Final Project Report

Mengli Hua
Southern Methodist University
6219 Shady Brook Ln.,
Dallas, Texas 75206
2143340070
mhua@smu.edu

ABSTRACT

As users' demands for personalized information continue to increase, recommendation systems are widely used in a variety of fields, including movies, music, news, books, research articles, even for general products. The mainstream recommendation systems are mainly Collaborative filtering, Content-based filtering and Hybrid recommender systems. For large-scale data samples, the performance of different recommended algorithms may differ. Finding the recommended algorithm that performs optimally when dealing with large-scale data helps improve recommendation efficiency and better serve users.

In this project, I implemented two different types of recommendation algorithms (Collaborative filtering and Content-based filtering) via package RNeo4j.R on a large-scale dataset, Million Song Subset, which contains "additional files" (SQLite databases) in the same format as those for the full set but referring only to the 10K song subset, to establish and visualized part of the recommender system. This project has also compared the predicting performance (RMSE) on large-scale dataset of User-based collaborative filtering algorithm and Content-based filtering algorithm. The recommendation results will be shown as tables and charts.

Keywords

Songs recommender system; Performance comparing; RMES; Neo4j

1. INTRODUCTION

With the rapid development of today's technology, the amount of data is increasing day by day, and people are increasingly feeling that they are at a loss in the face of massive data. In order to solve the problem of information overload, people proposed a recommendation system (corresponding to the search engine, people used to call the recommendation system as the recommendation engine). The long tail theory (people only pay attention to projects with high exposure and ignore projects with low exposure) can explain the existence of the recommendation system very well. The experiment shows that the project with low exposure at the long tail position produces profits no less than only the profit of projects with high sales exposure. The recommendation system can give all projects the opportunity to expose, in order to tap the potential profit of the long tail project. Therefore, finding a suitable high-performance recommendation system for data is more worthy of attention.

Recommender systems might be evaluated against various aspects of a recommender system, namely, functional and non-functional.[1] For the recommendation system is a sub-category of the information filtering system designed to predict the user's rating or preference for the specific product or information, functional aspects are typically the performance accuracy of utilized algorithms. It includes such accuracy metrics as predictive accuracy metrics measuring how accurate the recommender system predicts ratings or recommends top-N lists of items. In other words, accuracy is one of the crucial indicators we should keep in mind when talking about the performance of a recommender system.

In this paper, we will compare the performance (mainly accuracy indicators) of different algorithms by calculating the RMSE (Root Mean Squared Error), to find out for large-scale dataset, which algorithm will perform better on accuracy.

The structure of the full-text content is arranged as follows: section2 as the background mainly introduces the relevant background knowledge of the recommended algorithm and data set used; section3 mainly expounds the theoretical knowledge of the recommended algorithm and performance (accuracy) evaluation method, such as formula; Section 4 is a description of the specific process of the experimental implementation, covering the three parts of data processing, modeling and performance evaluation; the last part is the conclusion of the experiment and the summary of the full text, and proposes what can be further improved in future research.

2. BACKGROUND

This experiment mainly tried to compare two common recommendation algorithms, Collaborative filtering and Content-based filtering.

2.1 Background of Algorithms

2.1.1 Collaborative Filtering

Collaborative filtering algorithm[2], the principle is that users like products that users with similar interests like, such as your friends like the movie Harry Potter I, then it will recommend it to you, this is the simplest user-based collaborative filtering algorithm (User-based collaborative filtering), there is also an item-based collaborative filtering algorithm, which is to read all the user's data into memory for operation, so it is called Memory-based Collaborative Filtering[3], and the other is Model-based collaborative filtering, including Aspect Model, pLSA, LDA, clustering, SVD, Matrix Factorization, etc. This method has a long training process, but after the training is completed, the recommendation process is faster[4]. In this experiment, I chose user-based as the representative because for the dataset, there is not any general genre information of songs, so it is not so suitable to establish an item-based CF.

2.1.2 Content-based Filtering

The content-based[5] recommendation algorithm is based on the fact that users like items that are similar in content to the items they have been interested in. For example, you saw Harry Potter I, the content-based recommendation algorithm found Harry Potter II-VI, and you used to the content of the content (a lot of keywords) is very relevant, so I recommend it to you. This method can avoid the cold start of the item (cold start[6]: if an item has never been noticed,

other recommendations The algorithm rarely recommends, but the content-based recommendation algorithm can analyze the relationship between items and implement the recommendation); the drawback is that the recommended items may be repeated, and for some multimedia recommendations (such as music, movies, pictures, etc.) Since it is difficult to mention content features, it is difficult to recommend[7]. This time I chose the artist as the casement to define similar songs for songs from the same artist may more likely to be liked than other songs.

2.2 Background of Dataset

The dataset used in this experiment is The Million Song Database, or MSD[8], is an open source dataset of one million popular songs made freely available by The Echo Nest. Million Songs Dataset is a mixture of song from various website. It is a freely usable collection of one million contemporary pop music tracks for audio features and metadata.

The purpose is:

- Encourage research on algorithms that scale to commercial scale;
- Provide reference data sets for evaluation studies;
- As a shortcut to creating large data sets using the API (eg Echo Nest);
- Help new researchers get started in the MIR field;

The core of the data set is the feature analysis and metadata of a million songs. This data set does not contain any audio and only contains derived features. Sample audio can be obtained from services such as 7digital by using the code provided by Columbia University.

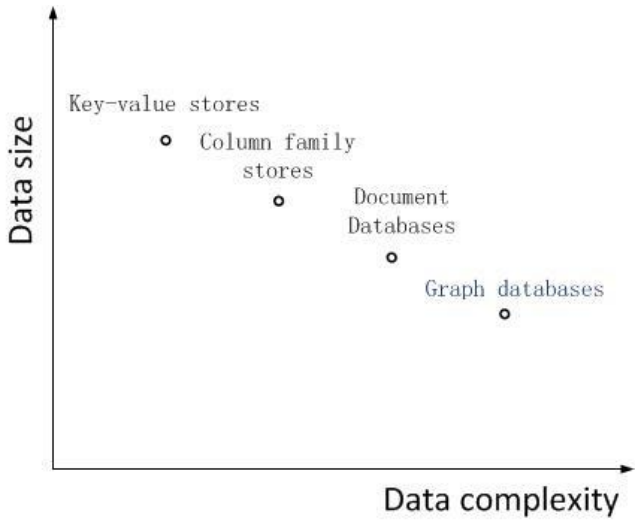
It contains of two files: triplet_file and metadata_file. The triplet_file contains user_id, song_id and listen time. The metadat_file contains song_id, title, release_by and artist_name. I found links to CSV files of both datasets in the Medium article "How to build a simple song recommender system" by Eric Le, which also introduced that the total length of this dataset indexed by song are 2,000,000. In this project, I will use the whole triplet_file and song_id, artist_name from the metadat_file. Because there is no explicit rating, I treat a user's listen count of a song – compared to their total number of song listens – as the user's implicit rating of the song.

