Information Retrieval Project Report

Karel Beckeringh, s2600358
Wessel Reijngoud, s2580748
Nik van 't Slot, s2536420

January 2018

1 Abstract

There is an abundance of music available in today's world. Social Media services have begun to implement recommender systems to help people discover artists or songs that suits them based on their or similar users' taste in music. In this project we test which method works best for accurately recommending music artists to listen to on the service Last.FM. We implement an algorithm based on user-based similarity, item-based similarity (in this case the artists), Friend-based similarity and a combination of the aforementioned. Friend-based similarity here works with the Friend system that Last.FM has implemented. Our results reveal that we can tremendously increase the accuracy of the recommendations compared to the baseline, which is determined by recommending random artists to the user. There is however a great disparity in musical taste, which is unique for everyone and this influences our way of evaluating the program. A recommendation system for music could work in a lot of different ways, so if further research would have been done into this subject matter finding a different, more reliable way to evaluate the recommendations would be preferred.

2 Problem Statement

There is an abundance of music available in today's world. Numerous songs, artists and even genres have been popping up all over the world over the last couple of decades. The sheer amount of musical material sometimes makes it difficult for a person to find new music that he or she would really like. To remedy this problem, services have begun to implement recommender systems to help people discover artists or songs that suits them based on their or similar users' taste in music.

This is where our project comes into play. We are going to build a recommender system for Last.FM, one of these services, because we would like to know which method works best (user-based, item-based or a combination of techniques) for accurately recommending music artists to listen to. Last.FM is

an online service that allows users to listen and share any music that they would like. Since their inception in 2003, over 100,000,000,000 tracks have been listened to on the service. Our recommender system will utilise a provided dataset containing the activity of users in regards to which artists they have listened to and how much they have listened to those artists to determine similar users for the user-based recommender. For the item-based recommender, we will make use of a provided dataset of tags that have been given to songs and artists to determine similar songs or artists. We believe this system could prove fruitful in helping improve the algorithm for discovering new music on the Last.FM service. What (combination of) recommender technique(s) generates the best results in recommending artists to users?

This project is directly derived from the research proposal submitted by Karel Beckeringh (s2600358). We felt that he hit the nail on the head with his proposal, since the subject matter and dataset is rather unique and appeals to all of us on a personal level and the project would offer a welcome challenge for us as a group. As such, we unanimously decided to perform his proposal.

3 Literature

The study performed by Purushotham, Liu and Kuo (2) partly focuses its efforts on the same dataset as our project. In their study, their objective is to predict a user's would-be rating of an item-based on available social network information, such as bookmarking preferences on del.icio.us, compared to personal taste as signalled by the indicated preferences of a user's profile on Last.FM. To achieve this, they employ a Collaborative Filtering model, much like we attempt to implement in our project. The general idea behind the concept of Collaborative Filtering is that users that are alike will favour products that are alike as well, which can be untrustworthy at times due to common issues such as a deficiencies and an unevenness of sufficient data.

Purushotham, Liu and Kuo go the extra mile however, by also implementing both Collaborative Topic Regression and Social Matrix Factorization into their study. Matrix Factorization requires computing the latent representations of users and items given a matrix of ratings, according to Wang and Blei (4). In the same paper, they explain that Collaborative Topic Regression constitutes users with topic interests, fluctuating the weight of both content and other users on the prediction based on the amount of users that have rated the topic. While all of these approaches are incredibly informative and certainly piqued our interest, we felt it was best to limit our project to Collaborative Filtering and not get in over our heads with much bigger-scale researches.

Additionally, Sarwar, Karypis, Konstan and Riedl (3) have experimented with a new thought-of algorithm for Recommender Systems based on Collaborative Filtering. They built an item-based recommender technique which they claim can provide an improved standard of quality when compared to widely utilised user-based recommender techniques. As part of the experiment, they investigated several possibilities for computing item-item similarities, for instance

correlation between two items and cosine similarities between item vectors. To eventually retrieve the recommendations, they mainly employed weighted sum and regression models. Regression analysis comes down to comparing a dependant variable to several independent variables, observing any possible variation in the dependant variable when altering any of the independent variables and leaving the others untouched.

The paper written by Yoshii, Goto and Komatani (5) presents a different approach in recommending music with their hybrid recommender system. Apart from the user ratings, they analysed acoustic features of audio signals as well to be able to rank musical pieces in multiple ways. Their study attempts to leave subjective variables such as an individual's taste and outliers out of the equation as much as they can. As a result, their system is quite extensive, but able to preserve a staggering accuracy sore, no matter the addition of users or items.

