Training a Playlist Curator Based on User Taste

AMEL AWADELKARIM, Computational and Mathematical Engineering, Stanford University
KEVIN COELHO, Computer Science, Stanford University

Playlist curation is time consuming and difficult with few tools available for avid music listeners. Spotify's playlist recommendations rely on song similarity (collaborative filtering), but these recommendations lack the novelty and creativity of an individual creator's taste. In this paper, we propose a playlist classifier, incorporating audio feature and genre data on each track from Spotify's public API. Numerical features include Spotify's audio features, "danceability", "energy", "tempo", etc. Categorical features include genre tags and artist names. For categorical variables, we experimented with different encodings including 1-hot vectors, word2vec, node2vec, and GLOVE. Among the tested models, a neural network with 1 hidden layer achieves the highest measured test accuracy on our "toy-dataset" of 82%.

Additional Key Words and Phrases: machine learning, deep learning, neural network, SVM, regression, feature extraction, audio & music, playlist, classification, Spotify

ACM Reference Format:
Amel Awadelkarim and Kevin Coelho. 2018. Training a Playlist Curator Based on User Taste. 1, 1 (December 2018), 5 pages.

1 INTRODUCTION

Humans are generally good at categorizing and organizing music. We may create novel playlists containing songs that may seem very dissimilar, according to any baseline similarity measures. It is this novelty and deeper level of understanding that we attempt to learn in this work: "How do users put together novel playlists?"

Music listeners create playlists based on a multitude of strategies and intents. Many users put similar sounding songs together, some make playlists solely from one artist, others generate heterogeneous mixes spanning multiple genres, eras, and soundscapes. Clearly, the problem of *playlist classification* increases in difficulty as playlist moods become less concrete and/or separable. To tackle this problem, we will be using Spotify's API to gather audio, artist, and genre data for selected playlists. With a more carefully curated and specialized fingerprint of each playlist, we hope to teach an algorithm to understand what makes that user and their playlists special.

Problem Statement. Given a set, K, of unfinished playlists, and a set of unclassified songs, S, can we sort song $s \in S$ into the best playlist $k \in K$, or suggest that a new playlist be created? The input to our classifier is a track vector, $x^{(i)} \in \mathbb{R}^n$, where n represents the number of features. For most tests in this paper, $n = 2105$, 10 of which are audio features, and the remaining 2095 is a one-hot vector of the song's genre tags (section 3). Depending on the experiment, the number of samples, m, corresponds to the sum of the playlist lengths in the dataset. For example, in the toy dataset (section 3), $m = 1044$. Each song is labeled with an integer, $y^{(i)} \in [C]$, representing one of the C output classes (playlists). In the toy case, $C = 13$.

Authors' addresses: Amel Awadelkarim, Computational and Mathematical Engineering, Stanford University, ameloa@stanford.edu; Kevin Coelho, Computer Science, Stanford University, katc@stanford.edu.

2 RELATED WORKS

In general, we would describe our problem space as "non-radio playlist continuation" to distinguish from real-time radio-based solutions. There are a few core concepts in our approach. First, we attempt to model the "theme" of a playlist with minimal assumptions about the user. Second, the model should be scalable and generalizable to any user and any song on a service (we use Spotify). Third, the model should be novel and specific to each user. These concepts are informed by three fundamental pieces of literature in the space which conclude the following:

- Individuals apply a wide variety of methods and reasoning for how and why their playlists are created [Cunningham et al. [n. d.]]
- Some types of music recommendation algorithms are infeasible at scale [Whitman 2012]
- Song similarity does not equate to user satisfaction with a playlist and current literature is less user centric than it could be [Ha Lee 2011]

Following this, we chose not to use the widely cited "Million Song Dataset" (MSD)[Bertin-Mahieux et al. 2011] since it would limit our scope of application to songs in that set. Related works have suggested a variety of algorithms including:

- Model playlists using random walks on a hypergraph of song nodes [Mcfee and Lanckriet 2018]
- Neural networks [Vall et al. 2017]
- Collaborative filtering [Vall et al. 2017]
- K-means [Lin et al. 2018]
- Hybrid collaborative / content-feature methods [Vall et al. 2017; Vall and Widmer 2018]

Drawbacks to some of these implementations include the "out of set" problem - if a song has not been seen by the model, predictions are poor [Mcfee and Lanckriet 2018]. Other weaknesses include lack of generalizability to individual users due to use of MSD data [Mcfee and Lanckriet 2018; Vall et al. 2017] or extremely large number of playlists [Mcfee and Lanckriet 2018; Vall et al. 2017]. Unsupervised methods naturally suffer from producing results that may not align with users' taste [Lin et al. 2018].