2.3 Introduction of Neo4j[26] and RNeo4j.R

2.3.1 Neo4j

A lot of data is expressed by graphs in real life, such as the relationship between people in social networks, map data, or genetic information. RDBMS (Relational Database Management System) is not suitable for expressing such data, and because of the existence of massive data, it is stretched. The rise of NoSQL database has solved the problem of storing huge amounts of data. The graph database is also a branch of NoSQL. Compared with other branches in NoSQL, it is suitable for the data of native expression graph structure.

The next picture shows that compared to other NoSQL, the data stored in the graph database has decreased in size, but it is more able to express complex data.



F 4.1 Comparison of Different Types NoSQL Databases

In general, a graph database stores a structure like a graph in a data structure, consisting of vertices and edges.

Neo4j is a major representative of the graph database, open source, and implemented in Java. After several years of development, it can be used in the production environment. It has two modes of operation, one is the service mode, and the external REST interface is provided; the other is the embedded mode, the data is stored locally in the form of a file, and the local file can be directly operated.

Neo4j is getting more and more attention due to its advantages of embedded, high performance and lightweight.

There are two basic data types in a diagram: Nodes and Relationships. Nodes and Relationships contain attributes in the form of keys/values. Nodes are connected by relationships defined by Relationships to form a relational network structure.

From these aspects, Neo4j is a suitable choice. Neo4j:

- Comes with a set of easy-to-learn query language (called Cypher)
- Do not use schema, so you can meet any form of your needs
- Compared to relational databases, queries for highly correlated data (graphic data) are much faster
- Its entity and relationship structure is very natural to the human intuitive feeling
- Support for ACID-compliant transaction operations
- Provides a high availability model to support large-volume data queries, support for backup, data locality, and redundancy
- Provides a visual query console that you won't get bored with

2.3.2 Package RNeo4j.R

RNeo4j is Neo4j's R driver. It allows you to read and write data from / to Neo4j directly from your R environment[27].

3. RELATIVE THEORY

3.1 User-Based Collaborative Filtering[9]

The user-based collaborative filtering algorithm is to discover the user's likes of goods or content (such as product purchase, collection, content comment or sharing) through the user's historical behavior data, and measure and score these preferences. The relationship between users is calculated based on the attitudes and preferences of different users for the same product or content. Product recommendations are made between users who have the same preferences. Therefore, the collaborative filtering algorithm is mainly divided into two steps:

1. Find similar collections of users;

2. Search for recommendations in the collection that the user likes and that the target user does not have.

3.1.1 Calculating Similarity

Here are four common ways to calculate the similarity between users.

3.1.1.1 Jaccard formula[10]

The Jaccard coefficient is mainly used to calculate the similarity between individuals of symbol metrics or Boolean metrics. Because the characteristic attributes of individuals are identified by symbol metrics or Boolean values, it is impossible to measure the size of specific values of differences, and only "whether the same" is obtained. This result, so the Jaccard coefficient only cares about whether the features common to individuals are consistent. If you compare the Jaccard similarity coefficients of X and Y, only compare the same numbers in xn and yn. Defined as follows:

Given two sets A, B, the Jaccard coefficient is defined as the ratio of the size of the intersection of A and B to the size of the union of A and B, which can be shown as:

$$J(A, B)= \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$$

3.1.1.2 Pearson correlation coefficient[11]

The Pearson correlation coefficient is a more complex method of judging the similarity of interest than the Euclidean distance. It tends to give better results when the data is not very standard. Defined as follows:

Given two variables A, B, the Pearson similarity coefficient is defined as the ratio of the A and B covariances to the standard deviation of A and B, which can be seen as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

Here 'cov' means covariances.

3.1.1.3 Euclid distance[12]

Euclid distance is a commonly used distance definition, referring to the true distance between two points in m-dimensional space, or the natural length of the vector (ie the distance from the point to the origin). The Euclidean distance in two-dimensional and three-dimensional space is the actual distance between two points. Defined as follows:

Assuming that two users A and B are both n-dimensional vectors, indicating the user's score for n items, then the Euclidean distance between A and B is:

$$\text{distance(A, B)}= \sqrt{\sum_1^N (x_i - y_i)^2}$$

The smaller the value, the higher the similarity, but for different n, the calculated distance is not easy to control, so the following conversion can be performed:

$$\text{similarity(A, B)}= \frac{1}{1+\sqrt{\sum_1^N (x_i - y_i)^2}}$$

Therefore, the result will be distributed over (0, 1), and the larger the value, the higher the similarity.

3.1.1.4 Cosine distance[13]

The cosine distance, also known as the cosine similarity, is the measure of the difference between two individuals using the two vector cosine values in the vector space. Defined as follows:

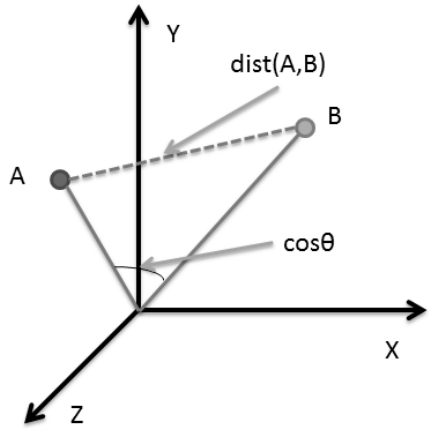
The cosine between the two vectors can be found by using the Euclidean dot product formula:

$$a \cdot b = \|a\|\|b\|\cos\theta$$

Given two attribute vectors, A and B, the remaining string similarity θ is given by the dot product and the vector length as follows:

$$\text{similarity(A, B)}=\cos\theta = \frac{A \cdot B}{\|A\|\|B\|}$$

Compared with the Euclidean distance, the cosine distance pays more attention to the relative difference between the two vectors in the direction, rather than the absolute distance in space. Specifically, we can use the following figure to feel the difference between the two:



F 3.1 Comparison of Cosine similarity and Euclid Similarity

3.1.2 Recommendation Generation

After selecting one of the above methods to obtain the similarity between the users, for the target user *u*, we select the most similar *k* users, and use the set *S(u, k)* to represent the items that all users in *S* like. Extract and remove items that the target user *u* already likes. The remaining items are then scored and similarly weighted, and the results obtained are sorted. Finally, the target user *u* is recommended by the sorting result. Among them, for each item *i* that may be recommended, the degree of interest of the user *u* can be calculated by the following formula[14]:

$$\text{p(u, i)}= \sum_{v \in S(u,k) \cap N(i)} w_{uv} \times r_{u_i}$$

'r_{vi}' represents the degree of preference of user *v* for *i*, that is, the score for *i*, and 'w_{uv}' represents the similarity between users *u* and *v*.