4 Materials

4.1 Dataset

The dataset that has been used is mostly the same as found in HetRec (1) with some of our own files added.

- artists.dat - This file contains information about music artists listened and tagged by the users.
- tags - This file contains the set of tags available in the dataset.
- user_artists.dat – This file contains the artists listened by each user. It also provides a listening count for each [user, artist] pair.
- user_friends.dat – These files contain the friend relations between users in the database.
- test.dat – This file contains 10
- training.dat – This file contains the other 90

Also pickle files created by ourselves that use files shown above to make the program more efficient have been used.

- item_sim50.pickle – This file contains the cosine similarity between artists, only containing those having a score of 0.50 or higher.
- user_sim01.pickle – This file contains the cosine similarity between users, only containing those having a score of 0.01 or higher.

5 Methodology

5.1 Cosine Similarity

In the case of cosine similarity, two items are thought of as to be vectors in a k dimensional space. The cosine of the angle between the two vectors is seen to be the similarity between the two items, being a value between 0 and 1. The cosine_sim.py program included in the files is a basic representation on how we computed similarity scores between users and items.

5.2 Baseline

In order to interpret the accuracy scores produced by the following recommender systems, we need some point of reference. For this purpose, we built a baseline recommender, whose task is to recommend 10 random artists for each user. The accuracy that follows from this recommendation gives an indication of how much of the correct recommendations is caused by pure chance.

5.3 Item-based

The most crucial step in the item-based recommender is to compute the similarity scores using cosine similarity. The file containg these similarities between artists is called "item_sim50" and only contains similarity scores higher than 0.50 as we found out that none of the results we first had, using a file containg only scores higher than 0.01, didn't contain any similarity scores lower than 0.50. Setting this value higher, made the similarity file much smaller, going from 23 million lines to 1.1 million lines. This in turn made running the program much faster.

Our item-based recommender first looks up what artists user U already follows, using the training dataset. These artists and their listencount are added to a list for the user U, sorted in a top N by listencount from highest to lowest. Then using the item similarity pickle explained above, similar artists to the ones U already follows could be computed. These also were sorted in a top N from highest to lowest on their similarity scores. Using the listen count for each already listened artist and the similarity scores for the similar artists, a weight is calculated by multiplying the listencount by the similarity scores. This is done to only get the most similar artists to artists that user U listens to a lot, but doesn't know yet. This weighted score for each similar artist is sorted in a top 10 and this in turn is the end result for the recommendation of artists that user U doesn't listen to yet, but might want to.

For the optimal value of N it was found that N=30 gives the most desirable results for speed and score.

5.4 User-based

A user-based recommender also works with similarity scores. Here, the similarities between users is needed. Therefore, the task is to compare user profiles and

compute their cosine similarity scores. We decided to keep the threshold for relevant similarities at 0.01, since the amount of user similarities was significantly lower than of item similarities. Also, the similarity values were a lot lower, so when increasing this threshold, relevant data would be discarded.

The task of our user-based recommender is to look up the most similar users for the user U and make a recommendation based on the artists that are most listened to by these similar users.

Finding the most similar users is achieved by consulting the "user_sim01" dictionary and ordering them by their similarity scores to create a top N. We found that a value for N of 34 turned out to work best. For each of the users in this top N, every artist listened to (excluding artists that are already listened to by user U) is added to a defaultdict with a weight that is determined by the user's similarity to user U. After that, all that's left is to sort the defaultdict in order to get 10 recommended artists for user U.

5.5 Friend-based

The task of the friend-based recommender is to recommend artists for user U based on the top listened artists of the user's friends.

For every user, the "friends" dictionary is consulted to find their friends. For every friend, the listened artists are added to a defaultdict with a weight that is determined by the corresponding listen count. Again, after sorting this defaultdict, 10 recommendations can be obtained.

5.6 Combinations

Since user-based and friend-based seemed to be the most promising methods, we also tried to combine the two in order to improve the results. We tried doing this in both ways, that is, implementing friend relations in the user-based recommender and implementing similarity scores in the friend-based recommender.

For the first one, all that was needed was a check for each similar user if he was a friend of user U. If so, their similarity score was increased to give their recommendations priority.

And for the second one, the recommendation of each friend of U was not only weighted by the listen counts, but also by their similarity to U. Each friend that wasn't similar to U, was not taken into account.

