Master Degree in Data Science

A Data Science Game: Assessing music recommendation with Factorization Machines

Authors:
Roger Garriga Calleja, Javier Mas Adell and
Saurav Poudel

Director: Christian Fons-Ronsen

June 2017

ABSTRACT IN ENGLISH:

Internet has seen a tremendous growth in the last few years. Because of that, we have a lot of information about most of the things in web. And the usage of Recommendation system has become more important than ever.
Recommendation systems address this problem, by guiding users through the big ocean of information. Until now, recommendation systems have been extensively used within e-commerce and communities where items like movies, music and articles are recommended. More recently, recommendation systems have been deployed in online music players, recommending music that the users probably will like.
This thesis will present the design, implementation, testing and evaluation of a recommendation system within the music domain, where three different approaches for producing recommendations are utilized.
Testing each approach is done by evaluating the recommendation systems using precision scores. Our results show that the functionality of the recommendation system is satisfactory, and that recommendation precision differs for the three filtering approaches.

ABSTRACT IN CATALAN:

Internet ha experimentat un gran creixement en els últims anys. Per això, tenim molta informació sobre la majoria de les coses a la web. I l'ús del sistema de recomanacions s'ha tornat més important que mai.
Els sistemes de recomanació tracten aquest problema, guiant els usuaris a través del gran oceà d'informació. Fins ara, els sistemes de recomanacions s'han utilitzat àmpliament en el comerç electrònic i comunitats on es recomana articles com ara pel·lícules, música i articles. Més recentment, els sistemes de recomanació s'han desplegat en reproductors de música en línia, recomanant música que probablement els usuaris agradin.
Aquesta tesi presentarà el disseny, implementació, avaluació i avaluació d'un sistema de recomanacions dins del domini musical, on s'utilitzen tres criteris diferents per a la producció de recomenacions.
La prova de cada enfocament es fa avaluant els sistemes de recomanació utilitzant puntuacions de precisió. Els nostres resultats mostren que la funcionalitat del sistema de recomanacions és satisfactòria i que la recomanació prediu els diàmetres dels tres enfocaments de filtració.

A Data Science Game: Assessing music recommendation with Factorization Machines

Roger Garriga Calleja, Javier Mas Adell and Saurav Poudel

June 30, 2017

Abstract

Internet has seen a tremendous growth in the last few years. Because of that, we have a lot of information about most of the things in web. And the usage of Recommendation system has become more important than ever.

Recommendation systems address this problem, by guiding users through the big ocean of information. Until now, recommendation systems have been extensively used within e-commerce and communities where items like movies, music and articles are recommended. More recently, recommendation systems have been deployed in online music players, recommending music that the users probably will like.

This thesis will present the design, implementation, testing and evaluation of a recommendation system within the music domain, where three different approaches for producing recommendations are utilized.

Testing each approach is done by evaluating the recommendation systems using precision scores. Our results show that the functionality of the recommendation system is satisfactory, and that recommendation precision differs for the three filtering approaches.

Contents

1 Acknowledments

We would like to thank our supervisor, Professor Christian Fons-Rosen, for his valuable ideas, support, motivation and constant feedback through out the project. Even at times when we were struggling with ideas, or lagging behind the timeline, he was extremely generous and supportive to us.

We would also like to thank Jonas Paul Westermann for his instrumental support through out the project. It's difficult to put down his contribution in words actually. His Harry Potter-esque magical skills of Python and Terminal, his Erlich Bachman-esque attitude of 'We can do it guys!', it has been an absolute privilege working with him. And to put it in his own words: 'we will forever cherish those last few days spent mostly sitting shirtless at the table of a very overheated apartment living off frozen pizza and chips'.

We would also like to thank our brother and friend Jose Fernando Moreno for his valuable discussions and ideas related to the project. Whenever we need some new insights on any project, we know where to come from now on. He never doubted that we would make it to the final even when we were stuck and it seemed totally impossible.

We would also like to thank Massimo Quadrana for his guidance on Factorization Machine. This is the model that made sure we see a bit of Paris in September.

And lastly, thank you Omiros Papaspiliopoulos. Thank you so much for introducing us in this wonderful world of Data Science. It has been an absolutely amazing journey. We will always be thankful to you for that.

2 Background

Getting the perfect music recommendation is a challenging task. Who has never dreamt of laying back and listening to some music, while having no buttons to press at all and still getting the perfect tune? But what defines this perfect tune?

Deezer is a music streaming app, also available on the web. It proposes more than 43 million tracks and is available in more than 180 countries, through a free limited service and a premium offer.

For this online challenge, Deezer wants you to look at Flow, its own music recommendation radio. The concept of Flow is simple: it uses collaborative filtering to provide a user with the music he wants to listen at the time he wants.... And if he does not want to listen to some specific tracks and skips songs by pressing the 'Next song' button, then the algorithm should detect it quickly. In this context, getting the first song recommendation right is really important.

The goal of the challenge is to predict whether the users of the test dataset listened to the first track Flow proposed them or not. Deezer considers that a track is "listened" if the user has listened to more than 30 seconds of it (is listened =1). If the user presses the skip button to change the song before 30 seconds, then the track is not considered as being listened (is listened = 0).

The test dataset consists in a list of the first recommended tracks on Flow for several users. Each row represents one user.

The train dataset was generated using the listening history of these Deezer users for one month. Each row represents one listened track. The list of distinct users in the train dataset matches exactly with the test dataset's one.

