A Recommendation System for Spotify

Luigi Barba
Lorenzo Colantoni
Francesco Guglielmino
Jacopo Masini
Gianluca Surico

28th July 2020

Contents

Abstract

Music streaming services had become widely used in the last years. From a Data Analysis point of view, this can be seen as a chance to gather easily new information about users' listening preferences. As a consequence, recommendation systems have been deeply studied also from a statistical point of view. Collaborative filtering with implicit feedback data consists in analyzing the relationship between user and item, working on implicit signals as track play counts or click through data. Project goal consists in implementing this kind of recommendation system through a model based on matrix factorization and latent factors. The analysis is based on data collected through Spotify API (starting from our personal libraries).

1 Introduction

Recommendation systems have become an important research area, in the last twenty years a lot of work has been done both in industry and in the academic world on the development of new approaches for these systems.
In general, recommendation systems are based on two different strategies: the content-based approach creates a profile for each user or product to characterize it, but requires the collection of external information that may be difficult to collect. An alternative strategy, called Collaborative Filtering (CF), is based only on the behavior of past users without requiring the creation of profiles. The CF analyzes the relationships between users and the interdependencies between products, in order to identify new user-item associations.
In addition, the recommendation systems are based on different types of inputs. The easiest to manage is explicit feedback, which includes explicit input from users about their interest in products. However, explicit feedback isn't always available. The easiest to get is implicit feedback, which indirectly reflects the user's opinion. Types of implicit feedback can be the listening history.
Our project is based on the exclusive use of songs collected in Spotify's Favorites and extrapolated through its API.

2 Model overview

We define $R = (r_{ui}) \in \mathbb{R}^{m \times n}$ to be the user-item interaction matrix. Each entry r_{ui} specifies the interaction of item i by user u and can be thought of as a single observation. There are m users rating n items. In implicit feedback settings the goal is usually to decompose R into two lower-rank matrices representing latent user and item factors.
p_{ui} is a set of binary variables, which indicates the preference of user u to item i. The p_{ui} values are derived by binarizing the r_{ui} values:

$$p_{ui} = \begin{cases} 0 & \text{if } r_{ui} > 0 \\ 1 & \text{if } r_{ui} = 0 \end{cases} \tag{1}$$

We introduce a set of variables, c_{ui}, which measure our confidence in observing p_{ui}.

$$c_{ui} = 1 + \alpha log(1 + r_{ui}) \tag{2}$$

Cross-validation is used to determine appropriate values for α.
Our goal now is to find a vector $x_u \in \mathbb{R}^f$ for each user u, and a vector $y_i \in \mathbb{R}^f$ for each item i that will factor user preferences. In other words, preferences are assumed to be the inner products: $p_{ui} = x_u^T y_i$. These vectors will be known as the user-factors and the item-factors, respectively.
Accordingly, factors are computed by minimizing the following cost function:

$$\min_{x_*, y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2) \tag{3}$$

The second term is necessary for regularizing the model such that it will not overfit the training data. Exact value of the parameter λ is data-dependent and determined by cross validation.
Observe that when either the user-factors or the itemfactors are fixed, the cost function becomes quadratic so its global minimum can be readily computed. This leads to an alternating-least-squares optimization process, where we alternate between re-computing user-factors and itemfactors, and each step is guaranteed to lower the value of the cost function.
Let us now assume that all item-factors are gathered within an $n \times f$ matrix Y . Before looping through all users, we compute the $f \times f$ matrix $Y^T Y$. For each user u, let us define the diagonal $n \times n$ matrix C^u where $C^u_{ii} = c_{ui}$, and also the vector $p(u) \in \mathbb{R}^n$ that contains all the preferences by u (the p_{ui} values). By differentiation we find an analytic expression for x_u that minimizes the cost function (3):

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u) \tag{4}$$

A recomputation of the user-factors is followed by a recomputation of all item-factors. We arrange all user-factors within an $m \times f$ matrix X. First we compute the $f \times f$ matrix $X^T X$. For each item i, we define the diagonal $m \times m$ matrix C^i where $C^i_{uu} = c_{ui}$, and also the vector $p(i) \in \mathbb{R}^m$ that contains all the preferences for i. Then we solve:

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i) \tag{5}$$

After computing the user- and item- factors, we recommend to user u the K available items with the largest value of $\widehat{p}_{ui} = x_u^T y_i$, where $\widehat{p}_{ui}$ symbolizes the predicted preference of user u for item i.

