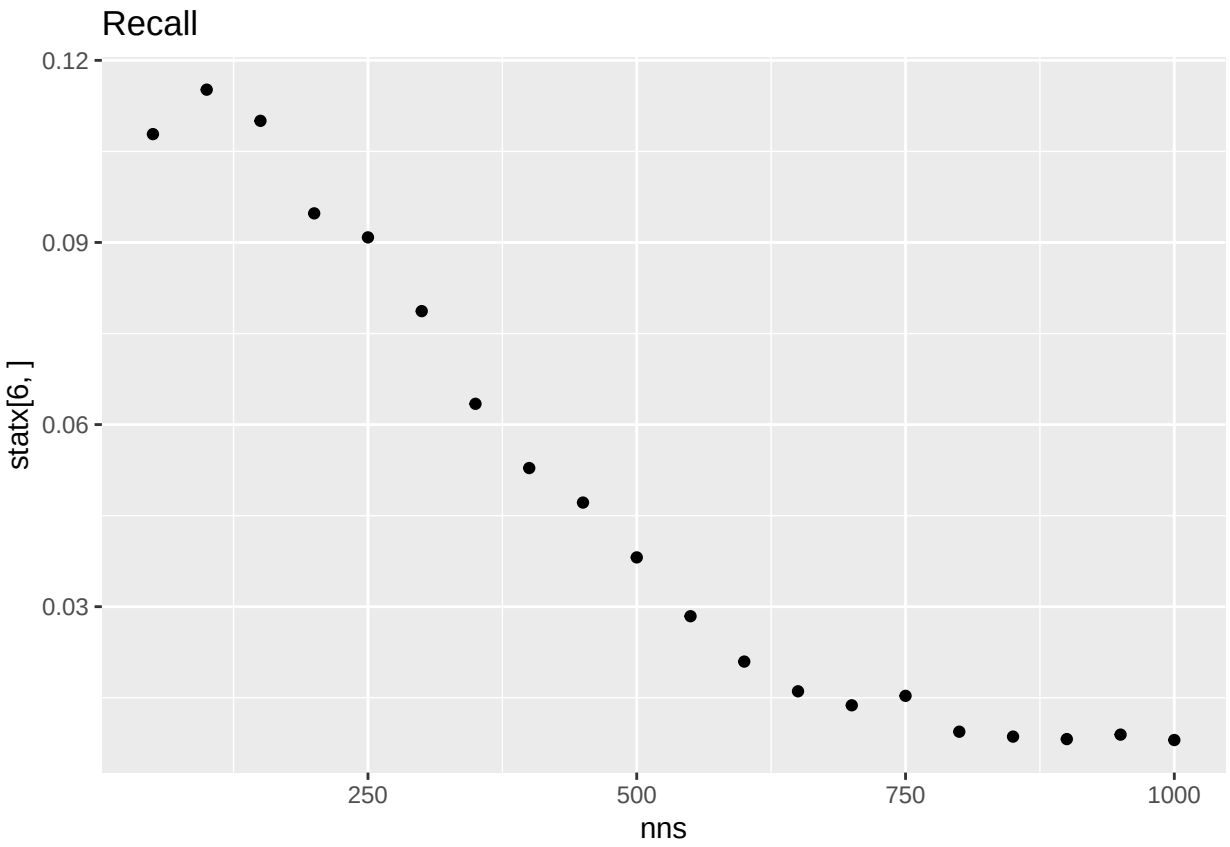

```
qplot(nns, statx[4,], main="True Negatives") #TN
```