3.2 Content-Based Filtering

Content-based filtering should be regarded as one of the first recommended methods. It is based on the products that the user liked in the past (collectively referred to as item), and recommended products similar to those he/ she liked in the past. cContent-based filtering was mainly used in information retrieval systems, so many methods of information retrieval and information filtering can be used in content-based filtering. The process generally includes the following three steps:

1. Item Representation

Extract some features for each item (that is, the content of the item) to represent this item;

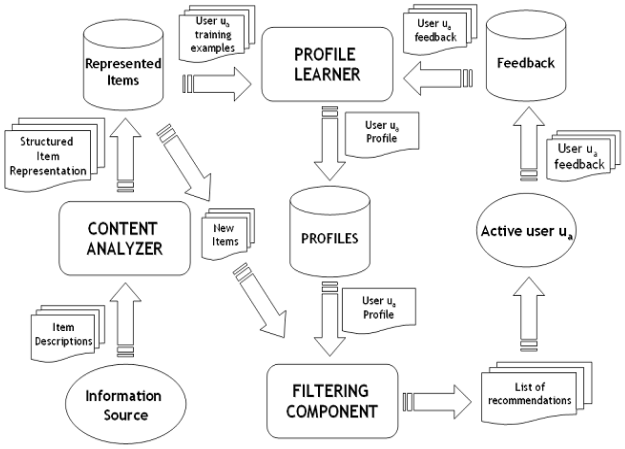
2. Profile Learning

Use the feature data of a user's favorite (and dislike) items in the past to learn the user's favorite profile;

3. Recommendation Generation

By comparing the user profile obtained in the previous step with the characteristics of the candidate item, a set of most relevant items is recommended for this user.

The whole process can refer to the following flow chart first given in [15] (the first step corresponds to Content Analyzer, the second step corresponds to Profile Learner, and the third step corresponds to Filtering Component):



F 3.2 Process of Content-Based Filtering

3.2.1 Item Representation[16]

Set that all the collections we want to represent as $D=\{d_1, d_2, \ldots, d_N\}$, and the collection of all the words that appear in the article (also called the dictionary) is $T=\{ t_1, t_2, \ldots, t_n \}$. In other words, we have *N* articles to process, and these articles contain *n* different words. We end up using a vector to represent an article. For example, the jth article is represented as $d_j=(w_{1j}, w_{2j}, \ldots, w_{nj})$, where w_{1j} represents the weight of the first word t_1 in the article *j*. The interpretation of other vectors in d_j is similar. So, in order to represent the j_{th} article, the key now is how to calculate the values of the components of d_j. For example, we can choose w_{1j} as 1 if the word t_1 appears in the *jth* article; choose 0 if t_1 does not appear in the *jth* article. We can also choose the number of times that w_{1j} appears in the *jth* article. However, the most used calculation method is the term frequency–inverse document frequency (tf-idf) commonly used in information retrieval. The *tf-idf* corresponding to the k_{th} word in the dictionary in the j_{th} article is:

$$\text{TF-IDF}(t_k, d_j) = \text{TF}(t_k, d_j) \cdot \log\frac{N}{n_k}$$

Where $TF(t_k, d_j)$ is the number of occurrences of the k_{th} word in the article *j*, and n_k is the number of articles including the k_{th} word in all articles.

3.2.2 Profile Learning

Suppose user u has given his preference to some items, like some of them, and dislikes the other. Then, this step is to create a model for the user u through these past preferences. With this model, we can use this model to determine if user u would like a new item. Therefore, what we have to solve is a typical supervised classification problem. In theory, the classification algorithm in machine learning can be used here. The following is a brief introduction to some of the learning algorithms commonly used in content-based filtering:

3.2.2.1 Nearest Neighbor Method (k-Nearest Neighbor, kNN[17])

For a new item, the nearest neighbor method first looks for the k items that the user u has judged and most similar to the new item, and then judges the preference of the new item based on the user u's preference for the k items. This approach is similar to the item-based kNN in Collaborative Filtering. The difference is that the item similarity is calculated based on the attribute vector of the item, and the Collaborative Filtering is calculated based on the score of all users on the item.

3.2.2.2 SDecision Tree Algorithm[18] (DT)

Decision trees are generally a good choice when item properties are small and structured. In this case, the decision tree can produce simple, intuitive, and understandable results. And we can show the decision process of the decision tree to the user u, telling him why these items will be recommended. But if the items of the item are more and they are derived from unstructured data (such as item is an article), then the effect of the decision tree may not be very good.

3.2.2.3 Naive Bayes[19] (NB)

The Naive Bayes algorithm is just like its abbreviation. NB is often used for text categorization, which assumes that the probability of occurrence of each word is independent of each other given the category of an article. Although its assumptions are not reliable, its results are often surprisingly good. In addition, NB's code implementation is relatively simple, so it is often the first algorithm to be tried in many classification problems. Our current profile learning problem includes two categories: the item that user u likes, and the item that he doesn't like. Given a category of an item, the probability of its values is independent of each other. We can use the historical preference data of user u to train NB, and then use the trained NB to classify the given item.

3.2.3 Recommendation Generation

If the previous step of Profile Learning uses a classification model (such as DT and NB), then we only need to return the n items that the model predicts the user most likely to be interested as a recommendation to the user. And if the method used in Profile Learning to directly learn user attributes (such as the Rocchio[20] algorithm), then we only need to return the n items most relevant to

the user attributes as recommendations to the user. The correlation between the user attribute and the item attribute can be obtained using a similarity measure such as cosine.

3.3 Performance Evaluation

The evaluation index is a function used to evaluate the performance of a system. It can be divided into a measure of the complexity of the algorithm and a measure of the accuracy of the algorithm. The complexity of the algorithm mainly considers the space and time complexity of the algorithm. Though the complexity of the algorithm is important, the accuracy metric of the algorithm is mainly discussed here.

The recommendation system is usually divided into two categories according to the recommended tasks: score prediction and Top-N list recommendation.

3.3.1 Score Prediction

Predict how many points a particular user can rate for items that have not produced behavior. Scoring predictions are generally calculated by Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). For user u and item i in the test set, r_{ui} is the real score of user u for item i, $\hat{r}_{ui}$ is the score predicted by the recommendation algorithm,

then RMSE:

$$\text{RMSE}=\frac{\sqrt{\Sigma_{u,i\in T}(r_{ui}-\hat{r}_{ui})^2}}{|T|}$$

MAE is:

$$\text{MAE}=\frac{\Sigma_{u,i\in T}(r_{ui}-\hat{r}_{ui})}{|T|}$$

In this experiment I chose this as the indicator to calculate the accuracy of the prediction for RMSE has increased the penalty for rating unacceptable user items (square-squared penalty), making the system more critical[21]. Also, the rating system is not based on integers (the score that users given are calculated by the ratio of their listen count, not all the ratio is an integer), so the method of rounding the prediction result cannot be used to reduce the error of the MAE.