Table 1: Original features from the competition dataset

Feature name	Description
genre id	ID of the genre of media (song)
ts listen	Unix time of the media
media id	ID of the media
album id	ID of the album of media
context type	Type of the context
release date	Release date of the media
platform name	Name of the platform
media duration	Duration of the media
listen type	Type of listen
user gender	Gender of the user
artist id	Id of the artist
user age	Age of the user
is listened	Whether the media is listened or not

3 Dataset

We have 7.558.834 observations with each observation having 15 features, so the shape of the train dataset without is 7.558.834 by 15.

The features provided in the training set proved to be very informative. Normally in collaborative filtering, we just have the tuples of users, items and ratings. But in our dataset of music, we have plenty of contextual information too. Like genre id, album id, age of the user, gender of the user, Unix time of the song, etc. Presence of these contextual information was one of the main reasons we used Factorization Machines, along with other conventional recommendation algorithms. Also using some of these features, we have done feature extraction and come up with more features to use in our model, which we will be explaining in a difference section below.

And as we can see from the plot below, the data we have is quite sparse, which is the case in most of the recommendation systems. There are very few songs that have been listened more than a few times. And similarly, there are very few users that have listened to more songs from the list.

Table 2: Unique values per feature

Feature name	Count of unique values
ts listen	2256230
media id	452975
album id	151471
context type	74
release date	8902
platform name	3
platform family	3
media duration	1652
listen type	2
user gender	2
user id	19918
artist id	67142
user age	13
is listened	2

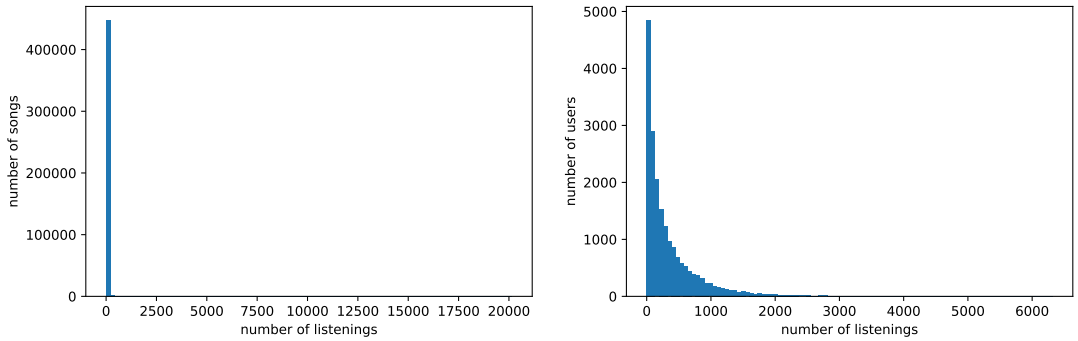

Figure 1: Distributions of listenings per for songs and users

4 General View on Recommendation Engines

4.1 Introduction

Recommendation Systems are software tools and techniques providing suggestions for items to be of use to a user.The goal of a Recommendation System is to generate meaningful recommendations to a collection of users for items or products that might interest them. Suggestions for books on Amazon, or movies on Netflix, are real world examples of the operation of industry-strength recommendation systems. The design of such recommendation engines depends on the domain and the particular characteristics of the data available. For example, movie watchers on Netflix frequently provide ratings on a scale of 1 (disliked) to 5 (liked). Such a data source records the quality of interactions between users and items. Additionally, the system may have access to user-specific and item-specific profile attributes such as demographics and product descriptions respectively.

4.2 Information Collection Phase

This collects relevant information of users to generate a user profile or model for the prediction tasks including user's attribute, behaviors or content of the resources the user accesses. A recommendation agent cannot function accurately until the user profile/model has been well constructed. The system needs to know as much as possible from the user in order to provide reasonable recommendation right from the onset. Recommendation systems rely on different types of input such as the most convenient high quality explicit feedback, which includes explicit input by users regarding their interest in item or implicit feedback by inferring user preferences indirectly through observing user behavior. Hybrid feedback can also be obtained through the combination of both explicit and implicit feedback.

There are mainly two kinds of information retrieval proces and a combination of both.

- Explicit Feedback: The system normally prompts the user through the system interface to provide ratings for items in order to construct and improve his model. The accuracy of recommendation depends on the quantity of ratings provided by the user. The only shortcoming of this method is, it requires effort from the users and also, users are not always ready to supply enough information. Despite the fact that explicit feedback requires more effort from user, it is still seen as providing more reliable data, since it does not involve extracting preferences from actions, and it also provides transparency into the recommendation process that results in a slightly higher perceived recommendation quality and more confidence in the recommendations.

- Implicit Feedback: The system automatically infers the user's preferences

by monitoring the different actions of users such as the history of purchases, navigation history, and time spent on some web pages, links followed by the user, content of e-mail and button clicks among others. Implicit feedback reduces the burden on users by inferring their user's preferences from their behavior with the system. The method though does not require effort from the user, but it is less accurate. Also, it has also been argued that implicit preference data might in actuality be more objective, as there is no bias arising from users responding in a socially desirable way and there are no self-image issues or any need for maintaining an image for others.

- Hybrid Feedback: The strengths of both implicit and explicit feedback can be combined in a hybrid system in order to minimize their weaknesses and get a best performing system. This can be achieved by using an implicit data as a check on explicit rating or allowing user to give explicit feedback only when he chooses to express explicit interest.