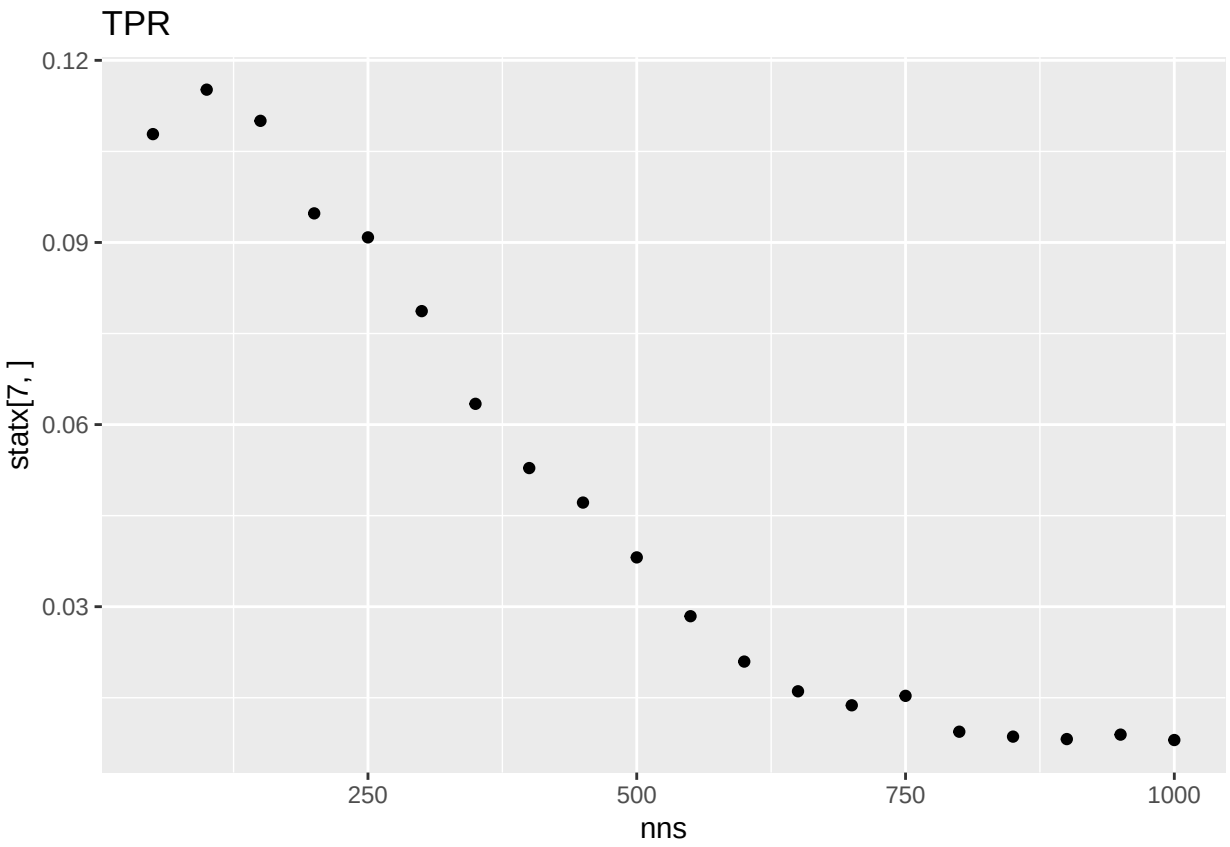

```
qplot(nns, statx[5,], main="Precision") #precision
```



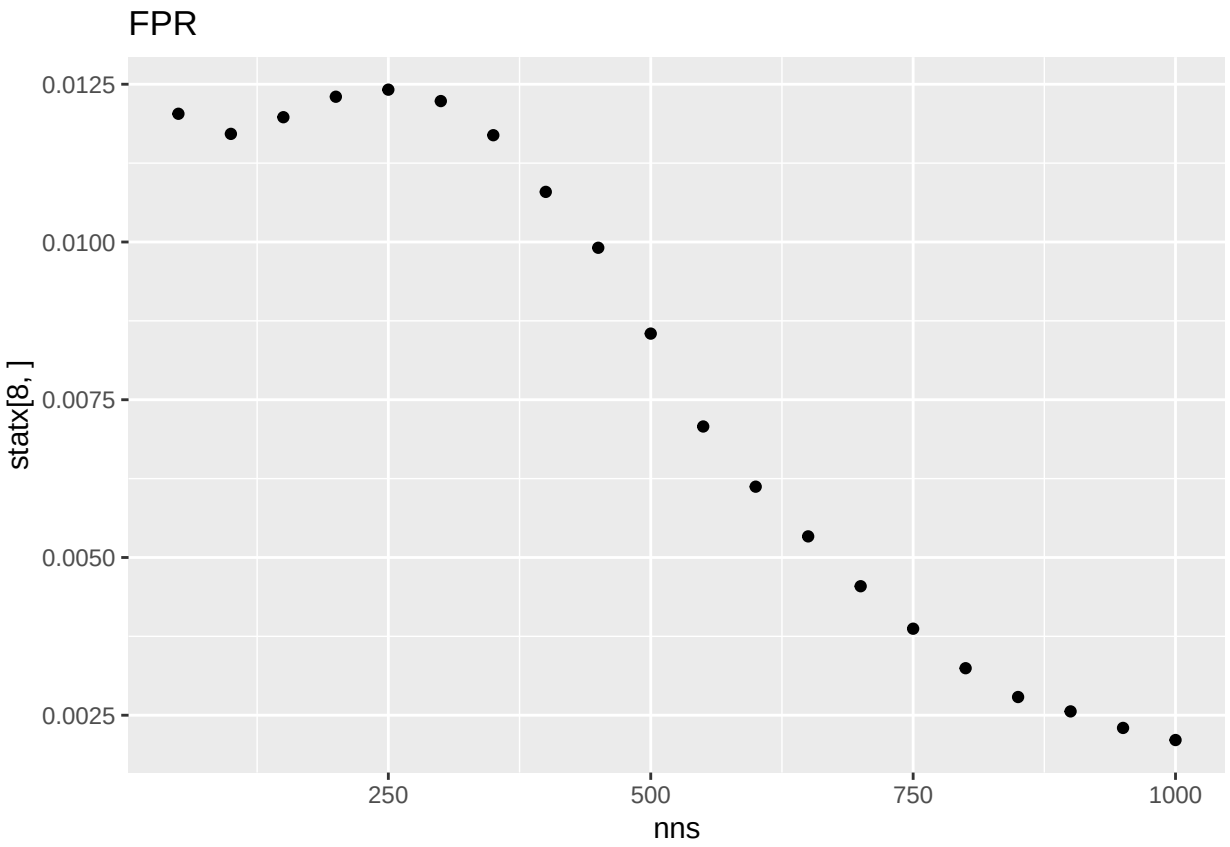

```
qplot(nns, statx[6,], main="Recall") #recall
```



