Mining Users Playbacks History for Music Recommendations

Alexandr Dzuba and Dmitry Bugaychenko

Odnoklassniki LTD,
Saint-Petersburg State University, Russia
alexandr.dzuba@gmail.com

Abstract. This paper presents a set of methods for the analysis of user activity and data preparation for the music recommender by the example of "Odnoklassniki"[1] social network. The history of actions is being analyzed in multiple dimensions in order to find a number of collaborative and temporal correlations as well as to make the overall rankings. The results of the analysis are being exported in a form of a *taste graph* which is then used to generate on-line music recommendations. The taste graph displays relations between different entities connected with music (users, tracks, artists, etc.) and consists of the following main parts: user preferences, track similarities, artists' similarities, artists' works and demography profiles.

Keywords: music recommendations, taste graph, item similarity.

1 Introduction

In the sphere of music services there are many ways for data extraction which can be useful for generating recommendations, especially in a social networks. Playback history, a user's profile and metadata of the tracks can give an estimate of relations between different entities: users, tracks, artists, etc. *Taste graph* [1] accumulates these knowledge and helps the recomender to solve different tasks. Above mentioned objects and its relations serve as vertices and edges of the graph. It is illustrated in a following way: a user likes an artist, which is similar to another artist, who has recorded a certain track, and each relation can be weighted by a quantitative metric, based on data analysis. Such edges construct numerous chains of edges between the user and unknown tracks.

The paper [3] is proposing the way to make music recommendations by means of *random walk with restarts* on a stochastic graph. The walk starts with an active user being at the graph vertex and continues to the adjacent vertices. The probability of transition along the edge equals its weight. Thus all weights are normalized. In addition, there is a high probability of return to the initial vertex on each walk step. *Steady state probability* distribution of this walk characterizes the relatedness of initial user and vertices within the connected component [5].

[1] www.ok.ru

P. Perner (Ed.): MLDM 2014, LNAI 8556, pp. 422–430, 2014.

However, the application field of this approach is much broader than just its usage for making the track list recommendation. We can use random walks to complement existing music collection with the closest tracks. You can also reorder the given list of songs according to user preferences. Similarity relations between tracks or artists can be used to cluster the tracks.

We present a set of methods, which are used to construct a taste graph in the general purpose social network "OK" (`www.ok.ru`). The main analyzing object is a history of user activity. It allows to estimate the connection between the user and the item vertices, as well as to identify collaborative correlations between the items (tracks and artists). In addition, we analyze a user profile and extract metadata of music files. Still important notice here is that after the metadata extraction a number of postprocessing phases is always required.

2 Taste Graph Construction

Each part of the taste graph is created using a separate algorithm. The user preferences are constructed by aggregating the history of all his playbacks by means of exponential running average. The artist similarities are calculated in multiple steps: representative tracks are selected from the set of all artist's tracks. Then playbacks for these tracks are being aggregated in order to get a user-artist matrix. In order to reduce the impact of the sparsity, this matrix is being converted to an artist-artist matrix by the usage of simple similarity measure. After that the matrix is being iteratively refined. The track similarity matrix is constructed, being based on the amount of playbacks from the same user in the scope of limited time window, which by turn is being discounted by the overall tracks' popularity. The relation between artists and their compositions is calculated by the analysis of the recent activity around the tracks. In order to avoid the cold-start problem, novel items are being additionally boosted. The demography profiles are constructed by averaging user preferences in the scope of a demography group (defined by age, sex and region) and calculating deviations from the system-wide top. The profiles are used to support new users until the time when enough statistical data is collected to reliably construct their own profiles.

The emphasis on different parts is placed due to practical point of view because it allows different parts of the taste graph to be constructed independently and to be combined afterwards. As will be stated below we assume that after determining the edges weights normalization and balancing will be made [1]. All weights are normalized in order to produce a *stochastic system*. Balancing function β is used to manage impact of different factors on the overall result. This function can be changed at the runtime without re-creation of the graph, what increases the flexibility of the system. We supplement the graph with the balancing vertex θ in order to compensate impact of nodes with small amount of outgoing edges. When the amount of edges for $v \in V$ is below limit, an edge from v to θ is being added to take away weights from the existing edges. Next, we describe the basis on which each part of the graph is made and the methods for edge weighting.