6 Evaluation of the results

In order to properly evaluate the different recommender systems, the complete dataset "user_artists.dat" was randomly split into a test set 10% and a training set 90%. This was done randomly so the recommender system will predict missing artists from the training set instead of recommending new ones. This way an accuracy can be easily calculated. From the top N recommended artists that the recommender gives back it will look if that artist is already in the

	Baseline	Item-based	User-based	Friend-based	User + friend	Friend + U sim
Correct out of 9283	28	643	1794	1010	1823	1186
Accuracy	0.30 %	6.93%	19.33%	10.88%	19.64%	12.78%
Seconds to finish	13.127	54.887	20.639	6.929	22.813	6.953

Table 1: Results

training dataset and if it is, it is a hit and if it is not it is a miss. So in the end you can divide the total amount of hits by the total amound of lines in the test set and you have the percentage of correct predictions. In order to see how long the program took to finish a timer command in the terminal has been added to evaluate the efficiency of the code.

7 Results

The results to our recommender programs, portrayed by time to finish and percentage correrctly recommended, can be found in Table 1 and Figure 1 .
The last 2 columns of Table 1 correspond to a user-based approach with friend relations integrated and a friend based approach with user similarity integrated.

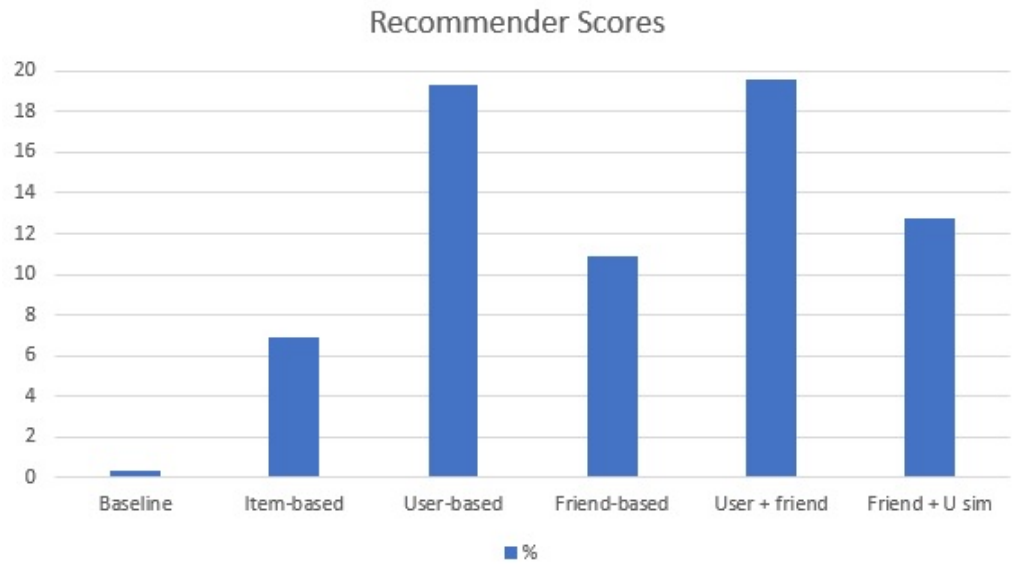

Figure 1: Recommender scores for the different techniques

8 Conclusion

It can be seen that all of the above techniques produce better results than the baseline, yet they do perform differently each. From Figure 1 it can be concluded that the user based approach to the recommender and the user based where friend relations have been implemented work the best. We can also conclude that for recommending artists an item based approach can definitely work, but the way of evaluation used in this research doesn't sufficiently support that.

9 Discussion

As you can see from Figure 1 our implemented approaches were all significantly better than the baseline. The user-based approach with friend implementation was most accurate and the item-based least accurate. The item-based recommender probably is least accurate as there is a lot of artists similar to the ones a user already listens to. Our way of evaluating the recommender by predicting the missing artists from the dataset with the test set, isn't saying much about the accuracy of a recommender. There's so much differences in musical taste, a user could have either a very narrow or a broad musical taste, which influences our way of evaluating the program. A recommendation for music could work in a lot of different ways so if further research would have been done to this, finding a different, more reliable way to evaluate the recommendations would be preferred.

References

[1] Iván Cantador. Last.fm, 2011.

[2] Sanjay Purushotham, Yan Liu, and C.-C. Jay Kuo. Collaborative topic regression with social matrix factorization for recommendation systems. *CoRR*, abs/1206.4684, 2012.

[3] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[4] Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 448–456, New York, NY, USA, 2011. ACM.

[5] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno. An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):435–447, Feb 2008.