3 Experimental setup

3.1 Data collection

The data we use has been collected from 12 Spotify accounts: 5 of them are our personal accounts, the rest from a circle of friends and relatives.

Each user has signed-in into the Spotify's Developer Platform. After signin-in, each user got a "Client ID" and a "Client Secret". Thanks to the "spotifyr" library in R, we were able to connect to the API (using the codes) and took all the information we needed to work with.
The data we worked with are the user's favourite songs.

3.2 Implementation

Our idea is to split the data into training and test set and then artificially split the test set into history and future data in order to quantify the performance of the model (this is due to the evaluation metric we chose). In R this will be done using the createDataPartition of the library caret.

The design matrix X is a user-item matrix in which the r_{ui} element is the number of saved songs made by artist i saved by user u. Obviously, the feedback is implicit (there is not an explicit rating of the items, but rather a level of confidence on how much a user likes an artist). In R, this will be saved in the 'sparseMatrix' format because of the sparsity of X and because rsparse works really well in combination with this type.

The tuning of the parameters will be done using a 4-fold cross-validation scheme, splitting the dataset into a training set made by 75% of the users and a test set made of the remaining part of the matrix. There are 3 tuning parameters: the number of latent factors, the α confidence parameter and the regularization parameter λ.

The fitting of the model is made possible by using the $model\$fit\_transform$ function of the rsparse package after setting up the model using $model = WRMF\$new()$.

The validation scheme is based on a precision measure called $MAP@k$: in fact, the loss function of the model has a different measure unit when changing the α and so we can't use eq. (3). In order to overcome this problem we will use the $MAP@k$ measure, where k is the number of items recommended to the user. $MAP@k$ is independent of α and it is pretty intuitive: suppose we mark with 1 an item that the users is going to like and 0 an item the user is going to hate (in our case, 1 if the artist belongs to the future test set and 0 if not): if - for example - the recommendation system gives the list $\{0, 1, 0, 1\}$, the $MAP@4$ gives the items the rating $\{0, \frac{1}{2}, 0, \frac{2}{4}\}$ and we average over the non-zero values of this list, thus we get $\frac{1}{2}$. In R, this is done using the ap_k function of the rsparse package, which returns the average precision of the predictions for each user in the test set: the $MAP@k$ will be the mean of this vector.

This evaluation metric will be used in order to tune the parameters.

After tuning the parameters, the recommended items will be displayed after using the $model\$prediction$ function, which - on a mathematical level - just performs another round of the ALS algorithm, minimizing the loss function for new test users and keeping the item embedded values fixed.

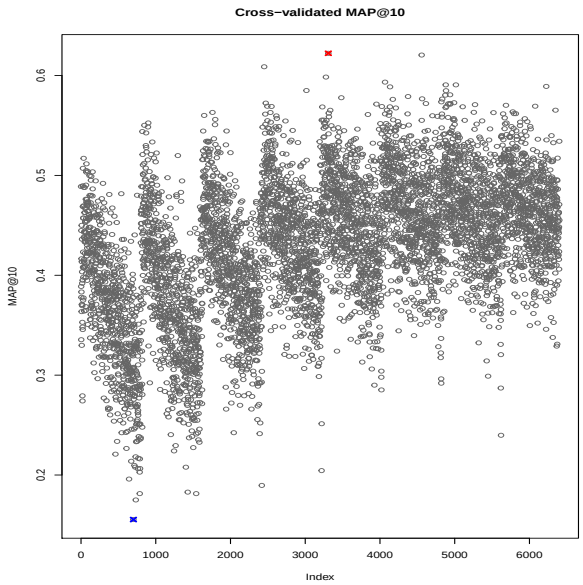
After implementing our recommendation system we tried to exploit the in-

formation coming from the "latent" item matrix in order to suggest to the users artists similar to the one already recommended. This was done using the rsparse $model\$components$, getting the item matrix obtained from the ALS algorithm, and computing cosine similarities between the columns of the matrix.