4.3 Prediction/Recommendation Phase

After the collecting phase, there is the recommendation phase, in which the recommender system recommends or predicts what kind of items the user may prefer. This can be made either directly based on the data set collected in information collection phase which could be memory based or model based or through the system's observed activities of the user.

5 Recommendation Filtering Techniques

Recommendation filtering techniques can be classified into the following types:

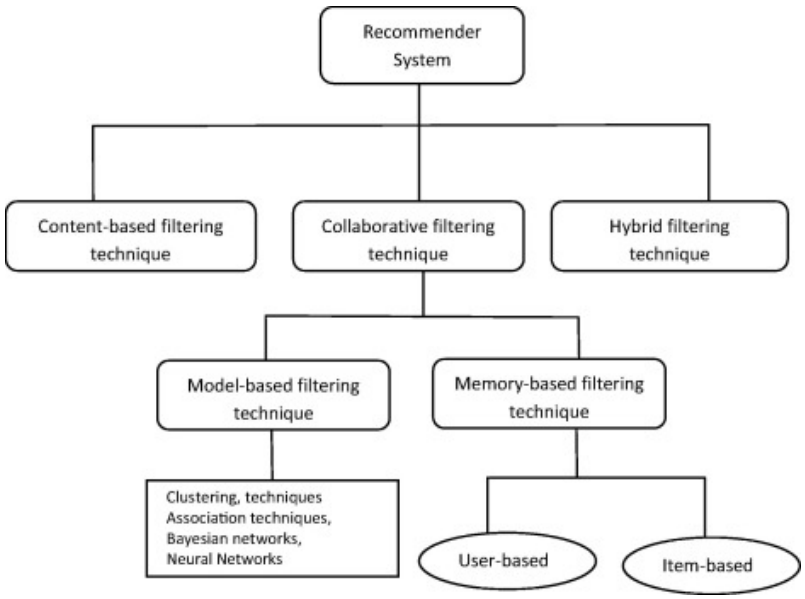

Figure 2: Recommendation Filtering Techniques

5.1 Collaborative filtering (CF)

Collaborative filtering (CF) is a popular recommendation algorithm that bases its predictions and recommendations on the ratings or behavior of other users in the system. The fundamental assumption behind this method is that other users' opinions can be selected and aggregated in such a way as to provide a reasonable prediction of the active user's preference. Intuitively, they assume that, if users agree about the quality or relevance of some items, then they will likely agree about other items — if a group of users likes the same things as Patrick, then Patrick is likely to like the things they like which he has not yet seen.

Two types of algorithms for collaborative filtering have been researched: Memory-based CF and Model-based CF.

5.1.1 Memory-based Collaborative Filtering

A memory-based Collaborative Filtering (CF) approach - also called nearest-neighbor approach -, is said to form an implementation of the "Word of Mouth" phenomenon by maintaining a database of all the users known preferences of

all items, and for each prediction perform some computation across the entire database. It predicts the user's interest in an item based on ratings of information from similar user profiles. Prediction of a specific item (belonging to a specific user) is done by taking the weighted average of the ratings by its similarity toward the user. As a result, ratings by more similar users will contribute more to the rating prediction. Different methods are used in order to compute the similarity between users and items. Among them, Pearson's correlation is probably the most common common one. Other similarity metrics could be mean squared difference or vector similarity, based on the distance measurements such as the cosine or jaccard distances.

Memory-based CF methods have reached a high level of popularity because they are simple and intuitive on a conceptual level while avoiding the complications of a potentially expensive model-building stage. Moreover they are sufficient to solve many real-world problems. Nevertheless. there are also shortcomings. One of them is sparsity. In practice, many memory-based CF systems are used to evaluate large sets of items. In these systems, even active users may have consumed well under 1 percent of the items. Accordingly, a memory based CF system may be unable to make any item recommendation for a particular user. As a result, the recommendation accuracy can be poor. Another problem is cold start. These systems often require a large amount of existing data on a user in order to make accurate recommendations. Memory-based CF methods also present problems of scalability. The algorithms used by most memory-based CF systems require computations that grow exponentially with the number of users and items. Because of this, a typical memory-based CF system with millions of users and items will suffer from serious scalability problems. Finally, they also lack what is traditionally called 'learning'. Since no explicit statistical model is constructed, nothing is actually learned from the available user profile and no general insight is gained.

The weaknesses of memory-based CF systems, especially the scalability and learning issue have led to the exploration of an alternative model-based CF approach.

5.1.2 Model-based collaborative filtering

The motivation behind model-based CF is that by compiling a model that reflects user preferences, some of the problems related to memory-based CF might be solved. This is achieved by first compiling the complete data set into a descriptive model of users, items and ratings. The model can be built off-line over several hours or days. Recommendations can then be computed by consulting the model on on-line data.

Early research on this approach evaluated two probabilistic models, Bayesian clustering and Bayesian networks. In the Bayesian clustering model, users with similar preferences are clustered together into classes. Given the user's class membership, the ratings are assumed to be independent. The number of classes

and the model parameters are learned from the data set. In the Bayesian network model, each node in the network corresponds to an item in the data set. The state of each node corresponds to the possible rating values for each item. Both the structure of the network, which encodes the dependencies between items, and the conditional probabilities, are learned from the data set.

