Recommendation Tools: Song Recommendation with last.fm

BY Remo Boesiger, Victor Ernoult, Patrick Dundon, Liu Yi

The purpose of this report is to explain the code and steps taken to solve the questions of the group assignment for our recommendation tools course. Using the data from last.fm, the tasks of the group assignment were split into 4 steps.

1. Pre-processing the dataset of last.fm
2. Creation of functions for collaborative filtering, clustering and evaluation metrics
3. Application of these functions as well as the interpretation of their results
4. Discussion and argumentation of the actions and results

1. Dataset Pre-processing

In order to apply the different functions, the data provided needed to be transformed in several ways. First of all, we read the different datasets separately into R. Second, we made sure that only the same user and same artists appear in both of the matrixes by subsetting them accordingly. Then we prepared the content-based matrix. Therefore, we merged as suggested the '**user_taggedartists.dat**' and '**tags.dat**' datasets. After the merge we used the dcast function, changed the row names, converted the dataframe to matrix and changed the missing values to zeros. Thirdly, we prepared the matrix for collaborative filtering. There we took a similar approach. We used the spread function from the dplyr package to get from the long to the wide format, changed the row names and converted to a matrix.

Afterwards, we completed variable categorization for user_artist$weight. After checking the weight variable of the **user_artist** dataset, we found that most weight values are under 100,000, however the range we got from the user_artist$weight plot is from 0 to 300,000. This means there are small amount of outliers which we still need to take into account. As a result, we created 10 equal sized bins for the weight variable to for ease of later data processing to build recommendation tools.

2. Creation of Functions

We created several functions over the course of developing our recommendation models, which are outlined below.

2.1 User-based and Item-based CF Functions

Based on the course R code materials, a user-based collaborative filtering function was made, using the correlation similarity measure. The function is basically divided in three main parts. First, a similarity matrix between all users (or items) is computed.The idea there is to calculate the Pearson correlation score for each pair of user, by looking at their common ratings (or whichever sort of feedback that's used). Based on this score, each user

gets assigned a list of nearest neighbors, the other users with which the highest correlation score is obtained. Finally, these similar users provide information on the likelihood of the user liking a given unknown song. Similarly, the item-based approach uses items instead of users as nearest neighbors, to predict the likelihood of a user liking an item based on his liking of similar items that he already knows.

2.2 Cluster-based CF Functions

We began with deploying a non-hierarchical clustering function, as the framework for that had previously been provided in the course materials. After running this, we created a function for hierarchical clustering as to be able to compare the results with the non-hierarchical function. We later attempted to build a function for a three-step clustering which was seen in a previous course. However, due to time constraints with respect to function running time, we decided against it.

2.3 Content-based Function

The content-based recommendation filtering was built on the basis of the function provided in class. In order to ensure the function ran properly, we made two separate matrices based on artistID ~ tagValue and userID ~ artistID. This ensured a match in terms of # of artistID's (the # of observations in one matrix matched with the number of columns in the other). Inputs in the function therefore included these two matrices, as well as the chosen number of nearest neighbours and number of recommendations to provide. The function is determining nearest neighbours through creating a similarity matrix and then making predictions much like the process of item-based collaborative filtering. In the end, we took a subset of the data to run the model as running at full size was impractical in terms of time.

2.4 MAE (Regression) and F1 (Classification) Evaluation Functions

Two additional functions were created, to evaluate both regression and classification results.

The MAE, Mean Absolute Error, gives the average deviation between predictions and reality, in a quantifiable way. It is a useful metric for transmitting results to any kind of audience, due to its intuitiveness and interpretability. On the other hand, it makes use of absolute values, which implies a loss of information. It also relies on the scale of the predicted variable, and therefore cannot be used as a comparison between models applied for different problems. The function is implemented in a very simple way. For each prediction in the test set, we look at how far it is from reality (in absolute terms). We then sum it all up and divide by the total number of predictions.

The F1 score is a synthesis of the recall and precision score. It is used in classification cases, where the average error cannot be quantified as a continuous variable. Instead, we look at the number of correctly classified cases, and compare it to every positive prediction (recall), and to truly positive cases (precision). Therefore, precision gives the percentage of cases predicted positives which are actually relevant, while recall gives the percentage of total relevant results correctly classified.

The F1 score is a way to combine those 2 measures into one meaningful metric. It simply takes the harmonic mean between recall and precision. It can be customized so as to weigh heavier the metric deemed more important (in some cases, false positives must be avoided at all costs, in others, false negatives are the ones that should be cared for). The function that implements this score takes in 2 extra arguments, besides the predictions and real values.

Threshold allows the user to pick the limit at which a prediction becomes a positive. Usually the threshold takes a default value of 0.5, meaning predictions under 0.5 are considered negatives. Based on this threshold, the number of true positives, false positives and false negatives are counted. The recall, precision and F1-score are then computed. A TopN option also exists, which computes those scores for the best N grades (or any metric used), so as to evaluate how well the algorithm predicted the favorite artists of a user.

3. Application of functions and our results

3.1 Rationale behind chosen recommendation models

For any project, it is important to try multiple approaches to not only compare the outcomes, but also better understand the data you are working with. We wanted to be able to implement the major model types learned throughout the course and ultimately have enough results to see how well different models perform.

Our two major focuses, content-based and collaborative filtering, are quite different in structure and thus beneficial for the purpose of evaluation. With content-based filtering, we wanted to be able to generate predictions of which artists users would like based purely on the similarity between the artists in regards to their tags.

Conversely, with item and user-based collaborative filtering, the aim was to identify user artist preference. We then could provide recommendations based on what other users who listened to that artist also listen to.