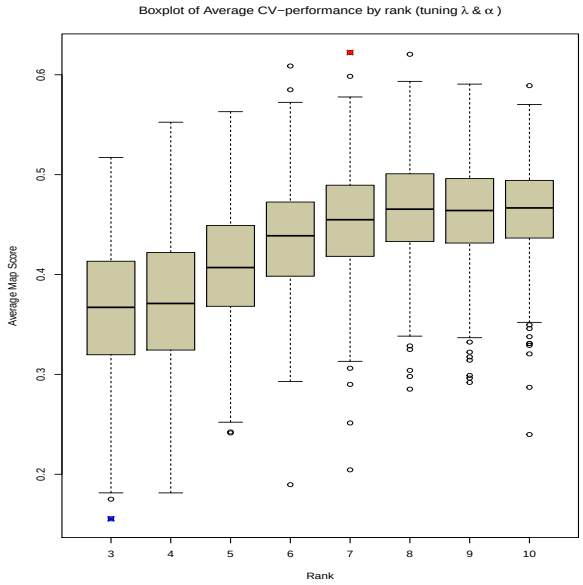
4 Results

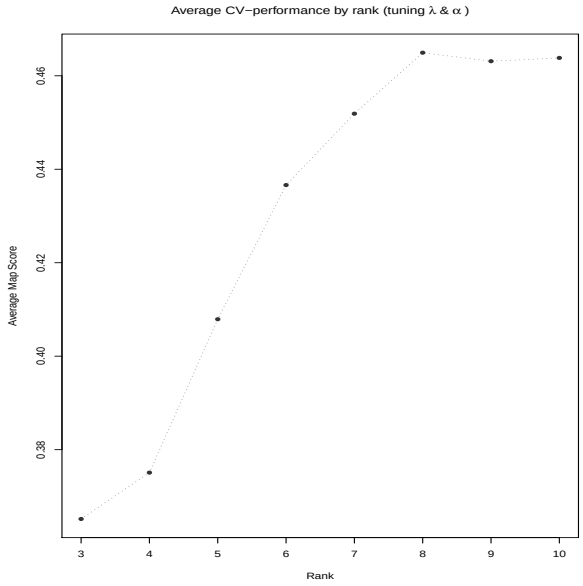
4.1 Cross Validation and Parameters Tuning

Before analyzing in depth the results, it's important to remark what we briefly said in the Implementation section: all we got from our algorithm isn't based on a "pure" Cross Validation based on the usually loss function but rather using the $MAP@k$ metric evaluation (independent from α) .



As we can see from the plot representing the Cross Validated parameters using the $MAP@10$, once the rank increases, at the same time also the Mean Average Precision (in average) increases.
The maximum value for the tuned parameter belongs to the model that uses 7 latent factors; the minimum one corresponds to the model that uses 3 latent factors. This probably can be interpreted in the way that once the rank increases, the R matrix approximates better the original one.
Then we continue our analysis spending few time on the Average Map Score as we can see in the following plot:





This clearly shows how the model is sensitive to the rank increasing. It also seems that performances of models with 8 or more latent factors are kind of stable. Although the latter models on average perform better than the previous ones, the best result comes from the set of models with 7 latent factors, which on average performs less than the models with more factors.

4.2 Model Recommendation

Once we got the values of the hyperparameters, we're ready to fit the model. It gives us the following predictions, considering 3 users (selected randomly).

Lorenzo	
Score	*Recommendation*
0.852	Arctic Monkeys
0.797	Bruno Mars
0.785	Black Eyed Peas
0.761	Billie Eilish
0.744	Sia
0.744	Michael Jackson
0.734	Mannarino
0.721	Eminem
0.708	Rino Gaetano
0.684	Nitro

Jacopo	
Score	*Recommendation*
0.562	Bruce Springsteen
0.528	Simon & Garfunkel
0.472	Coez
0.471	J-AX
0.471	The Clash
0.462	U2
0.459	The Beatles
0.455	Linkin Park
0.449	Ludovico Einaudi
0.438	alt-J

Gianluca	
Score	*Recommendation*
0.769	Coez
0.704	Simon & Garfunkel
0.650	J-AX
0.615	Bob Dylan
0.590	Linkin Park
0.576	U2
0.571	Jefferson Airplane
0.564	The Doors
0.558	The Smashing Pumpkins
0.538	Jarabe De Palo