There has also been a suggestion for clustering as a natural preprocessing step for CF. Users and items are classified into groups. For each category of users, the probability that they like each category of items is estimated. Results of several statistical techniques for clustering and model estimation are compared, using both synthetic and real data. Late research has also focused on a rule-based approach for doing model-based CF. This approach applies association rule discovery algorithms to find associations between co-purchased items and then generates item recommendations based on the strength of the association between items.

Model-based CF has several advantages over memory-based CF. First, the model-based approach may offer added values beyond its predictive capabilities, by highlighting certain correlations in the data. Second, memory requirements for the model are normally less than for storing the whole database. Third, predictions can be calculated quickly once the model is generated, though the time complexity to compile the data into a model may be prohibitive, and adding one new data point may require a full recompilation. The resulting model of model-based CF systems is usually very small, fast and essentially as accurate as memory-based methods. Model-based methods may prove practical for environments in which user preferences change slowly with respect to the time needed to build the model. Model-based methods, however, are not suitable for environments in which user preference models must be updated rapidly or frequently

5.2 Content-based filtering

Another common approach when designing recommendation systems is content-based filtering. Content-based filtering methods are based on a description of the item and a profile of the user's preference. In a content-based recommendation system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research. To abstract the features of the items in the system, an item presentation algorithm is applied. A widely used algorithm is the tf–idf representation (also called vector space representation).

To create a user profile, the system mostly focuses on two types of informa-

tion: (i) a model of the user's preference, (ii) a history of the user's interaction with the recommendation system. Basically, these methods use an item profile (i.e., a set of discrete attributes and features) characterizing the item within the system. The system creates a content-based profile of users based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually rated content vectors using a variety of techniques. Simple approaches use the average values of the rated item vector while other sophisticated methods use machine learning techniques such as Bayesian Classifiers, cluster analysis, decision trees, and artificial neural networks in order to estimate the probability that the user is going to like the item. Direct feedback from a user, usually in the form of a like or dislike button, can be used to assign higher or lower weights on the importance of certain attributes (using Rocchio classification or other similar techniques).

A key issue with content-based filtering is whether the system is able to learn user preferences from users' actions regarding one content source and use them across other content types. When the system is limited to recommending content of the same type as the user is already using, the value from the recommendation system is significantly less than when other content types from other services can be recommended. For example, recommending news articles based on browsing of news is useful, but would be much more useful when music, videos, products, discussions etc. from different services can be recommended based on news browsing.

5.3 Hybrid filtering

Hybrid filtering technique combines different recommendation techniques in order to gain better system optimization to avoid some limitations and problems of pure recommendation systems. The idea behind hybrid techniques is that a combination of algorithms will provide more accurate and effective recommendations than a single algorithm as the disadvantages of one algorithm can be overcome by another algorithm. Using multiple recommendation techniques can suppress the weaknesses of an individual technique in a combined model. The combination of approaches can be done in any of the following ways: separate implementation of algorithms and combining the result, utilizing some content-based filtering in collaborative approach, utilizing some collaborative filtering in content-based approach, creating a unified recommendation system that brings together both approaches.

6 Approaches used in the thesis

It is not the purpose of this paper to comprehensively explain all algorithms used in the competition. However, a short explanation of the methods we used seems appropriate for a proper understanding of our strategy.

6.1 Exploring the performance of decision-tree algorithms

As a first approach, in order to have a notion of our baseline performance, we implemented XGBoost, a boosting decision-tree classifier known for its flexibility and predicting power. Decision-tree methods such as XGBoost, Gradient Boosting or Random Forests, as their name suggests, are based on decision-trees. The key characteristic of this group of algorithms is that they recursively split the data by the values of single variables that maximize the 'isolation' of 0's and 1's on our target variables. Decision-trees can be easily understood with an example. Let's say we want to predict whether an user will listen a song. We take K features and run a decision-tree algorithm on it. At each iteration, the algorithm splits the data in two parts for each of the features at the point that best 'isolates' the 1's from the 0's. When the algorithm has finished, it takes the majority vote (0 or 1) on every leave of the tree and returns a vector of predictions. The differences between all the different decision tree algorithms lie on the way they select the variabes to do the splits, perform the splits and combine the resulting predictions.

In particular, XGBoost is a bit different of a Random Forest, in which several trees are run using only parts of data (for each tree it uses a subset of the data -bagging- and at each step it takes randomly some features to be considered) and the predictions of the different trees are averaged or a majority of vote is taken. XGBoost is a Boosting algorithm, which means that each new tree trained has aims to correct the error of the previous tree and then ensemble (average) the predictions of the trees.[1]

XGBoost had a good performance, but it was significantly lower than potentially achievable scores - as shown by the Kaggle leaderboard. That was expected, because tree-based method are not the most appropriate to tackle a recommendation problem. Hence, we decided to move into algorithms specifically suited for them. Nevertheless, we continued training different configurations of XGBoost models for future ensemblings in order to find the best setting.

6.2 Implicit Collaborative Filtering

One of the challenges of recommender systems in the wider commercial world is that one rarely has explicit ratings data (Netflix strongly encourages users to rate movies and leverage the tech built around those ratings). Instead, user-item interactions like clicks, purchases, song listens (or fast-forwards), etc, are used as a proxy to indicate a preference or distaste for a particular item (possibly very weakly). If a user listens to a song only once, maybe it indicates that they didn't like the song. However, maybe they were busy or weren't paying attention and actually would like to hear the song again. If a user listens to a song 100 times, it is a safe bet that they like the song. But it is hard to draw a line between unknown preference and known preference.