All the computations are implemented in Java using Apache Hadoop and Mahout. The daily audience of the `www.ok.ru` is more than 40 million users mainly from the Russian Federation, Eastern Europe and Middle Asia, but despite the huge volume of statistics to be analyzed, taste graph is updated on a daily basis. In combination with the on-line storage for the today's user actions it allows to generate relevant recommendations.

2.1 User Preferences

User preferences consist of two parts, the preference for artists and the preference for tracks, which are based on the calculation of playbacks by the user. However, in practice, we often face the situation when once many times listened artists or tracks which are not popular anymore, are still being invariably placed at the top of recommendations because of the big amount of past playbacks. We use exponential moving average to update our preferences:

$$\begin{aligned} pref_0(u,i) &= plays_0(u,i) \\ pref_t(u,i) &= \alpha \cdot plays_t(u,i) + (1-\alpha) \cdot pref_{t-1}(u,i) \end{aligned} \tag{1}$$

where $plays_t(u,i)$ is the number of artist or track i listenings by user u within a t-th month and $\alpha \in (0,1)$ is a constant smoothing factor. No further action is being done regarding preferences, except the application of the balancing function, which helps to manage the impact of similarities of different types.

2.2 Demography Profiles

The demography profile is used to avoid the cold start problem for new users. When $pref(u) = \{i | pref(u,i) \neq 0\}$ contains not enough elements we use *demographic group* vertex as a start of random walks. Demographic groups $U_1, \ldots U_k$ are disjoint subsets of users U, formed from the profiles with the same values of demographic characteristics (gender, age, region).

$$d_i^p = \sum_{u \in U_p} pref(u,i) \tag{2}$$

In order to extract demographical identity from some group we can compute deviations of top group preferences from the system-wide top swt, where $swt_i = \sum_{u \in U} prefs(u,i)$

$$prefs(U_p) = \frac{top_n(d^p)}{\|top_n(d^p)\|_1} - \frac{top_n(swt)}{\|top_n(swt)\|_1} \tag{3}$$

Above mentioned profiles do not have incoming edges in the graph, therefore as soon as the user gathers enough preferences for independent recommendations, these profiles do not affect the random walks.

2.3 Track Similarities

The standard way of similarities determination is to calculate the collaborative correlations between items based on user ratings. Such methods are established on some similarity measure (usually variations of the Pearson correlation coefficient [2]). Within this approach we need to calculate the metric between hundreds of thousands and millions of tracks, represented by ratings vectors. Algorithms for distributed computation of similar items, such as jobs in the library Apache Mahout, can be time saving, but having a large music catalog and lots of users, demands a powerful machine cluster for timely statistics updating. In order to reduce computational cost we use track *temporal correlations* instead of similarity measure.

Assume $p^u_{i,j}$ is an amount of tracks i and j listenings by the user u in the scope of limited time window. Denote by $p_{i,j}$ the sum of $p^u_{i,j}$ from all users. $p_{i,j}$ reflects how well the tracks are listened together, but this is not enough to conclude that i and j are similar. We need to subtract the popularity of the similar track in order to get pure temporal correlations $t_{i,j}$:

$$t_{i,j} = \frac{p_{i,j}}{\sum\limits_{j=1}^{N} p_{i,j}} - b^i_j, \tag{4}$$

where b^i_j is a *baseline* of the track j adopted to similarity with track i. If $p_j = \sum\limits_{i=1}^{N} p_{i,j}$ and $T^i = \{k | p_{i,k} \neq 0\}$ then

$$b^i_j = \frac{p_j}{\sum\limits_{j=1}^{N} p_j \cdot \mathbf{1}_{T^i}(j)} \tag{5}$$

Thus for calculating similar tracks we normalize rows of $P = \{p_{i,j}\}^N_{i,j=1}$ and $B = \{p_j \cdot \mathbf{1}_{T^i}(j)\}^N_{i,j=1}$ and compute the *temporal correlation matrix* $T = \{t_{i,j}\}^N_{i,j=1} = N - B$.