3.3.2 Top-N list recommendation[22]

The score prediction can only be applied to a small part of the scene, such as the score of the movie, the book, in fact, the Top-N recommendation is more in line with the current needs, giving the user a recommended list for selection. Top-N recommendations are generally measured by accuracy and recall. Where *R(u)* is a list of recommendations made to the user based on the user's behavior on the training set (referred to as a list of predicted recommendations), and *T(u)* is a list of behaviors of the user on the test set (refers to the real The list of GroundTruth), in this author is always easy to confuse the meaning of both.

The significance of the accuracy rate is how much of the predicted recommendation list is really interested in the user, that is, the accuracy of the prediction list, then the accuracy rate is defined as[23]:

$$\text{Precision}=\frac{\Sigma_{u\in U}|R(u)\cap T(u)|}{\Sigma_{u\in U}|R(u)|}$$

The significance of the recall rate is how much of the list of real users is accurately predicted by the recommended algorithm, that is, the recall rate of the real list, then the recall rate is defined as[23]:

$$\text{Recall}=\frac{\Sigma_{u\in U}|R(u)\cap T(u)|}{\Sigma_{u\in U}|T(u)|}$$

The two evaluation indicators measure the quality of the recommendation system from different aspects. The two show a negative correlation state, that is, the recall rate tends to be low in the case of high accuracy, and vice versa. Therefore, a *F-score*[24] that combines the accuracy and recall rate is proposed, which makes it easier to observe the quality of the recommendation system. The formula is as follows:

$$\text{F1}=\frac{2PR}{P+R}$$

Where *P* means precision rate and *R* means recall rate.

We can also use *confusion matrix*[25] like following to judge the accuracy.

Table 3.1 Confusion Matrix

Ground Truth	**Prediction Result**	
	Positive case	**negative case**
Positive case	TP	FN
negative case	FP	TN

Here *TP* means positive example has correct prediction results while *FP* means positive example has incorrect prediction results; *FN* represents negative example has incorrect prediction results while *TN* represents negative example has incorrect prediction results.

4. IMPLETATION

In this section, I will use the Million Songs Database to create a song recommendation system based on user-based collaborative filtering and a song recommendation system based on content filtering. I will use the R package RNeo4j to call the graphical database platform Neo4j, where I will train and evaluate the data source and visualize the structure of some recommended systems.

4.1 List of libraries used

*Some libraries may need other libraries which are not mentioned in this list as prerequisites.

Table 4.1 List of Packages

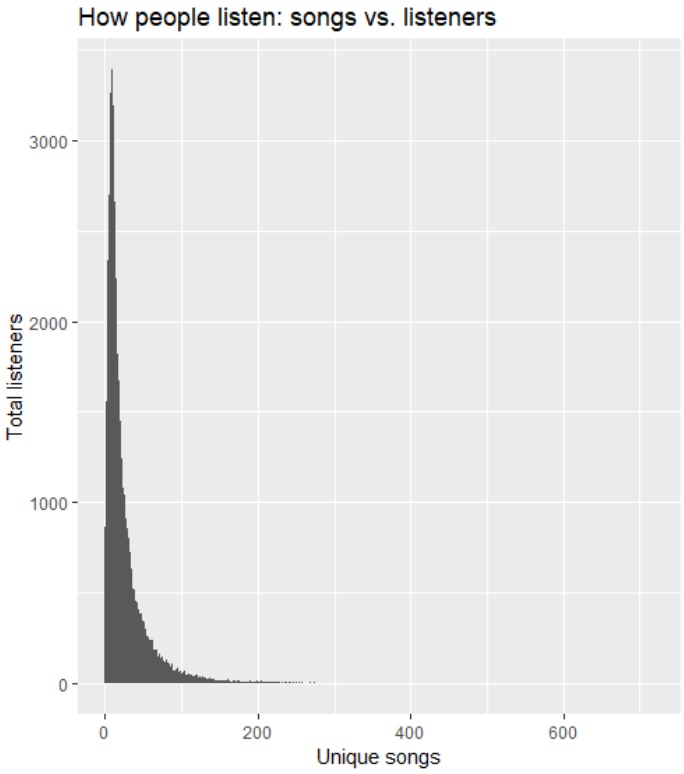
Library Name	Description
ggplot2[29]	A system for 'declaratively' creating graphics, based on "The Grammar of Graphics".
recommenderlab[30]	Provides a research infrastructure to test and develop recommender algorithms including UBCF, IBCF, FunkSVD and association rule-based algorithms.
dplyr[31]	A fast, consistent tool for working with data frame like objects, both in memory and out of memory.
magrittr[32]	The magrittr (to be pronounced with a sophisticated french accent) is a package with two aims: to decrease development time and to improve readability and maintainability of code.
stringr[33]	A consistent, simple and easy to use set of wrappers around the fantastic 'stringi' package. All function and argument names (and positions) are consistent, all functions deal with ``NA"'s and zero length vectors in the same way, and the output from one function is easy to feed into the input of another.
tidyr[34]	An evolution of 'reshape2'. It's designed specifically for data tidying (not general reshaping or aggregating) and works well with 'dplyr' data pipelines.
knitr[35]	Provides a general-purpose tool for dynamic report generation in R using Literate Programming techniques.
kableExtra[36]	This package simplifies the way to manipulate the HTML or 'LaTeX' codes generated by 'kable()' and allows users to construct complex tables and customize styles using a readable syntax.
RNeo4j[27]	It allows you to read and write data from / to Neo4j directly from your R environment.

4.2 Data Source

I found links to TXT and CSV files of both datasets in the article "How to build a simple song recommender system"[37] by Eric Le.

4.3 Data Discover

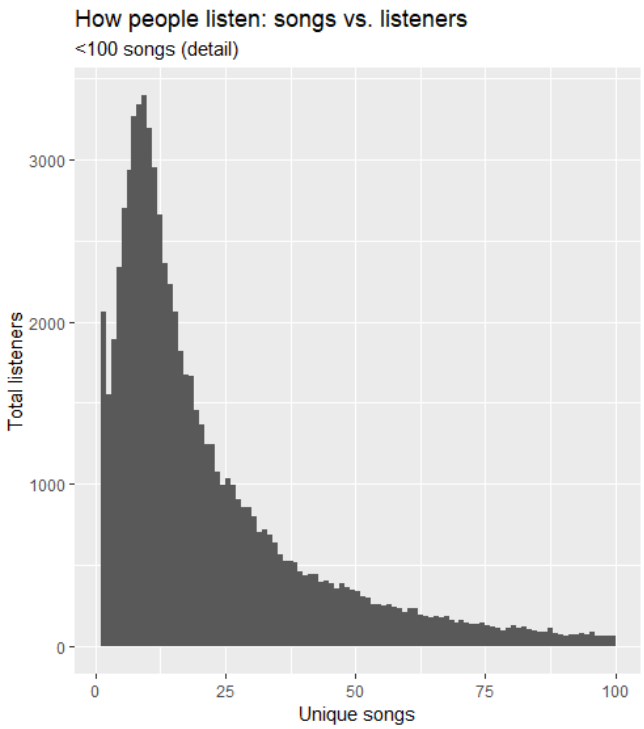
As can be seen after summarized the entire data set, the MSD includes 76,353 user data, and each person has heard at least one song. On average, individuals only listen to more than 16 songs, although there are hundreds of songs with hundreds of playlists. During the data collection period, the individual totally averaged over 81 listens.