This kind of indirect information about user-item preferences is known as implicit feedback, and many smart people have thought long and hard about how to deal with it. We won't delve much into the modeling of implicit feedback in today's post. It turns out to be relatively effective to model a user's preference on a scale of 0 (bad) to 1 (good) instead of modeling the user's raw number of interactions (or whatever number is actually being measured). We can interpret the number of user-item interactions (song listens, for example) as a measure of our confidence in our model's prediction for the user's preference of the item.

In our online competition, since we only know whether an user has listened to a particular song or not, implicit collaborative filtering is one of the approaches that we used. Basically, we need to understand how the users and the items are related to each other. In order to do so, we apply matrix factorization techniques. Often, matrix factorization is applied in the realm of dimensionality reduction, where we are trying to reduce the number of features while still keeping the relevant information. This is the case with principal component analysis (PCA) and singular value decomposition (SVD). In our case, we used Alternative Least Squares, which transforms a large matrix of user/item interactions in order to find out the latent (or hidden) features that relate them to each other in a much smaller matrix of user features and item features. That's exactly what we will do in Matrix Factorization via Alternative Least Squares (ALS).

As Figure below demonstrates, let's assume we have an original ratings matrix R of size MxN, where M is the number of users and N is the number of items. This matrix is quite sparse, since most users only interact with a few items each. We can factorize this matrix into two separate smaller matrices: one with dimensions MxK which will be our latent user feature vectors for each user (U) and a second with dimensions KxN, which will have our latent item feature vectors for each item (V). Multiplying these two feature matrices together approximates the original matrix, but now we have two matrices that are dense including a number of latent features K for each of our items and users.

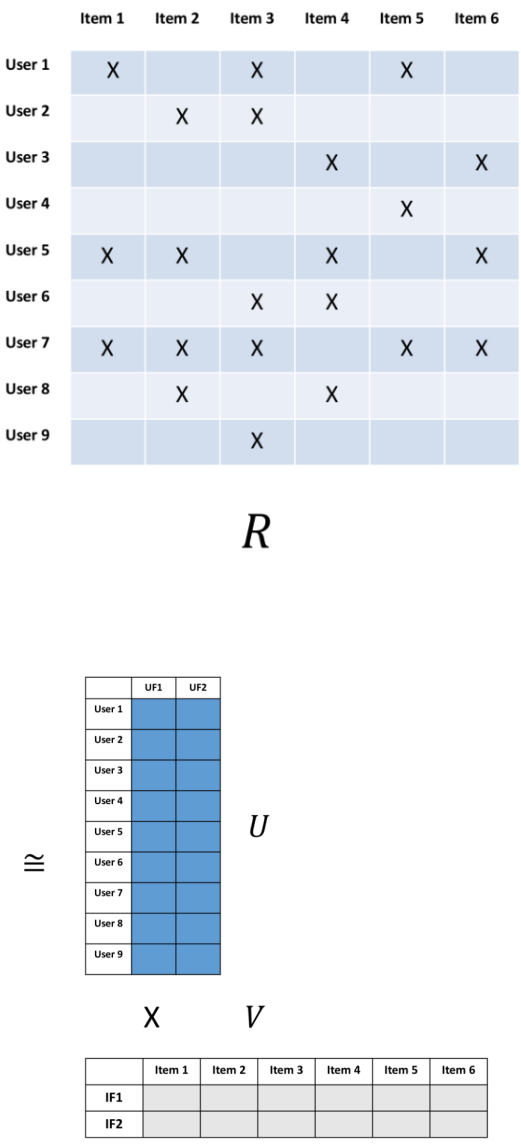

Figure 3: Matrix Factorization

In order to solve for U and V, we could either utilize SVD (which would require inverting a potentially very large matrix and be computationally expensive) to solve the factorization more precisely or apply ALS to approximate it. In the case of ALS, we only need to solve one feature vector at a time, which means it can be run in parallel! (This large advantage is probably why it is

the method of choice for Spark). To do this, we can randomly initialize U and solve for V. Then we can go back and solve for U using our solution for V. Keep iterating back and forth like this until we get a convergence that approximates R as best as we can.

After this has been finished, we can simply take the dot product of U and V to see what the predicted rating would be for a specific user/item interaction,even if there was no prior interaction. This basic methodology was adopted for implicit feedback problems in the paper Collaborative Filtering for Implicit Feedback Datasets by Hu, Koren, and Volinsky. We haved used this paper's method for our dataset.

6.3 Final approach

Given that the performance of the previous algorithms was not as high as expected, we ended up moving to Factorization Machines (see below), which gave us the highest performance. As a final approach, we used the collaborative filtering and factorization machine's prediction vectors as features, which we fed into an XGBoost model that we had previously tuned. Additionally, we ensembled the prediction from this XGBoost model with a previous prediction produced by a different instance with different settings of another XGBoost model. To better account for the quality of the predictions, we weighted them with the performances they obtained as singles predictions in the public leaderboard.

The underlying logic of ensembling methods is that some classes of algorithms outperforms the rest at predicting different groups of users. For instance, linear methods might be the best at predicting users with numerous records, whereas decision-tree based methods might show higher performance with users with less records. Indeed, in our case we had three different types of algorithms: kernel methods (factorization machines), decision-tree algorithms (XGBoost), and collaborative filterning methods. The degree of variety of methods - as well as the disimilarities between them - made us think that ensembling methods would work particularly well in our case. Indeed, the best prediction turned out to be the one that used the explained ensembling, and the one that allowed us to qualify for the final of the competition.