2.4 Artist Similarities

In case of artists we do not have the problem of big data, what enables the one to use more complex algorithms. Unlike the classical approach [6], where a distance metric between vectors of user ratings is calculated during similar items searching, we start with a vector of artists' common playbacks.

Using the Artist-Artist input instead of the User-Artist one, we can many times increase the vectors' density. This reduces an impact of outliers being revealed in into the input data on the founded collaborative correlations. Also this aggregation eliminates the need to work with the high dimension vectors. The calculations are done in three stages:

1. Pre-filtering of triples *(User, Track, PlayCount)*. We filter the triples with small values of *PlayCount* and users with small amount of statistics. Then we

keep only the reliable tracks. At this point we throw out the tracks which do not pass filtering by density from section 3. Finally we group the playbacks by artist and use the amount of common users as the initial approximation $a^0_{i,j}$ for artist similarity.

2. Iterative calculation of the vector similarity measure between rows of $A^k = \{a^k_{i,j}\}^N_{i,j=1}$. The best results achieved the Euclidean distance and specifically

$$a^k_{i,j} = \frac{1}{1 + \sqrt{\sum_{l=1}^{N}(a^{k-1}_{i,l} - a^{k-1}_{j,l})^2}} \tag{6}$$

After each iteration elements of the main diagonal are artificially increased for a better convergence:

$$a^k_{i,i} = \alpha \cdot \sum_{i \neq j} a^k_{i,j}, \tag{7}$$

where $\alpha \in (0, 1)$ is a constant reducing factor.

3. Similarity lists are filtered by the methods described in section 3.

2.5 Artists' Works

The relations between artists and tracks are selected from the music catalog. Obviously, if the active user vertex is close to the artist vertex, it is reasonable to recommend him the most popular tracks of this artist. But artist's works are not only representing a way to propagate the recommendations of similar artists on the tracks, but are also providing a solution for the cold start problem, so topical for new tracks in a music catalog.

$$w_i = b \cdot \sum_{u \in U} pref s(u, i), \tag{8}$$

where $b > 1$ is a *novelty boost factor*.

Comparing different artists, we can notice large variation in the size of works. In accordance with a stochasticity of the graph, tracks of artists with lots of related songs will be suppressed. We take two steps to avoid this suppression. First, we limit by L the number of adjacent edges with Track-Artist type. Second, for artists with small number of tracks we simulate existence of entry tracks with weights which are close to real. Simulation is gained by adding the edge from the artist j vertex to θ balancing vertex with weight $w_\theta(j)$. Denote by W_j tracks of an artist j:

$$\begin{aligned} w_\theta(j) = &|W_j| \cdot \min_{t \in W_j} a_t \cdot (\frac{1}{|W_j| + 1} + \\ &\frac{1}{|W_j| + 2} + \cdots \frac{1}{L}) \end{aligned} \tag{9}$$

9 is a model of hyperbolic popularity decay. As shown in fugure 1, the rating of most popular tracks decreases like a hyperbola, so we simulate such decay from the lower track's position to L.