F 4.1 Songs Vs. Listeners

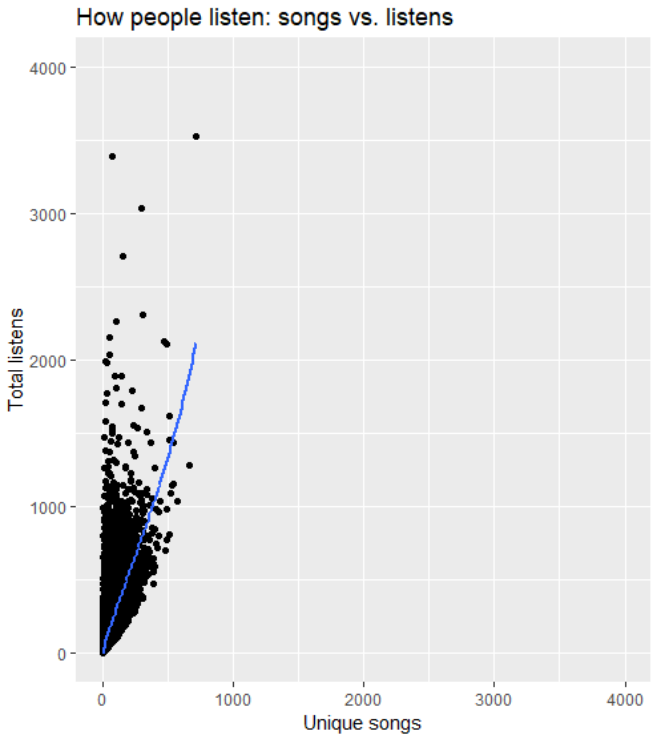
The picture above shows the frequency distribution of the number of songs the individual listens to (unique songs) - depicting the significant tilt of the data set, highlighting a large number of 76,353 people who have heard less than a few songs and listeners who stand on the long tail that have listened a lot of songs for a long time.

The next figure is more detail for the one before. It focuses on the audience of 100 or fewer songs. Before reaching the peak of 10 songs, it shows an increase in the listener of the single song on the far left of the chart. The "flat" listening mode of a single listener does not provide variability that helps to understand preferences.



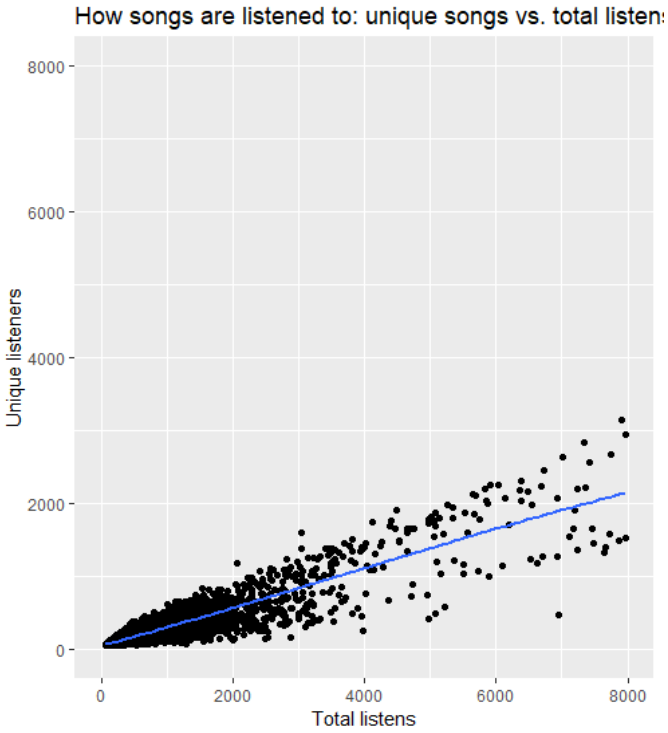
F 4.2 Songs Vs. Listeners(Detail)

A scatter plot that examines the behavior of the listener - the total number of songs the individual listens to on the horizontal axis and the number of times the whole person listens to the song on the vertical axis - we find a wider distribution than it. In other words, listeners tend to listen to fewer songs and show preference. A locally weighted scatter plot smoothing line highlights this general trend.



F 4.3 Songs Vs. Listens

The scale of the chart shows how this works on different spectra, with a dense audience cloud on many levels. For example, although many people have heard more than 500 songs to 100 songs, some people listen to 100 songs each time.