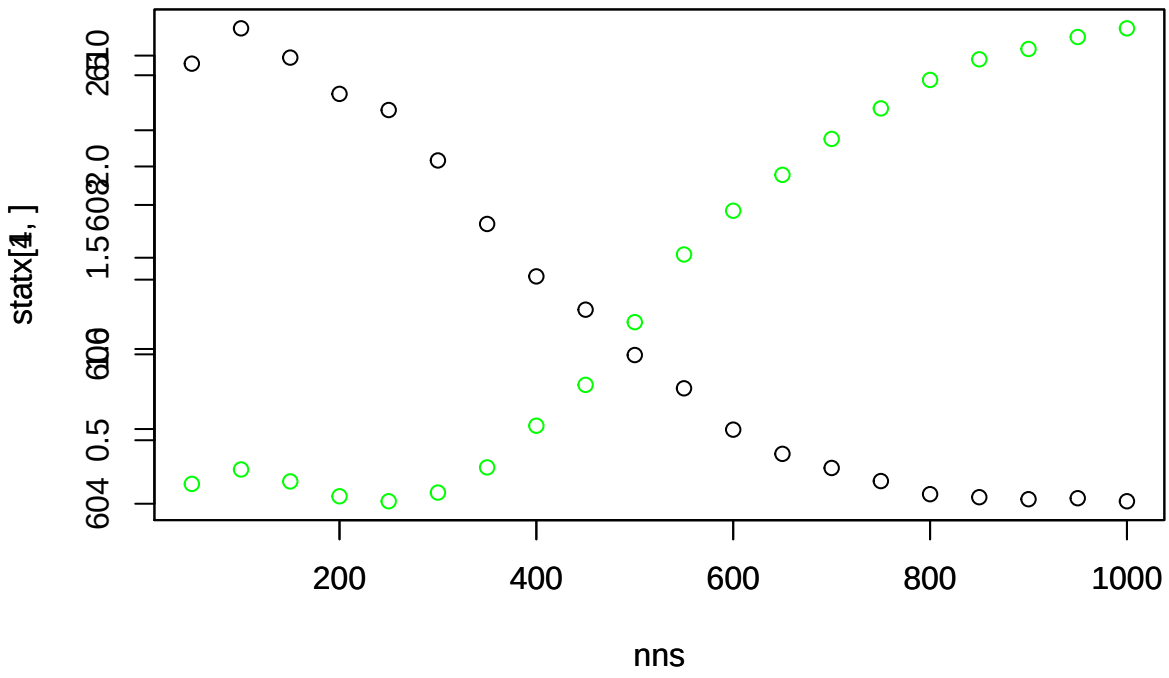

```
qplot(nns, statx[7,], main="TPR") #TPR
```




```
qplot(nns, statx[8,], main="FPR") #FPR
```




```
plot(nns, statx[1,])
par(new=TRUE)
plot(nns, statx[4,],col="green")
```



In this chart, we see the true positives go down as the true negatives go up. They balance each other at close to 500.

When we analyze the suggestions, we get that values lower than 500 tend to be more general and make popular suggestions, while values higher than 500 are more specific but more prone to false negatives.

Let's run the comparison with nn=300

```
rec <- Recommender(rrm, method = "UBCF", parameter = list(nn=300))
pre <- predict(rec, rrm["9999"], n=10)
data.frame(artistID = as.integer(as(pre, "list")[[1]])) %>%
  left_join (artists, by="artistID")
```

```
##    artistID              artistName
## 1         7          Marilyn Manson
## 2        15            Dimmu Borgir
## 3        25         Cradle of Filth
## 4        30                 And One
## 5        45 Mindless Self Indulgence
## 6        51             Duran Duran
## 7        53                     Air
## 8        55           Kylie Minogue
## 9        56               Daft Punk
## 10       58               Goldfrapp
```

```
pre <- predict(rec, rrm["9998"], n=10)
data.frame(artistID = as.integer(as(pre, "list")[[1]])) %>%
  left_join (artists, by="artistID")
```

```
##    artistID              artistName
## 1         7           Marilyn Manson
## 2        15             Dimmu Borgir
## 3        25          Cradle of Filth
## 4        30                  And One
## 5        45 Mindless Self Indulgence
## 6        51              Duran Duran
## 7        53                      Air
## 8        55            Kylie Minogue
## 9        56                Daft Punk
## 10       58                Goldfrapp
```

Both datasets are the same and not that specific to their particular taste.