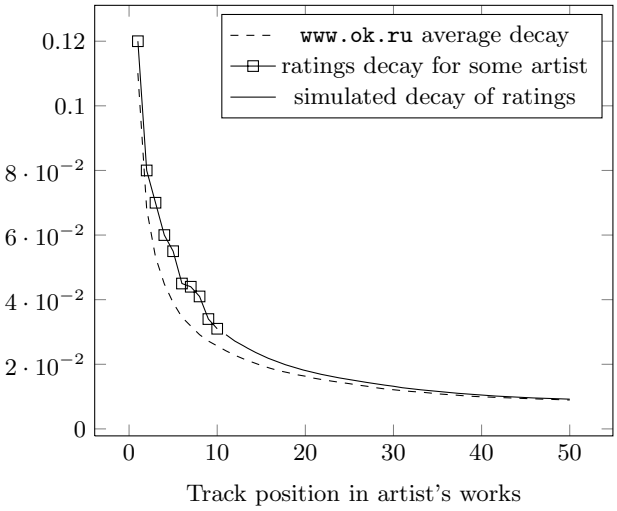

Fig. 1. The weights of Artist-Track edges

2.6 Summary

Along with the graph structure described in this chapter, we have following types of paths from the user to the tracks:

1. $User \to Track \to Track$
2. $User \to Artist \to Track$
3. $User \to Artist \to Artist \to Track$

It is clear that if we have high probability of restarting, the tracks of the second type paths will have much greater impact than tracks reached by the third type paths (for any balancing function). It reflects real relations between entities, but most likely the user is already familiar with works of his favorite artists. In order to increase recommendation novelty we can divide the artist vertex type into two types: an Artist is connected with a Similar Artist who is by turn connected with tracks. So paths in the graph become

1. $User \to Track \to Track$
2. $User \to Artist \to SimilarArtist \to Track$

and the difference between these ways can be controled by balancing function.

3 Similarity Filtration

The methods of outliers filtering for the lists of similar items are presented below. Filtration of similarities with insufficient statistics is close to correlation coefficient shrinkage from [4]. Similarity $s_{i,j}$ between items i and j is discounted like:

$$\hat{s}_{i,j} = \frac{n_{i,j} - 1}{n_{i,j} - 1 + \lambda \cdot min(n_i, n_j)} \cdot s_{i,j}, \tag{10}$$

where $n_{i,j}$ is a number of users who rated both items i and j, n_k is a number of users who rated item k, $\lambda > 0$ is small constant. Note that in different situations, instead of the $min(n_i, n_j)$ arithmetic or geometric mean can be used. Thus the selection function requires additional experiments.

The second filter is used for filtering outliers. To detect outliers we compute sum of similarity values to other elements of items list for each element of this list. In other words, we compute $\tilde{S} = S^2$ where $S = \{s_{i,j}\}_{i,j=1}^{N}$ and filter $s_{i,j}$ with small values of $\tilde{s}_{i,j}$.

The third filter is used to remove similar lists containing some exessively diverse information. As in the case of outliers filtration, we use the similarity to other list items in terms of the subgraph density defined as

$$D = \frac{2|E|}{|V|(|V| - 1)} \tag{11}$$

To check similarity set $s(i) = \{j | s_{i,j} \neq 0)\}$ we compute a density D_i of the G subgraph, induced by $s(i)$ vertices. Too low values of D_i tend to indicate non reliable list, which will have a negative impact on the recommendations.

After all filterings, items with short similar lists appear. In order to compensate their increased impact on recommendation we use edge to zero balancing vertex θ. It simulates the presence of missing edges with weights decreasing linearly from a minimum similarity to a 0.

4 Evaluation

We evaluated the graphs using all our users' preferences. Tens of millions of the preferences was splitted into training set and testing set. After that we constructed the graphs from our similarities and baseline similarities. All the elements of the testing set assumed as relevant when calculating recall-precision curves (RPC). Recommendation lists for the users are being compared here. The lists are generated by using random walks with restarts on the graphs [1].

As a baseline for track similarities, we used a measure from [4], including items' baselines and shrunk correlation coefficient. Figure 2 demonstrates good quality of the method, which has been proposed in section 2.3.

We have chosen the data provided by Last.fm API[2] to evaluate similar artists. The graphs are constructed on the base of various Artist-Artist edges and identical Artist-Track edges. Results are shown in figure 3.