F 4.4 Unique Songs Vs. Total Listens

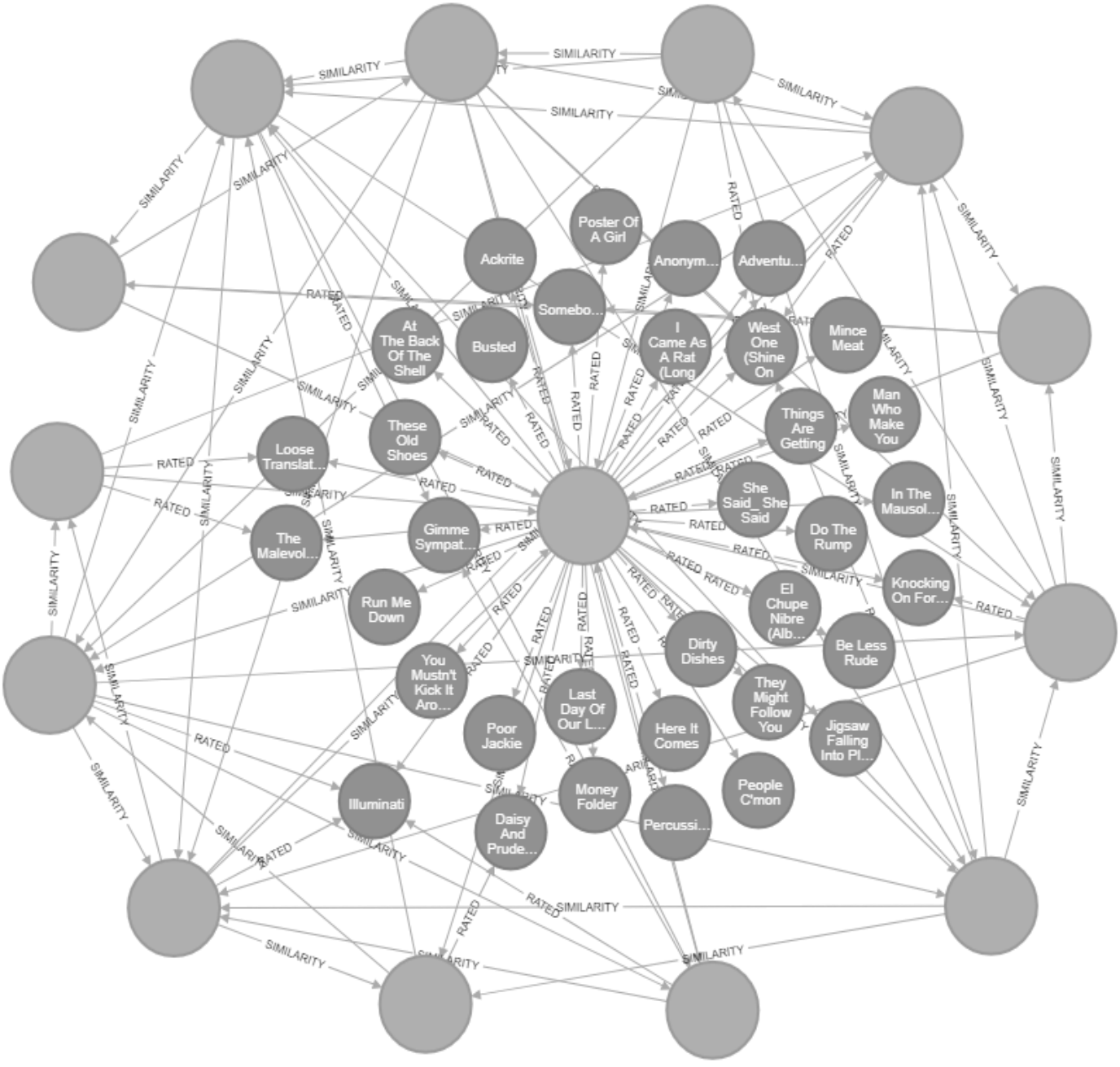
Listening to these millions of songs on average more than 3 times, and the locally weighted scatter plot smoothing line again highlights the songs tending to be repeatedly listened to (ie, preferred) by the same audience.

4.4 Build Recommender System

Thanks for the article “Movie Recommendations with k-Nearest Neighbors and Cosine Similarity”[38] by Nicole White and the RPub by Kavya Beheraj and Jeremy O’Brien[28], I finally successfully build the recommender system with Neo4j used the R environment.

Part of the user-based recommender system can be visualized as Figure 4.5[28].

- The node without text in it are represent users;
- The node within text are songs.
- The links are relationship which can be separated into two classes:
  1. A user’s rating a song (user-song relationship)
  2. A user’s similarity to another user (user-user relationship)



F 4.5 Recommender System Visualization

song_id	title	artist	user_rating	predicted_rating
SOUCHPA12AB0184B1A	Rain	Subhumans	0.84	9.115000
SOFONOE12A58A7C14A	Whatever You Like (Single Version)	Anya Marina	8.37	5.560000
SOCEWMO12A8151CBDE	Foundations	Kate Nash	3.35	5.260000
SOFLJQZ12A6D4FADA6	Tive Sim	Cartola	0.42	5.200000
SOOFYTN12A6D4F9B35	ReprÃ©sente	Alliance Ethnik	1.26	5.043333
SOMCAFM12A58A7B024	Who Can Compare	Foolish Things	1.67	2.970000

F 4.6 User-Based Recommendation

4.5 Recommendation test

To test the performance of the recommender on a small scale, we will take one random user ID and:

- Get the user's real song ratings.
- Get recommended songs that the user has not already listened to based on similarity to other users.

F 4.6 is one time recommendation result of user-based recommender.

4.6 Performance Evaluation

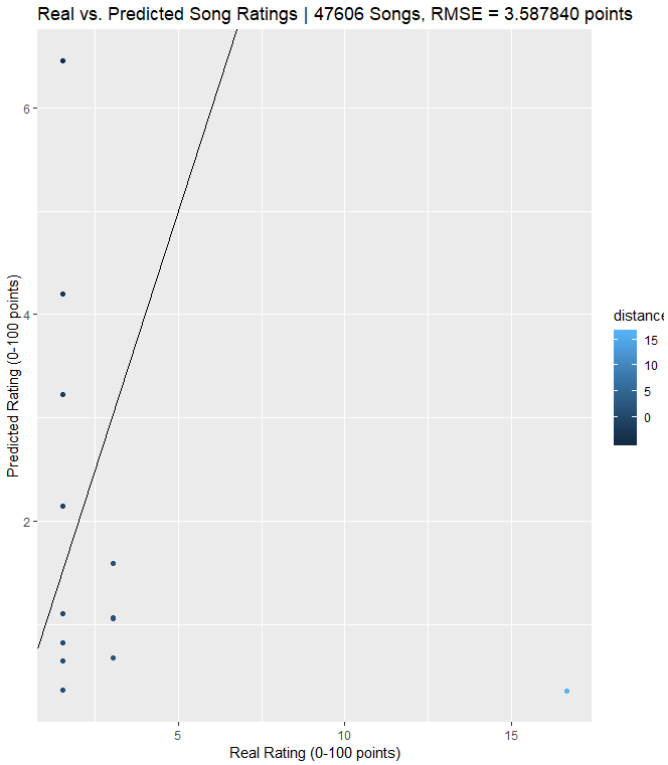
To evaluate the performance of the two recommender systems, I have done 10 random test cases (each algorithm 5 cases) to see their performance difference.

4.6.1 User-Based CF

```
[1] "Done with random sample #1!"
[1] "Done with random sample #2!"
[1] "Done with random sample #3!"
[1] "Done with random sample #4!"
[1] "Done with random sample #5!"
> rmse <- RMSE(dat2$user_rating, dat2$predicted_ra
ting, na.rm = TRUE)
> obs <- nrow(dat2)
> print(sprintf('After taking %d random samples of
 %d users each, with %d total song ratings and pre
dictions, the root mean squared error (RMSE) of ou
r recommender was %f %s.', i, test_size, obs, rmse
, "points"))
[1] "After taking 5 random samples of 100 users ea
ch, with 47606 total song ratings and predictions,
 the root mean squared error (RMSE) of our recomme
nder was 3.587840 points."
```

F 4.7 Calculate the RSME of 5 sample cases(a)

The RMSE values can be shown as following chart.
It can be seen that when the user scores a higher song, the recommendation system becomes less reliable and tends to underestimate the user's preference for the song. However, a high rating is an outlier in the data set - for most ratings, the forecast is close to the real user rating (represented by the diagonal).