Especially due to large data, these implementations would not be fully appropriate for our application, however they introduce important core concepts and algorithms that we have borrowed such as

- Hybrid feature sets (both collaborative filtering and content-based features) [Vall et al. 2017; Vall and Widmer 2018]

- Segmentation using content-based features [Lin et al. 2018] or emotional perception / tagging [Bohra et al. 2015]
- Pre-selecting music a user likes based on their "library" [Bohra et al. 2015; Lin et al. 2018]
- Neural networks to learn playlist-song membership relations [Vall et al. 2017]
- Better evaluation metrics [Vall and Widmer 2018]
- Interactive user interfaces for feedback [Vall and Widmer 2018]
- Vector representations of graphs [Grover and Leskovec 2016]
- Modeling co-occurence [Goldberg and Levy 2014; Pennington et al. 2014]

The biggest difference between our approach and current literature is our small-data approach, training models on a per-user basis, instead of using many thousands of playlists across many users. We believe this makes our model more novel and user-centric. Using features that are available for every song and not using MSD data makes our approach more scalable and generalizable to arbitrary songs and users, and also more robust to the "out of set" problem. For these reasons, we believe our approach to be a significant step towards better playlist curation at an individual level.

3 DATASET & FEATURES

For our first study, we started with data we perceived to be most separable, a "toy set" of $C_1 = 13$ Spotify-curated playlists:

- Dance Rising
- All The Feels
- Have a Great Day!
- Kitchen Swagger
- Swagger
- Sad Vibe
- Afternoon Acoustic
- Tender
- Jazz Vibes
- '90s Baby Makers
- Celtic Punk
- Country by the Grace of God
- Spread the Gospel

for a total of $m_1 = 1044$ tracks. The first experiment is an idealized, fully-supervised setting. We also conduct a second study, generalizing the most successful model on real user data, to truly challenge the models.

For each playlist, we used Spotify's API to pull tracks, audio features per track, and genre tags per artist in the dataset [aud 2018]. Track audio features include the following:

- Danceability
- Energy
- Key
- Loudness
- Mode
- Speechiness
- Acousticness
- Liveness
- Valence
- Tempo

Spotify has developed 2095 genre tags, spanning from broad, ex. "pop" or "r&b", to extremely granular, ex. "australian dance" and "big room". These audio features and genre tags are created with Spotify's internal machine listening and analysis capabilities, acquired mostly from EchoNest [Whitman 2012]. We presume that Spotify's playlist-curation algorithm relies heavily on these features. To handle categorical data, such as these, we appended one-hot vector representations of the genre tags onto each track, and plan to incorporate NLP methods like `node2vec` and `word2vec` to make better use of such tags (section 7).

Fig. 1. Ratio of audio features to genre features for a sample track, $x^{(i)}$.

As above mentioned, we also conduct a study on $C_2 = 116$ of our friends playlists, broadly representing unique moods/styles, for a total of $m_2 = 5721$ tracks. As the end goal is to build a tool for avid music listeners, it is only right that we ultimately test our successful models in a real-world deployment.

For both experiments, we split the dataset 80/20, 80% train and 20% test. In the future, we may explore data augmentation techniques to expand the size of these datasets.

Most of the audio features, ex. "danceability", "energy", and "speechiness", are measures between [0,1]. However, "key" takes on an integer value $\in [0, 11]$, "loudness" is a float $\in [-60, 0]$ measuring average decibel (dB) reading across a track, and "tempo" is a float representing beats-per-minute (BPM). As such, we explore the effects of applying the standard preprocessing step of subtracting the mean and scaling by the variance of each feature. This ensures that the relative scale of these features do not negatively skew the results. A summary of our results is in section 5.