As we can see from the tables, the small number of users we took affects the results in the way that the scores are not always high (which would guarantee ad hoc advice for the user based on his preferences).
In fact our analysis can focus on the score variability that seems quite high due to the value we see in the table. For example Lorenzo's scores are quite high and it seems that the model using the information we gave to it works well, but at the same time Jacopo's scores are too low and this means that there are not too many artists that match with his preferences.
Nevertheless the recommendations are quite good considering the lack of information.
This should be a starting point for a future work, in which we could consider a larger number of users that should improve the performances and so the model recommendations.

5 Bonus: Similarity between Artists

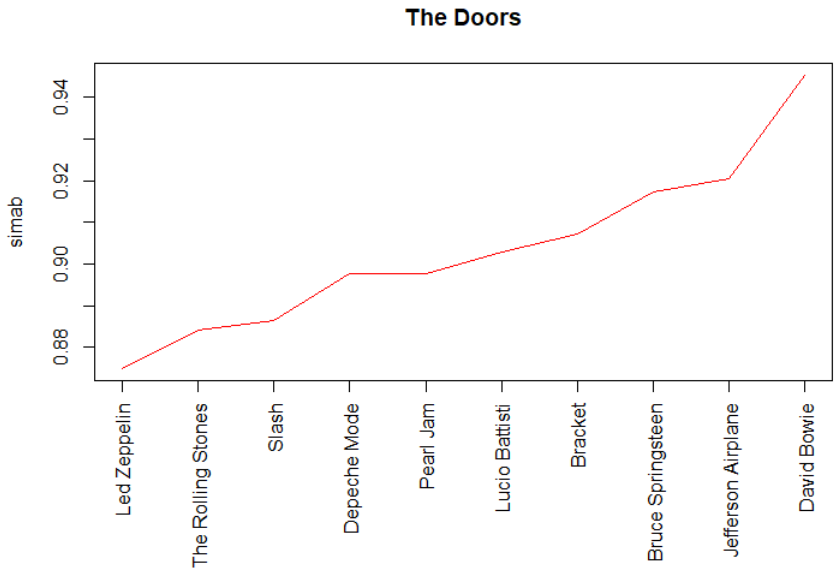
After achieving our goal, we investigated the similarities between the various artists. We used cosine similarity to measure the similarity between item vectors in a latent factor space. Given two vectors a and b the cosine similarity between the vectors is defined as:

$$sim_{ab} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}} \tag{6}$$

The top related items for a given artist a can be defined as the top artist b ranked by sim_{ab}. We calculated the top five related artists for those predicted by the algorithm using the learned item vectors.

Similar artists		
Gorillaz	*The xx*	*Iron Maiden*
Pink Floyd	Beck	Lou Reed
Led Zeppelin	Talking Heads	alt-J
Radiohead	Little Simz	The Rolling Stones
Pinguini Tattici Nucleari	Fleet Foxes	Phil Collins
Lindsey Stirling	Massive Attack	Caravan

In the following plot we can see for the artist "the Doors" the 10 most similar artists resulting from the matrix of latent factors, and the respective sim_{ab} values.



References

[1] Y. Hu and Y. Koren and C. Volinsky. *Collaborative Filtering for Implicit Feedback Datasets*. 2008 Eighth IEEE International Conference on Data Mining, pp 263-272, 2008.

[2] William Morgan *Weighted Alternating Least Squares with Implicit Feedback Data* STP 598 Spring 2018

[3] Christopher C. Johnson *Logistic Matrix Factorization for Implicit Feedback Data*. 2014.

[4] Himanshu Kriplani *Alternating Least Square for Implicit Dataset with code* https://towardsdatascience.com/alternating-least-square-for-implicit-dataset-with-code-8e7999277f4b

[5] Moussa Taifi *MRR vs MAP vs NDCG: Rank-Aware Evaluation Metrics And When To Use Them* https://medium.com/swlh/rank-aware-recsys-evaluation-metrics-5191bba16832.

[6] Dmitriy Selivanov *Matrix factorization for recommender systems* http://dsnotes.com/post/2017-06-28-matrix-factorization-for-recommender-systems-part-2/.