F 4.8 Real Vs. Predicted Song Ratings(a)

song_id	title	artist	user_rating	predicted_rating
SOXPFLM12A8AE48C50	Electric Feel	MGMT	3.41	5.696667
SOUSMXX12AB0185C24	OMG	Usher featuring will.i.am	0.38	4.396667
SODGVGW12AC9075A8D	Somebody To Love	Justin Bieber	7.58	2.540000
SODGVGW12AC9075A8D	Somebody To Love	Justin Bieber	7.58	2.540000
SOUGOHW12AB018AEA3	Not In Love	Crystal Castles	3.03	1.820000
SODGCRN1288D3EB7B9	Pon De Floor	Major Lazer / Vybz Kartel / Afrojack	5.68	1.210000

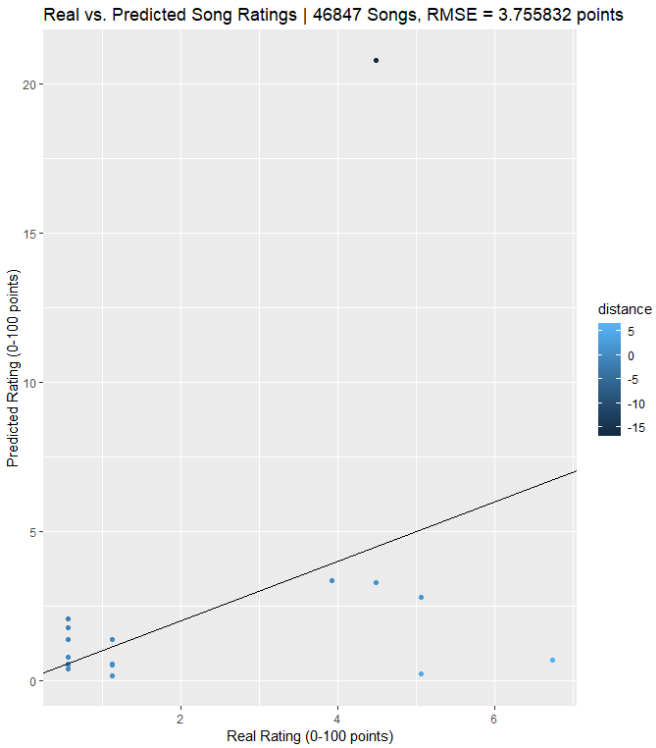
F 4.9 Content-Based Recommendation

4.6.2 Content-Based Filtering[39]

```
[1] "Done with random sample #1!"
[1] "Done with random sample #2!"
[1] "Done with random sample #3!"
[1] "Done with random sample #4!"
[1] "Done with random sample #5!"
> rmse <- RMSE(dat2$user_rating, dat2$predicted_ra
ting, na.rm = TRUE)
> obs <- nrow(dat2)
> print(sprintf('After taking %d random samples of
 %d users each, with %d total song ratings and pre
dictions, the root mean squared error (RMSE) of ou
r recommender was %f %s.', i, test_size, obs, rmse
, "points"))
[1] "After taking 5 random samples of 100 users ea
ch, with 46847 total song ratings and predictions,
 the root mean squared error (RMSE) of our recomme
nder was 3.755832 points."
```

F 4.10 Calculate the RSME of 5 sample cases(b)

The RMSE values can be shown as F4.10.
It can be seen that when the user score higher an artist's song, the recommendation system becomes less reliable and tends to underestimate the user's preference for the artist. However, a high rating is an outlier in the data set - for most ratings, the forecast is close to the real user rating (represented by the diagonal).



F 4.11 Real Vs. Predicted Song Ratings(b)

5. CONCLUSION

From the evaluation: both of the two recommenders predicted song ratings for users that were usually within 4 points of the user's real song rating, on a scale of 0 to 100.

Therefore, we can say that the proportion of user song listens is an implicit rating that can reliably predict user song preferences.

Another conclusion is that via compare the two recommenders RMSE values, we can see that there is not so much difference when dealing with large-scale data. However, user-based may need more time and space to do training. So, under this circumstance, content-based filtering may be ideal for those do not have enough memory.

For further research, we can do larger sample and more indicators to imply more comprehensive comparison.

6. ACKNOWLEDGMENTS

Thanks to Kavya Beheraj and Jeremy O'Brien's coding sample[28] for help me understand better about how to do recommender system visualization.

7. REFERENCES

[1] Mortensen, M. (n.d.). Design and Evaluation of a Recommender System. [online] Pdfs.semanticscholar.org. Available at: https://pdfs.semanticscholar.org/5036/1d7dfff1430f0578172bd431ea9259b56c62.pdf [Accessed 13 Apr. 2019].

[2] Loren Terveen, W. (n.d.). Beyond Recommender Systems: Helping People Help Each Other. [online] Citeseerx.ist.psu.edu. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.2437 [Accessed 13 Apr. 2019].

[3] Su, X. and Khoshgoftaar, T. (n.d.). A Survey of Collaborative Filtering Techniques.

[4] F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation. Egyptian Informatics Journal, Volume 16, Issue 3, 2015, Pages 261-273. ISSN 1110-8665, DOI= https://doi.org/10.1016/j.eij.2015.06.005.

[5] Aggarwal, C. (n.d.). Recommender Systems - The Textbook. [online] Springer.com. Available at: https://www.springer.com/us/book/9783319296579 [Accessed 13 Apr. 2019].

[6] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. 2008. Addressing cold-start problem in recommendation systems. In Proceedings of the 2nd international conference on Ubiquitous information management and communication (ICUIMC '08). ACM, New York, NY, USA, 208-211. DOI= http://dx.doi.org/10.1145/1352793.1352837.

[7] Lops P., de Gemmis M., Semeraro G. (2011) Content-based Recommender Systems: State of the Art and Trends. In: Ricci F., Rokach L., Shapira B., Kantor P. (eds) Recommender Systems Handbook. Springer, Boston, MA. DOI= https://doi.org/10.1007/978-0-387-85820-3_3

[8] Oord A., Dieleman, S. and Schrauwen, B. (n.d.). Deep content-based music recommendation. [online] Dl.acm.org. Available at: https://dl.acm.org/citation.cfm?id=2999907 [Accessed 13 Apr. 2019].

[9] John S. Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (UAI'98), Gregory F. Cooper and Serafín Moral (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 43-52.

[10] Chahal, M. (2016). Information Retrieval using Jaccard Similarity Coefficient. International Journal of Computer Trends and Technology, 36(3), pp.140-143.