3.2 Evaluation Metrics Chosen

As our predictions are continuous, we are in a regression case and need to adapt our evaluation metrics accordingly. More precisely, MAE and RMSE were considered. As mentioned earlier, MAE is very interpretable but has its limits. RMSE helps with these, by providing a measure that can be compared cross-cases. It also penalizes more heavily the big variations in error. As we are comparing models on the same data, and for the sake of understandability, we used MAE as the main comparative metric.

However, by feeding in a relevant threshold, we can transform the prediction task into a classification problem, where the model tries to predict whether a song will be liked or not. As our dependent variable is split into 10 groups, we can consider a prediction equal or

above 7 out of 10 to be a good song. We can therefore use the classification evaluation metrics, such as recall, precision and F1. These will help us consider the actual value brought to the customers, and not simply the accuracy of the model. In other words, we can now evaluate how many bad artists are predicted compared to good ones.

For a song recommendation system, it could be argued that precision matters more than recall. The user can listen to a finite number of artists, which we want to make as enjoyable as possible. Making sure to limit the number of false positives should therefore be a priority over reducing false negatives.

3.3 Hybrid Recommendation Techniques

We constructed two separate hybrid recommendation techniques as part of our testing, a weighting system and a switching system. We decided on these techniques because of the following reasons. Firstly, the switching system tries to improve the final predictions by combining the predictions of two systems. It takes the prediction of either system when it is most suitable. Secondly, the weighting approach intends to reduce the dependency of the final results on only one model. Last but not least, both methods are flexible, meaning the weight given in to each system could be changed easily but also the interchangeability of the systems is given.

For the weighting system, we took the prediction results of the previously established item-based and collaborative filtering models and placed them together into a final dataframe. From this point, both prediction results were added together and divided by two, so that both content-based and collaborative filtering models were weighted equally in determining the predictions. This approach increases the stability of the overall recommendation system, as each model can compensate for the perceived weaknesses of the other. If we felt that one of the particular models was more/less important, the weightings could be adjusted accordingly.

Our second hybrid approach again used the two previous item-based and user-based collaborative filtering models. This time, we wanted to help prevent sparsity by ensuring as many users as possible would get a recommendation. We therefore used the dataframes for each model and compared them for each user. If only one model provided a prediction score for that user, then that would be there final prediction score assigned. If both models provided a prediction score for that user, then the average of the two scores would be taken and used as the final prediction score. If both models had no prediction score for that user, then obviously no final score would be assigned.

The collaborative filtering models were chosen to build the hybrid models because they were of the same size and yielded better overall results than the other models according to the Mean Absolute Error function. However, the both methods could also be used to compare for example a clustering-based approach with the results of a content-based system.

3.4 Review of all systems - which one was best?

The following table shows the results for the different recommendation systems. The results for the item-based, user-based, content-based as well as the hybrid ones were calculated on subsets of the full data, as running the complete matrix was not possible in the given timeframe. Final results could therefore differ from the ones provided here.

System	MAE	RMSE	Recall	Precision	F1
non-hierarchical	2.62	0.13	0.28	0.50	0.36
hierarchical	2.14	0.16	0.16	0.76	0.26
user-based	**1.46**	**0.34**	**0.63**	**0.86**	**0.72**
item-based	4.28	0.91	0.01	1	0.02
content-based	4.33	0.77	0.04	0.86	0.08
hybrid-weighting	2.41	0.56	0.05	0.96	0.10
hybrid-switching	1.99	0.48	0.25	0.88	0.39

Considering the different evaluation-scores it becomes obvious that the user-based collaborative filtering recommendation system delivers the best results. One can also observe that the clustering methods are doing fairly well despite the spares matrix. On the other end we can find the item-based collaborative filtering and the content-based filtering which perform poorly. As already mentioned above, this could also result from the fact that we run the code on a subset and not on the entire matrix. The hybrid systems performed better than the item-based and content-based. However, the performance of both hybrid systems was worse than the one of the user-based system. This makes sense as both take into account, to a certain degree, the results of the item-based system which performs badly. If we have a look at the runtime of course the hybrid systems are the fastest as the just combine the results of the previously run systems. The clustering-methods are also quite fast with an approximate runtime of 10 minutes. Finally, the collaborative filtering and the content-based system take several hours up to days to run on the complete matrix. If this is a concern our ranking would maybe change. However for a company such as last.fm computational power should not be a problem. Therefore, we would suggest them to use the user-based collaborative filtering.

4. Discussion

In this specific case, some approaches made more sense than others. The number of users here was much smaller than the number of items, making the user-based collaborative filtering much more effective and faster than the item-based method. Furthermore, the data available at hand favorized user feedback rather than information about the products (not much information was known about the artists). It is therefore not surprising to see the collaborative filtering models performing better than content-based ones. Hybrid systems are one interesting approach to fixing some of the issues with a single model. However, in our case they were not able to improve the result in regards to the evaluation metrics we used.

As outlined previously, user-based recommendation was our best performing model. This approach is especially beneficial because it will continue to improve as the user-base increases; the more users available to analyze, the more accurate the recommendations will become. It should also be mentioned that the increase in user-base size does not require additional programming, thus keeping the model relatively low-maintenance compared to others (such as content-based).

Although, the user-based system gave the best results there is still room for improvement. One way would be by testing hybrid combinations between the complete results of several different systems. Maybe one of the combinations we were not able to test because of the tight schedule would have resulted in better results. Another way would be a combination with additional data which could be gathered by providing a sign in option with a Facebook or Google account.

Overall, collaborative filtering shows good performances if the data provided is sufficient. The model must however be selected depending on the case at hand, the quantity and quality of data available, and take into account other metrics than pure performance. These include computation time, computation resources used, ease of implementation, but also interpretability in some cases.