7 Factorization Machines

The core algorithm used in the competition and the one that made us get to the final was Factorization Machines (FM). This algorithm is a generic approach that allows to mimic most factorization models applying unsupervised feature engineering. This model is particularly good for prediction in recommender systems when there is context information as in our case. The model is able to pick interactions between the different variables (feature engineering) even in cases

dealing with a lot of sparse data. [8] FM can be compared with SVM with polynomial kernel [5], however the big advantage of FM is their ability to perform well in sparse data, where SVM tend fail. Also FM models do not depend on support vector (training observations) like SVM and the number of parameters to estimate on the FM model is lower, with some dependencies between them, while in SVM all the parameters are learnt.

Let us denote the target variable $Y \in \Re^n$ and the design matrix as $X \in \Re^{n \times p}$ (n number of observations, p number of features), with $x_i \in \Re^p$ a single observation. Factorization machines model all the interaction between features up to order d. In our case we will only consider up to the second order interactions, so $d = 2$ and so our model will be

$$y(x) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{j,f} v_{j'f}, \tag{1}$$

where k is the dimensionality of the factorization (a parameter that can be set on the algorithm) and then the model parameters are $\theta = (w_0, w_1, \ldots, w_p, v_{1,1}, \ldots, v_{p,k})$. Observe that a FM with dimensionality k there are $1+p+kp$ parameters, linear with respect to the number of features p and dimensionality k. Adding this dimensionality parameter force the rank of the pairwise interactions low, which is important to avoid overfitting due to a growth of parameters and allows to estimate reliable parameters even when the data is very sparse (like it is in the case of most recommender systems). [8]

Since FM are so general, it has been proved that they can mimic most of the current state of the art models that are being used to approach specific problems, being able to treat categorical and continuous variables mixed [5]. Because of this flexibility we thought that FM would be very appropriated for the challenge we had to face, and so it was.

As in all the algorithms used for prediction, a loss function that measures how good or wrong is the predicted value has to be defined and minimized with respect to the parameters. In the challenge the loss function was the Area Under the Curve (AUC) score. The AUC score is based in the Receiver Operating Characteristic curve (ROC), which represents a ratio between the True Positive Rate and the False Positive Rate at different probability thresholds. The AUC score, as its name suggests, represents the area under the ROC curve. However, in the implementation by Rendle that we used (libFM) it was not possible to choose the loss function to be optimized, the one used in the library is the negative logistic loss function $\mathcal{L}(y, \hat{y}) = -\ln(\sigma(y\hat{y}))$, where $\sigma(x)$ is the sigmoid (logistic) function. There is a possibility to add a L2 regularizer in the loss function to penalize complexity and avoid overfitting, but in our case we did not need it.

To minimize the loss function of the model different algorithms have been

proposed, the ones that the library used (libFM) support are three typically used to optimize Machine Learning methods. They are Stochastic Gradient Descend (SGD) [5], Alternating Least Squares (ALS) [6] and Markov Chain Monte carlo MCMC [7].

- SGD is the most famous algorithm to optimize lost functions in Machine Learning and was the first one to be presented by Rendle in the first paper on FM. For SGD there are a lot of parameters that have to be tuned. The learning rate, which is the most important to assure convergence; the regularization parameters, which are used to penalize complexity and avoid overfitting, one could put a regularization parameter for each feature but usually the features are grouped (by user, item, context, etc.); and the initialization of the factors of pairwise interactions, which are usually initialized as close to 0 numbers.

- ALS is commonly used in recommendation algorithms (like matrix factorization or collaborative filtering) to optimize the parameters to estimate the target as well as the SGD. In this case there is not learning rate.

- MCMC is a Bayesian technique that generates the distribution of the target by using Gibbs sampling from the posterior distribution conditioned to the hyperparameters. In order to do the Gibbs sampling it is necessary to add a prior for the parameters of the method, in this case a Gaussian prior the mean of each parameter w_i and $v_{i,j}$ of the model and a Gamma prior to the regularization terms. The advantage of MCMC over the rest is that it is not necessary to set as many parameters, only requires the initialization parameter and new parameters for the hyperpriors. However, the initialization parameter can be set to 0 and the hyperpriors' parameters do not have much influence to the optimization, only in the number of steps to converge. Also, a Bayesian approach can lead to better precision.

Since training our model took a lot of time and from the beginning MCMC gave better results than the other algorithms, we used MCMC to make our predictions.

8 Feature Extraction

The variables in the dataset were already very informative, but there was still some information that could be captured and needed some transformations. The most important one was to make sense of the timestamp when the song was presented to the user. This variable tells the exact second in which the song was presented encoded in UNIX time which is the number of seconds that have passed from the 1st of January of 1970. As it was given, the variable had no value, but we could extract from it four important features.

We converted the variable into datetime format and from it we could extract the day, month, year and the hour. We realized that the year was not relevant

because more than 99% of our data was from 2016 and the rest seemed to have a measurement error, because there were very few of them and the ranges went from 1970 to 2015, also they had incongruence with the release date of the album. So, we decided to remove the year variable and those observations that were not from 2016 (only 812 out of 7.5 million observations).

Then, from the new variable day we created a feature that would tell whether it was holiday (or weekend) or workday and from hour of the day we created a feature that would say whether it is morning, afternoon, evening or night. We based the hour range on French daily routines: 6-13 morning, 13-19 afternoon, 19-22 evening, 22-6 night.