4. Results

Since the nn parameter actually changes from user to user, I decided to create a function to get suggestions for a specific user, by maximizing the nn until before you get no results.

```
get_suggestions <- function(id) {
  nnx = 400
  resp = NULL
  while (nnx<2000) {
    rec <- Recommender(rrm, method = "UBCF", parameter = list(nn=nnx))
    pre <- predict(rec, rrm[ toString(id) ], n=10)
    if (length(as(pre, "list")[[1]])<10 || nnx>=1950){
      return(data.frame(artistID = as.integer(as(resp, "list")[[1]])) %>%
        left_join (artists, by="artistID"))
    } else {
      resp = pre
    }
    nnx=nnx+50
  }
}
get_suggestions(9999)
```

```
##    artistID                  artistName
## 1       195                Bright Eyes
## 2      1105           Jack's Mannequin
## 3      1510 Black Rebel Motorcycle Club
## 4      1512     The Last Shadow Puppets
## 5      1976              The Black Keys
## 6      2524                  Mando Diao
## 7      3400                         Jet
## 8      3616                   Brand New
## 9      6453           The Academy Is...
## 10     7324           Empire of the Sun
```

Bright Eyes was a bit too slow for my taste, but not that far off. Jack's Mannequin has a pop/rock angsty tone that reminds me of the songs I listened in the 2000s. Black Rebel Motorcycle Club has that indie rock feel that I really enjoy. The last Shadow Puppets is a rock duo from the lead singer of Arctic Monkeys.

These suggestions are awesome!

```
get_suggestions(9998)
```

```
##    artistID                  artistName
## 1       612                Talking Heads
## 2       733                  John Lennon
## 3      1105             Jack's Mannequin
## 4      1414               Paul McCartney
## 5      1416              George Harrison
## 6      1510 Black Rebel Motorcycle Club
## 7      1707               Dead Can Dance
## 8      1755                Chico Buarque
## 9      2139                  Beach House
## 10     3400                          Jet
```

There is some overlap, but we can account for that considering that there is a soft rock vibe going through both of them. The interesting thing to notice is that the differences strongly suggest better suggestions to account for this user's taste in music, like how the members of the Beatles are there.

These suggestions rock too!

5. Conclusions

We started this project with the objective of getting suggestions for music related to my particular taste and the objective was accomplished both in theory but also by actually looking up the bands and enjoying their music.

Future work:

- The original dataset used for this work contains information related to user defined tags and friendships between users. That information clearly defines similarities between artists and between users respectively. Used properly it should allow a much more precise algorithm.
- The original Last.fm database provides an API to get the information. It is where the dataset originally came from and could be used to get much more detailed and bigger dataset.
- In relation to the previous observation. One of the biggest improvements would be to work on a song level rather than an artist level. There has been work in that area and there is a database of similarities between songs. http://millionsongdataset.com/lastfm/
- The Recommenderlab framework has implemented several other algorithms that could improve on the solution, for example, the Item Based Collaborative Filtering (IBCF), that works on similarities, like UBCF, but not between users, but in this case, between artists. This is particularly powerful when we consider that Recommenderlab also allows for Hybrid solutions that mix multiple algorithms to give one weighted result.

6. Bibliography

- Irizarry, Rafael A. "Data Analysis and Prediction Algorithms with R." Introduction to Data Science, 22 Apr. 2019, rafalab.github.io/dsbook/.

- Hahsler, Michael. Recommenderlab: A Framework for Developing and Testing Recommendation Algorithms.

- Luo, Shuyu. "Intro to Recommender System: Collaborative Filtering." Towards Data Science, Towards Data Science, 10 Dec. 2018, towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26. Accessed 17 June 2019.

- Pinela, Carlos. "Recommender Systems — User-Based and Item-Based Collaborative Filtering." Medium, Medium, 6 Nov. 2017, medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f. Accessed 17 June 2019.

Version

This document was created using R

```
version
```

```
##                _
## platform       x86_64-pc-linux-gnu
## arch           x86_64
## os             linux-gnu
## system         x86_64, linux-gnu
## status
## major          3
## minor          4.4
## year           2018
## month          03
## day            15
## svn rev        74408
## language       R
## version.string R version 3.4.4 (2018-03-15)
## nickname       Someone to Lean On
```