4 METHODS

The models we elected to compare are perceptron regression, with *one-versus-all* binary classification for each class, various SVMs (polynomial, sigmoid, and RBF kernels), and two neural network architectures.

The perceptron update rule, given a sample prediction $\hat{y} = h_\theta(x^{(i)})$, is given by

$$\theta := \theta + \alpha(y^{(i)} - \hat{y}) \cdot x^{(i)}$$

where

$$h_\theta(x) = g(\theta^\top x) = \begin{cases} 0, & \text{if } \theta^\top x < 0 \\ 1, & \text{otherwise.} \end{cases}$$

This update rule comes a generalization of the gradient descent update on the negative log-likelihood for logistic regression.

SVMs attempt to find a boundary which maximizes the geometric margin between classes. Specifically, denoting γ the geometric margin, we look to solve the following constrained maximization problem:

$$\begin{aligned} \max_{\gamma, w, b} \quad & \gamma \\ \text{s.t.} \quad & y^{(i)}(w^\top x^{(i)} + b) \geq \gamma, \quad \forall i \in [m] \\ & ||w|| = 1. \end{aligned}$$

I.e., find the separating boundary, given by w and b, such that all samples are a distance of at least γ away from the boundary. The true algorithm solves the dual form of this primal problem, as the

problem as stated above is non-convex. The library used for the regression and SVMs is scikit-learn, Python's machine learning platform.

Lastly, a neural network can be thought of as a sequence of linear transformations + (non)linear functions applied to an input set of samples, $X \in \mathbb{R}^{m \times n}$, to obtain a prediction vector, $y \in \mathbb{R}^n$.

$$y = a_h(...a_2(W_2 a_1(W_1 X))...)$$

where W_i is a linear transformation, a_i is a (non)linear activation function applied element wise to its argument, and h denotes the number of hidden layers. The neural network attempts to learn a weighting of the input features, W_i's, that best classifies the output. The neural networks were implemented in PyTorch.

Though we test 6 models/architectures in this work, we have elected to focus most of our efforts on the neural networks to solve the problem. It is difficult, sometimes even for humans, to discern exactly what holds a playlist together. Furthermore, the relationship between song features and playlists will not lend itself well to geometric separation in a feature space, further complicated by the fact that one data point (song) may have multiple labels (belong to multiple playlists). The complexity of this problem indicates that less complex algorithms may not be suited to solving the problem effectively.

5 RESULTS

For all tests below, we took *test accuracy* to be our primary metric of success.

$$\text{TA} = \frac{\text{\# correctly classified}}{\text{total \# of samples}}$$

Many models achieved high training accuracy, but unless the test accuracy matched this level, the model was likely overfitted.

5.1 Regression/SVMs

We compared perceptron with polynomial, sigmoid, and RBF kerneled SVMs, with and without the preprocessing step of rescaling the features, on the toy dataset (13 Spotify-curated playlists). We performed k-fold cross validation with $k = 5$. The results are summarized in Table 1

	Scaled	**Unscaled**
RBF SVM	0.77(±0.05)	0.25 (± 0.04)
Sig SVM	0.74 (± 0.07)	0.09 (± 0.04)
Poly SVM	0.17 (± 0.02)	0.48 (± 0.05)
Perceptron	0.76 (± 0.05)	0.13 (± 0.10)

Table 1. Test Accuracies. Trained and tested with audio feature data + one-hot genres.

Without performing the preprocessing step, the accuracy of the results takes a significant hit. On preprocessed data, RBF kerneled SVM reliably performed the best (achieved the highest test accuracy). Tuning the penalty parameter on the error term, we obtain a final test accuracy of 0.80 ± 0.05. The precision, recall, f-score, and support results are tabulated in Table 2.