Another variable that we created was to reduce the number of classes in the release year. We classified then into decades so the models could pick something from it.

We also created some variables that gave information about how the user had behaved before with the songs. That means, we added a variable telling if the song was seen before by the user, the proportion of times it had been listened. And then a large number of features that summarize the past behavior of the user, the general behavior of the users towards songs, artist, albums and context, and also a summary of the interactions between those variables pairwise. Adding those variables into our model looked promising, but in total we engineered 114 features like that and they were not sparse as the other features. That made impossible to train a FM model with them, so we selected five of them by training ten xgboost models and selecting the ones that showed more times in the top 10, this method could have been improved by regressing only those features with an L1 norm and we wanted to explore it later on if these features added value. When we added them into the FM model we saw an improvement in our cross validation score, so our feeling was that we could have added gradually some more of the features and kept tuning the dimensionality parameter k of the FM model in order to increase performance, however the improvement did not show in the Public Leaderboard score and we let this aside to move forward later. Sadly, the time constrain did not allow us to do it.

9 Evaluation of Recommendation Engines

One of the key elements of model building is being able to monitor the performance of your models. In order to determine which algorithm suits best the problem, as well as assessing the effect of a specific modification in the algorithm, it is important to have a tool that provides reliable measurements of performance. Traditionally, in statistics, multiple tools have been used to evaluate the so-called 'Goodness of fit' of a model. Among these, accuracy or Area Under the Curve (AUC) are two of the most used ones in binary classification. Given that the competition was based on AUC score, we focused on that when

evaluation our models.

In order to compute the AUC score, we need to split the data into two sets, train and test, so we can train our models in the train set and evaluate the quality of our predictions on the test set. A well-known generalization of this process is what is called 'cross-validation'. The full dataset is split into K-folds and, iterating K times over these folds, the model is trained in K-1 folds and tested in the Kth fold. Usually, after K iterations, we compute the sample mean and the variance Cross-validation reduces the risk of overfitting, automatizes the process of evaluation, and lets us to obtain reliable estimates of the actual performance of our algorithm. Cross-validation, however, also has disadvantages. First, one validation of a model requires K iterations. Even though the complexity scales linearly with K, given the size of our data, one training can take around two hours, so linear is not good enough. To overcome this problem, we parallelized the process, which speeded up the evaluation significantly. Furthermore, after some tests, we realized that the AUC score was robust enough in order to modify our algorithm at every iteration. The importance of a reliable cross-validation score in a competition such as the one we were in is very well known, up to the point that every top kaggler in their posts explaining how they won a challenge in Kaggle or giving advice always point out 'Build a good cross-validation and always trust your CV score, never the LB (Leader Board) score'. That is because the Public Leader Board score is computed using only half of the test dataset, relying on that score may easily lead to overfitting.

9.1 Building a reliable cross-validation

Even though in many problems the cross-validation process is relatively straigthforward, in some other cases it can run into multiple problems. The assumption underlying the reliability of cross-validation estimates is that our train-test split replicates the train-test sets that we have in reality. Indeed, applying traditional cross-validation to our problem resulted in a AUC score higher than 0.8, whereas the actual score was around 0.66. With such a disparity of results, any further modification could not be evaluated, and any progress in the model building process would have had to be done blindly. To obtain a reliable estimate of the performance, we had to carefully inspect our data.

The train dataset consists of around seven million records from approximately 19,000 unique users. The test dataset is significantly different to the train dataset in three features: (i) there is one record per unique user (around 19,000), (ii) all songs have been suggested to the user through the 'Flow' service (variable *IsFlow*), (iii) all records come from only four different types of *ContextID* (which represents the process that lead to the user listening to that song) as opposed to 74 different types in the train set. Finally, an additional obstacle in the cross-validation process for our problem is that the design matrix needs to be in a *OneHotEncod*e format. However, if we split the data into train and test sets, most likely each set will have different songs and users, which

will result in different columns and matrices after reformatting it, making the implementation of any algorithm impossible.

To account for these differences we proceed in the following way. First, we separate the data by *IsFlow*== 1 and *IsFlow*== 0. Then, we sample K records per user from the train dataset, creating K different test sets. Next, we remove those records from the original set and split it in K training sets. We append the *IsFlow*== 0 data and the test data to the train set, then we OneHotEncode it and extract the train set again. Note that this way, we OneHotEncode the train and test data altogether and we can easily extract the train data as it will be located at the bottom of the resulting sparse matrix. Then, we save these 2K objects so we avoid carrying out this process every time we need to evaluate our model - eliminating users and reformatting the data could take around 30 minutes per iteration.

9.2 Tuning the parameters

Parameter tuning is a key element in any machine learning challenge. It helps us improve the performance of most algorithms, specially in our case, since both Factorization Machines and XGBoost are specially tunable. The most commonly used technique is probably GridSearch. It basically consists in iterating over lists of parameter values and saving their performance in order to semi-automate the process. Nevertheless, given the size of our dataset, standard GridSearch involved excessive human intervention and slowed-down the process. In order to speed it up, we explored the possibilities of bayesian optimization for parameter tuning. As a first approach, we implemented a markov chain that sampled from a distribution with parameters that gets updated every time a proposal of a new parameter value is accepted by some decision rule. The basic version of the algorithm is described in Algorithm 1.