[11] Benesty J., Chen J., Huang Y., Cohen I. (2009) Pearson Correlation Coefficient. In: Noise Reduction in Speech Processing. Springer Topics in Signal Processing, vol 2. Springer, Berlin, Heidelberg. DOI= https://doi.org/10.1007/978-3-642-00296-0_5

[12] A. H. Rangkuti, R. B. Bahaweres and A. Harjoko, "Batik image retrieval based on similarity of shape and texture characteristics," 2012 International Conference on Advanced Computer Science and Information Systems (ICACSIS), Depok, 2012, pp. 267-273.

[13] Sandeep Tata and Jignesh M. Patel. 2007. Estimating the selectivity of tf-idf based cosine similarity predicates. SIGMOD Rec. 36, 2 (June 2007), 7-12. DOI: https://doi.org/10.1145/1328854.1328855

[14] Papagelis M., Rousidis I., Plexousakis D., Theoharopoulos E. (2005) Incremental Collaborative Filtering for Highly-Scalable Recommendation Algorithms. In: Hacid MS., Murray N.V., Raś Z.W., Tsumoto S. (eds) Foundations of Intelligent Systems. ISMIS 2005. Lecture Notes in Computer Science, vol 3488. Springer, Berlin, Heidelberg. DOI= https://doi.org/10.1007/11425274_57

[15] Pasquale Lops, Marco de Gemmis and Giovanni Semeraro, Chapter 3 in Recommender Systems Handbook, 2011

[16] Hiemstra, D. Int J Digit Libr (2000) 3: 131. https://doi.org/10.1007/s007999900025. DOI= https://doi.org/10.1007/s007999900025

[17] P. Soucy and G. W. Mineau, "A simple KNN algorithm for text categorization," Proceedings 2001 IEEE International Conference on Data Mining, San Jose, CA, USA, 2001, pp. 647-648. DOI= 10.1109/ICDM.2001.989592.

[18] Yoav Freund and Llew Mason. 1999. The Alternating Decision Tree Learning Algorithm. In Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99), Ivan Bratko and Saso Dzeroski (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 124-133.

[19] Mohamed El Kourdi, Amine Bensaid, and Tajje-eddine Rachidi. 2004. Automatic Arabic document categorization based on the Naïve Bayes algorithm. In Proceedings of the Workshop on Computational Approaches to Arabic Script-based Languages (Semitic '04). Association for Computational Linguistics, Stroudsburg, PA, USA, 51-58.

[20] Joachims, T. (1996). A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.. [online] Apps.dtic.mil. Available at: https://apps.dtic.mil/docs/citations/ADA307731 [Accessed 15 Apr. 2019].

[21] Xavier Amatriain. 2013. Big & personal: data and models behind netflix recommendations. In Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications (BigMine '13). ACM, New York, NY, USA, 1-6. DOI= http://dx.doi.org/10.1145/2501221.2501222

[22] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In Proceedings of the fourth ACM conference on Recommender systems (RecSys '10). ACM, New York, NY, USA, 39-46. DOI= https://doi.org/10.1145/1864708.1864721

[23] Feng-Hsu Wang, Hsiu-Mei Shao, Effective personalized recommendation based on time-framed navigation clustering and association mining, Expert Systems with Applications, Volume 27, Issue 3, 2004, Pages 365-377, ISSN 0957-4174, DOI= https://doi.org/10.1016/j.eswa.2004.05.005.

[24] Goutte C., Gaussier E. (2005) A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In: Losada D.E., Fernández-Luna J.M. (eds) Advances in Information Retrieval. ECIR 2005. Lecture Notes in Computer Science, vol 3408. Springer, Berlin, Heidelberg. DOI= https://doi.org/10.1007/978-3-540-31865-1_25

[25] Story, M. and Congalton, R. (1986). Accuracy Assessment: A User's Perspective. [online] Asprs.org. Available at: https://www.asprs.org/wp-content/uploads/pers/1986journal/mar/1986_mar_397-399.pdf [Accessed 16 Apr. 2019].

[26] Neo4j Graph Database Platform. (n.d.). Neo4j Graph Platform – The Leader in Graph Databases. [online] Available at: https://neo4j.com/ [Accessed 20 Apr. 2019].

[27] White, N. (n.d.). RNeo4j package | R Documentation. [online] Rdocumentation.org. Available at: https://www.rdocumentation.org/packages/RNeo4j/versions/1.5.0 [Accessed 21 Apr. 2019].

[28] Beheraj, K. and O'Brien, J. (n.d.). RPubs - DATA 607, Final Project – Music Recommender with Neo4j. [online] Rpubs.com. Available at: https://rpubs.com/ksb357/388973 [Accessed 22 Apr. 2019].

[29] Wickham, H., Chang, W., Henry, L., Pedersen, T., Takahashi, K., Wilke, C. and Woo, K. (n.d.). Create Elegant Data Visualisations Using the Grammar of Graphics [R package ggplot2 version 3.1.1]. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/ggplot2/index.html [Accessed 23 Apr. 2019].

[30] Hahsler, M. (n.d.). Lab for Developing and Testing Recommender Algorithms [R package recommenderlab version 0.2-4]. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/recommenderlab/index.html [Accessed 23 Apr. 2019].

[31] Wickham, H., François, R., Henry, L. and Müller, K. (n.d.). Package 'dplyr'- A Grammar of Data Manipulation. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/dplyr/dplyr.pdf [Accessed 23 Apr. 2019].

[32] Bache, S. (n.d.). Abstract of Magrittr. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html [Accessed 23 Apr. 2019].

[33] Wickham, H. (n.d.). Introduction to stringr. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html [Accessed 23 Apr. 2019].

[34] Wickham, H. and Henry, L. (n.d.). Easily Tidy Data with 'spread()' and 'gather()' Functions [R package tidyr version 0.8.3]. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/tidyr/index.html [Accessed 23 Apr. 2019].

[35] Xie, Y. (n.d.). Package 'knitr'- A General-Purpose Package for Dynamic Report Generation in R. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/knitr/knitr.pdf [Accessed 23 Apr. 2019].

[36] Zhu, H. (n.d.). Construct Complex Table with 'kable' and Pipe Syntax [R package kableExtra version 1.1.0]. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/kableExtra/index.html [Accessed 23 Apr. 2019].

[37] Le, E. (n.d.). How to build a simple song recommender system. [online] Towards Data Science. Available at: https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85 [Accessed 20 Apr. 2019].

[38] White, N. (n.d.). graphgist - Neo4j Graph Database Platform. [online] Neo4j.com. Available at: https://neo4j.com/graphgist/movie-recommendations-with-k-nearest-neighbors-and-cosine-similarity [Accessed 23 Apr. 2019].

[39] Guides.neo4j.com. (n.d.). [online] Available at: http://guides.neo4j.com/sandbox/recommendations [Accessed 20 Apr. 2019].