Playlist	**A**	**B**	**C**	**D**
"Swagger"	0.65	0.62	0.64	76
"Spread the Gospel"	1.00	0.93	0.96	40
"'90's Baby Makers"	0.74	0.79	0.76	47
"Tender"	0.75	0.36	0.49	33
"Have a Great Day!"	0.69	0.83	0.75	100
"Dance Rising"	0.85	0.94	0.89	99
"Sad Vibe"	0.71	0.83	0.77	42
"Afternoon Acoustic"	0.79	0.80	0.80	76
"Kitchen Swagger"	0.49	0.43	0.46	75
"All The Feels"	0.85	0.88	0.86	65
"Jazz Vibes"	0.92	0.92	0.92	118
"Celtic Punk"	1.00	0.94	0.97	49
"Country by the ..."	0.95	0.84	0.89	49

Table 2. Results of tuned RBF-kerneled SVM on test set. **A**: Precision, **B**: Recall, **C**: F1-score, **D**: Support (number of songs in playlist).

Note that the playlists "Celtic Punk" and "Spread the Gospel" achieved 100% precision by the tuned SVM – all corresponding tracks in the test set were correctly classified into these playlists. This demonstrates the efficacy of these methods on more unique, genre specific, lists. "Kitchen Swagger", on the other hand, is a harder playlist to classify, as it does not match any one genre.

5.2 Neural network

We considered many parameters of our network: number of hidden layers, activation functions (sigmoid, relu, identity, softmax/log-softmax), with vs. without L_2 regulation, number of iterations, stochastic/batch/full gradient descent, loss function (MSE vs NLL), etc. We ultimately found that a neural network, minimizing NLL loss via full gradient descent, with one hidden layer of C neurons, and a LogSoftmax output layer performs best on the toy set. Figure 2 illustrates the increase in training and test accuracies over multiple epochs of the dataset. The gap between these accuracies is small, indicating a low-bias model.

Table 3 compares the training and test accuracies between one hidden layer with C neurons and two hidden layers with $2 \cdot C$ and C neurons, respectively. We see that the optimal neural network achieves 82% test accuracy on the toy-dataset, an improvement on our RBF SVM result.

\# Hidden layers	**Train**	**Test**
2 (identity + sigmoid)	0.91	0.77
1 (sigmoid)	0.89	0.82

Table 3. Train and test accuracy on the toy dataset for various architectures (activation functions in parentheses).

With the architecture finalized, we tested on a handful of real users, again training on 80% of their playlists, and testing on the remaining 20%. The results are summarized in Table 4.

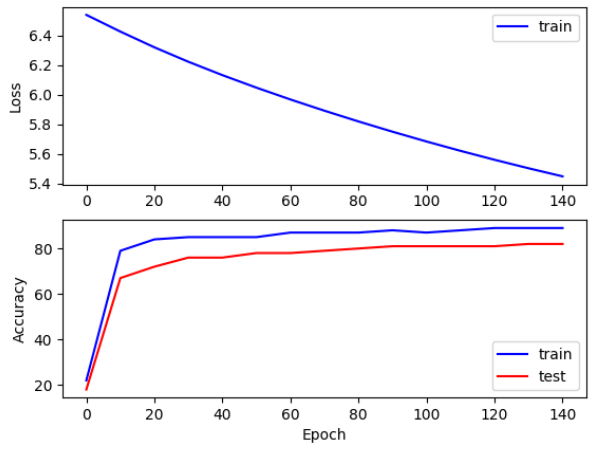
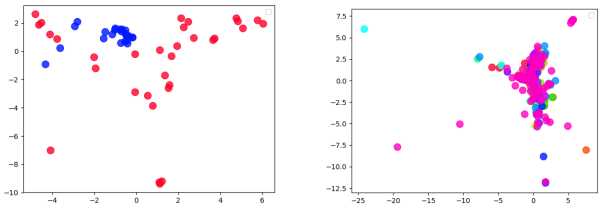


Fig. 2. Results of training neural network with one hidden layer (sigmoid activation) and LogSoftmax output layer, minimizing NLL loss with L_2 regularization over 150 epochs.

User	Train	Test
Jacob	1.00	0.90
Kevin	0.94	0.78
Miz	0.62	0.40
Myles	0.64	0.15

Table 4. Train and test accuracy on various users using final neural network architecture. Jacob's dataset is relatively separable, while Myles' playlists are heavily overlapping in sound.