[2] `www.lastfm.ru/api/show/artist.getSimilar`

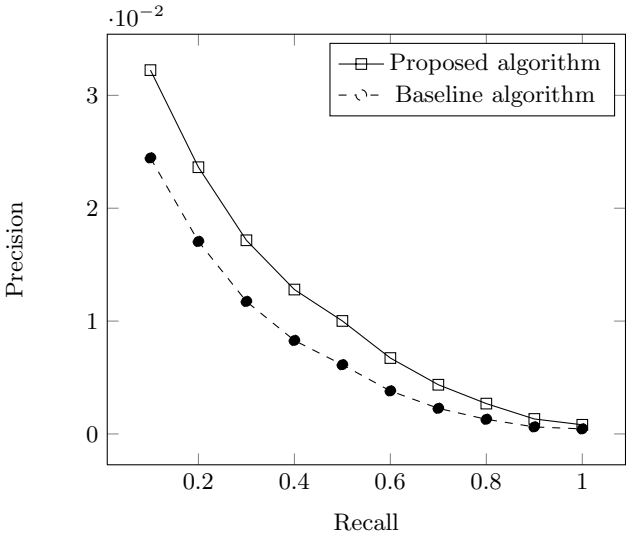

Fig. 2. RPC of the similar tracks

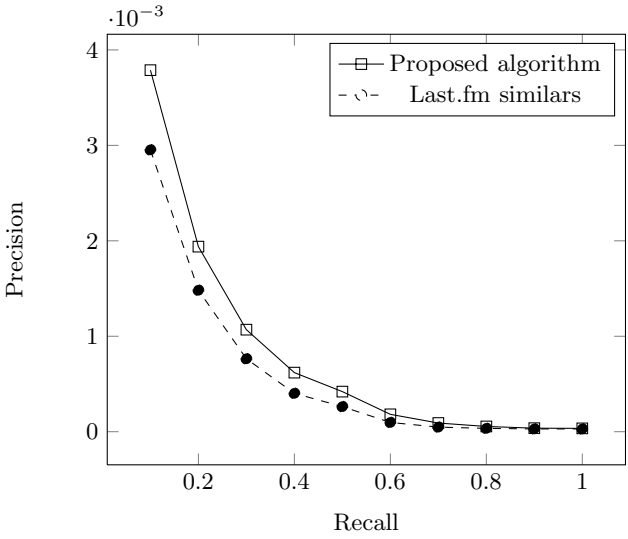

Fig. 3. RPC of the similar artists

Online experiments confirm the decision about quality. After switch from the main graph to the baseline graph, users' activity reduced significantly within the range of 10%. Thereby all the suggested methods demonstrate the ability to recommend relevant content with the quality of state-of-art recommendation methods.

5 Conclusion and Future Work

In this paper we presented the set of methods concerning data analysis, used for evaluation of connections between entities of a music service. The proposed approach allows to calculate these relations independently and to combine them in a taste graph, which provides online recommendations and personalization. At the same time, computational costs for daily graph construction and its updating can be considered to be rather small.

In future we are going to continue working on the taste graph and to pay more attention to the following aspects:

Demographical Adjustments. Large demographic groups can spread collaborative correlations affecting the recommendations of other groups throughout the whole system.

Integration of Other Collaborative Data. Introduction of SVD latent factors to the graph can reduce its size and can make it possible to supplement it with the relations between users without considerable overheads.

References

1. Bugaychenko, D., Dzuba, A.: Musical recommendations and personalization in a social network. In: Proceedings of the 7th ACM Conference on Recommender Systems, pp. 367–370. ACM (2013)
2. Desrosiers, C., Karypis, G.: A comprehensive survey of neighborhood-based recommendation methods. In: Recommender Systems Handbook, pp. 107–144 (2011)
3. Konstas, I., Stathopoulos, V., Jose, J.M.: On social networks and collaborative recommendation. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, pp. 195–202. ACM, New York (2009)
4. Koren, Y., Bell, R.: Advances in collaborative filtering. In: Recommender Systems Handbook, pp. 145–186 (2011)
5. Lovász, L.: Random walks on graphs: A survey. Combinatorics, Paul erdos is eighty 2(1), 1–46 (1993)
6. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web, pp. 285–295. ACM (2001)