Algorithm 1. Algorithm used for parameter tuning

Initialize $\theta^{(0)} \sim \mathcal{N}(\mu_0, \sigma_0)$ and $\rho = c$
for i = 0, . . . , T **do**
  Fit model with parameter $\theta^{(i)}$
  Compute AUC score and save it as ρ'
  Sample $\epsilon \sim Unif(\nu, 1)$
  if $\rho'/\rho > \epsilon$ **then**
    $\rho = \rho'$
    Sample $\theta^{(i+1)} \sim \mathcal{N}(\theta^{(i)}, \sigma)$
  end if
end for

where μ_0 is set at the default parameter value, σ_0 is computed as a function of the parameter to be tuned (e.g. 20 for FM's number of iterations), c is some constant higher than 0 that guarantees the acceptance of the first proposal, ν

was fixed at 0.98 to allow for some random exploration within acceptable decreases in performance and σ is a function of the parameter to be tuned and the iteration to ensure initial exploration in the first iterations as well as precise optimization in the last iterations.

As the reader might realize, this algorithm can be considered as an adaptation of the Metropolis-Hastings algorithm, in which we propose a new parameter, evaluate it by a AUC score ratio - instead of a likelihood ratio - and accept or reject the proposal by an *ad-hoc* decision rule. By construction, the parameters will converge to their optimal values after a number of iterations.

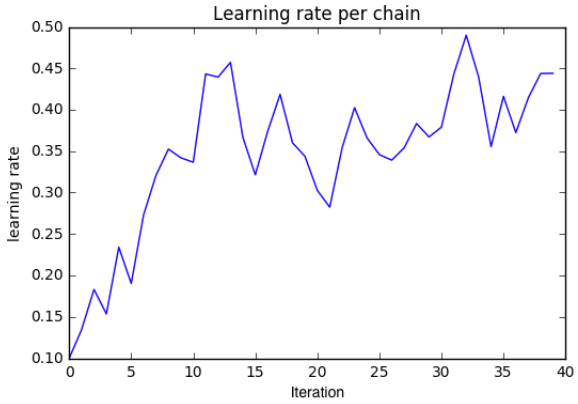

Figure 4: Evolution of one chain tuning the learning rate parameter of SGD in Factorization Machines.

A reasonable concern regarding this algorithm is that the parameters might get stuck in local optimas. After thorough testing, we realized that the distribution of parameters over performance tends to be unimodal, that is, the AUC score increases as we modify the parameter in one direction up to some value after which it starts decreasing. Given this finding, we further speed up the process of finding the mode of the distribution by 'guiding' the random draws in the following way: if an increase in the AUC score is found, we sample from a distribution skewed towards the direction that lead to that improvement. By implementing this process of parameter tuning, we managed to increase the performance of our algorithms in an automated way.

10 Conclusion

So as we see, we have used different approaches to come up with the best prediction for the music recommendation system. We started out with XGBoost with all the features we had. Then we moved on to the recommendation system algorithms like Implicit Collaborative Filtering and Factorization Machines.

Implicit Collaborative Filtering was used as we only had the information of whether an user listened to a song or not, and not whether the user liked it or not.

And Factorization Machine was used as it takes the benefit of the contextual information too, which we had in our case.

And finally, we decided to see the ensemble method, as we hoped the short comings of one method would be overcome by the addition of another. The prediction vectors obtained from collaborative filtering and factorization machine's were fed into the previously tuned XGBoost model. And to account for their accuracy, we weighted them as the performance given by them in public leaderboard.

And the ensemble method actually proved to be the winning method, paving our way towards the final of the competition.

11 Future Work

There are ways we could still improve the model. With more time and resources we could have added more of the engineered features and tuned the parameters of FM according to them. It would have been also good to explore other kinds of simple models (KNN, logistic regression, SVM, etc.) that could have been used as a baseline prediction to add them into the last xgboost model, this way it could have used the strength of the different models in the different patterns of the data to improve the final prediction. Also, it would have been nice to explore more deeply the dataset, because there were a lot of peculiarities to be captured and maybe take advantage to them in order to make separate predictions for some kind of users may would have help. Finally, we wanted to explore an even newer method called Field-Aware Factorization Machines (FFM), which had good results in similar kind of problems, but it was more difficult to understand and required proper tuning and being more careful due to their tendency to overfit easily.

References

[1] Tianqi Chen, Carlos Guestrin *XGBoost: A Scalable Tree Boosting System.* arXiv:1603.02754, 2016.

[2] Yifan Hu, Yehuda Koren, and Chris Volinsky *Collaborative Filtering for Implicit Feedback Datasets.*

[3] Phil Chen and Dan Posch *Collaborative Filtering on Very Sparse Graphs.*

[4] Yehuda Koren, Chris Volinsky, Robert Bell *Matrix Factorization Techniques for Recommender Systems.*

[5] Steffen Rendle. Factorization Machines. *Department of Reasoning for Intelligence The Institute of Scientific and Industrial Research Osaka University, Japan*

[6] S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast Context-aware Recommendations with Factorization Machines. *In Proceedings of the 34th ACM SIGIR Conference on Reasearch and Development in Information Retrieval,2011, New York. ACM.*

[7] Christoph Freudenthaler, Lars Schmidt-Thieme and Steffen Rendle, 2011. Bayesian factorization machines. *In Proceedings of the NIPS Workshop on Sparse Representation and Low-rank Approximation*

[8] Steffen Rendle. Factorization Machines with libFM. *ACM Trans. Intell. Syst. Technol. 3, 3, Article 57 (May 2012)*