(a) Jacob's playlists (b) Myles' playlists

Fig. 3. User playlists, mapped to two dimensions via PCA. We can see that Jacob's set is far more separable than Myles', even in two-dimensions

We see that certain users are more challenging to classify than others, ex. Myles vs. Jacob. Figure 3 illustrates the separability of these users sampled playlists. The selected playlists for Jacob are even separable in two-dimensions, whereas Myles' playlists are clearly more overlapping and similar in sound.

6 DISCUSSION

In some ways, the observed results are not surprising. We have observed that a relatively minimal neural network does a better job at classifying playlists than perceptron and SVMs do. This is understandable as the nature of curating a playlist is rather subjective, and playlists are not always separable. Even so, the neural network struggled against a real user set. There are multiple factors that play into this:

- **Larger labeled datasets.** In many instances, the user data we were training/testing on consisted of only hundreds (sometimes *tens*) of samples. Training on only 80% of this dataset decreases this number further. This is not enough information to train a neural network classifier on, nor to gather meaningful test results on, given our limited features. In the future, we may either select larger user playlists, or consider applying data augmentation techniques to increase the size of our set.
- **Segmentation.** It is easier to classify a song into a playlist when the number of output classes, C, is small. This concept is known as *segmentation* [Bohra et al. 2015], which we have not performed in this study. For users with tens of playlists in the dataset, we have run preliminary experiments only considering up to $C = 5$ of their playlists at a time when classifying. Running segmentation on Miz's playlists, we see an average test accuracy of 0.79 running the RBF SVM over 100 trials, vastly outperforming both train and test scores from our neural network (train accuracy was 0.62). We will continue to explore this in future iterations of our algorithm.
- **Different algorithm.** It may be the case that even neural networks are not the best option for this problem, though literature in the field suggests otherwise [Jannach and Ludewig 2017; Vall et al. 2017]. We plan to test other fully-supervised, multi-class classifier algorithms, such as decision trees.
- **More track features.** We have built a classifier with only audio features and one-hot vectors of genre tags for each track. We are missing vital data, such as artist info, release date, etc (section 7).

7 CONCLUSION & FUTURE WORKS

Playlist curation and music recommendation is an art that trained music analysts and disc jockeys have perfected over decades. With the rise of massive listener data and the wealth of statistical and algorithmic developments, we have begun to see a rise in computer-generated playlists and mixes. At this stage, however, there is massive room for improvement. We have attempted to build a playlist classifier using audio features and genre tag data from Spotify's public API [aud 2018]. The optimal classifier of those tested was a single-layered neural network, achieving a test accuracy of 0.82 on a toy-dataset. Though the results are promising in this case, we have many future directions to explore in optimizing for real users.

Node2Vec. Through Spotify's API, we have access to related-artists data: each artist comes with a list of 20 similar artists. We have gathered this information and built a related-artists graph, where each node represents an artist, and edges link like artists. Using SNAP [Leskovec and Sosič 2016], a public network analysis library built at Stanford, we fed the graph through its node2vec framework, and have computed 128-dimensional artist embeddings. Figure 4 illustrates a two-dimensional visualization of the resulting

vector space via PCA. We believe that the representation successfully captures artist similarity.

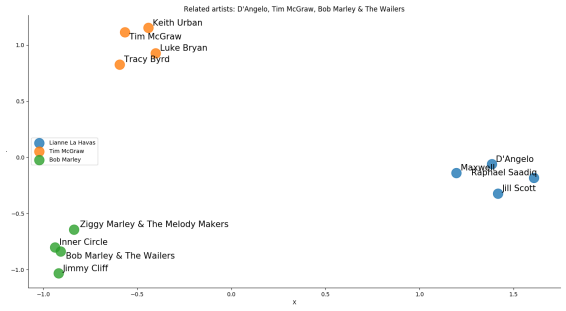


Fig. 4. 2D node2vec data on related artists graph. D'Angelo, Tim McGraw, and Bob Marley and the Wailers are seed nodes, plotting 4 nearest neighbors of each. Isolated clusters demonstrate the lack of similarity in these groups.

These vectors can then be appended to each data point as an additional feature of a track, connecting like-artists.

Word2Vec. Google has published an open-source library for computing vector embeddings of words. In the future, we hope to integrate this into our framework, taking advantage of large free corpuses such as Wikipedia and music website scrapes. Results from EchoNest show that NLP "understanding" of the text around music are highly effective in music classification [Whitman 2012]. This allows us to gain a "cultural understanding" of what the words in our documents mean.

For example, the "`stevie_wonder`" vector may be near the words "`soul`", "`michael_jackson`", "`piano`", or "`motown`" in the vector space. Candidates for `word2vec` representations in our data include artist names, user generated tags (both from our users and from open sources like AcousticBrainz), genre tags, playlist titles, song names, etc.

FastText. Facebook has developed the FastText library for document classification. We hope to try using it with the variety of texts that we hope to collect including the Wikipedia pages for our artists, user-generated tags, genre tags, and a variety of other textual metadata from track information, artist information, and more. In this case, a playlist can be considered a document category and we will attempt to classify song documents as belonging to the document category.

8 MEMBER CONTRIBUTIONS

The data used for this problem comes from Spotify's Web API. Kevin obtained the necessary authorization with Spotify, automated the request process for all playlists, tracks, and feature data, and built a Postgres database from scratch via Sequelize, an object-relational manager (ORM) for Node.js. Once all the data for the toy-set was in the database, the duo implemented the baseline Perceptron/SVM tests on the toy-set using scikit-learn. Amel designed and tested our neural networks using PyTorch & wrote script to get PCA visualizations. Kevin took on advanced data collection, building a related artists graph and computing vector embeddings of each node using `node2vec`.

9 CODE REPOSITORIES

Javascript repo (database initialization): https://github.com/kevin-coelho/playlistr-ml-v1
Python repo (algorithms): https://github.com/kevin-coelho/playlistr-ml-py-v1

REFERENCES

2018. Get Audio Features for Several Tracks. https://developer.spotify.com/documentation/web-api/reference/tracks/get-several-audio-features/

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

T. S. Bohra, V. Kumar, and S. Ganesan. 2015. Segmenting music library for generation of playlist using machine learning. In *2015 IEEE International Conference on Electro/Information Technology (EIT)*. 421–425. https://doi.org/10.1109/EIT.2015.7293429

S. J. Cunningham, D. Bainbridge, and A. Falconer. [n. d.]. "More of an art than a science": Supporting the creation of playlists and mixes. *Proceedings of the 7th International Conference on Music Information Retrieval Conference, ISMIR 2006*.

Yoav Goldberg and Omer Levy. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *CoRR* abs/1402.3722 (2014). arXiv:1402.3722 http://arxiv.org/abs/1402.3722

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. *CoRR* abs/1607.00653 (2016). arXiv:1607.00653 http://arxiv.org/abs/1607.00653

Jin Ha Lee. 2011. How Similar Is Too Similar?: Exploring Users' Perceptions of Similarity in Playlist Evaluation. *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011*, 109–114.

Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. 306–310. https://doi.org/10.1145/3109859.3109872

Jure Leskovec and Rok Sosič. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1.

Diana E Lin, Sampath Jayarathna, and Yu Sun. 2018. *An Application for Automated Playlist Generation From Personal Music Libraries Using Clustering Algorithms and Music Analysis.* Ph.D. Dissertation.

Brian Mcfee and Gert Lanckriet. 2018. HYPERGRAPH MODELS OF PLAYLIST DIALECTS. (12 2018).

Jeffrey Pennington, Richard Socher, and Christoper Manning. 2014. Glove: Global Vectors for Word Representation. *EMNLP* 14, 1532–1543. https://doi.org/10.3115/v1/D14-1162

Andreu Vall, Hamid Eghbal-zadeh, Matthias Dorfer, and Markus Schedl. 2017. Music Playlist Continuation by Learning from Hand-Curated Examples and Song Features. *CoRR* abs/1705.08283 (2017). arXiv:1705.08283 http://arxiv.org/abs/1705.08283

Andreu Vall and Gerhard Widmer. 2018. Machine Learning Approaches to Hybrid Music Recommender Systems. *CoRR* abs/1807.05858 (2018). arXiv:1807.05858 http://arxiv.org/abs/1807.05858

Brian Whitman. 2012. How music recommendation works - and doesn?t work. https://notes.variogr.am/2012/12/11/how-music-recommendation-works-and-doesnt-work/