Steerable Playlist Generation by Learning Song Similarity from Radio Station Playlists.

Conference Paper · January 2009

Source: DBLP

CITATIONS
28

READS
52

4 authors, including:

François Maillet
Datacratic
5 PUBLICATIONS **167** CITATIONS
SEE PROFILE

Paul Lamere
Spotify
29 PUBLICATIONS **1,396** CITATIONS
SEE PROFILE

STEERABLE PLAYLIST GENERATION BY LEARNING SONG SIMILARITY FROM RADIO STATION PLAYLISTS

François Maillet, Douglas Eck, Guillaume Desjardins
DIRO, Université de Montréal, CIRMMT
Montreal, Canada
{mailletf,eckdoug,desjagui}@iro.umontreal.ca

Paul Lamere
The Echo Nest
Somerville, USA
paul.lamere@gmail.com

ABSTRACT

This paper presents an approach to generating steerable playlists. We first demonstrate a method for learning song transition probabilities from audio features extracted from songs played in professional radio station playlists. We then show that by using this learnt similarity function as a prior, we are able to generate steerable playlists by choosing the next song to play not simply based on that prior, but on a tag cloud that the user is able to manipulate to express the high-level characteristics of the music he wishes to listen to.

1. INTRODUCTION

The celestial jukebox is becoming a reality. Not only are personal music collections growing rapidly, but online music streaming services like Spotify [1] or Last.fm [2] are getting closer everyday to making all the music that has ever been recorded instantly available. Furthermore, new playback devices are revolutionizing the way people listen to music. For example, with its Internet connectivity, Apple's iPhone gives listeners access to a virtually unlimited number of tracks as long as they are in range of a cellular tower. In this context, a combination of personalized recommendation technology and automatic playlist generation will very likely form a key component of the end user's listening experience.

This work's focus is on providing a way to generate steerable playlists, that is, to give the user high-level control over the music that is played while automatically choosing the tracks and presenting them in a coherent way. To address this challenge, we use playlists from professional radio stations to learn a new similarity space based on song-level audio features. This yields a similarity function that takes audio files as input and outputs the probability of those audio files being played successively in a playlist. By using radio station playlists, we have the advantage of having a virtually unlimited amount of training data. At the same time, we are able to generalize the application of the model to any song for which we have the audio files. We believe this will be the case in any real-life application we can foresee for the model.

Furthermore, we use the concept of a steerable tag cloud [2] to let the user guide the playlist generation process. Tags [6], a type of meta-data, are descriptive words and phrases applied to any type of item; in our case, music tracks. Tags are words like like *chill*, *violin* or *dream pop*. They have been popularized by Web 2.0 websites like Last.fm, where users can apply them to artists, albums and tracks. The strength of tags, especially when used in a social context, lies in their ability to express abstract concepts. Tags communicate high-level ideas that listeners naturally use when describing music. We tag all tracks in our playlists using an automatic tagging system [7] in order to ensure that they are all adequately tagged. Then, given a seed song, the learnt similarity model is used to preselect the most probable songs to play next, after which the similarity between the user's steerable tag cloud and each of the candidate songs' cloud is used to make the final choice. This allows users to steer the playlist generator to the type of music they want to hear.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of related work in music similarity and playlist generation. Section 3 explains how the radio station playlists data set was collected and assembled. Section 4 presents the creation and evaluation of our new similarity space. Section 5 explains how we propose implementing a steerable playlist generator. Finally, section 6 explores future research avenues.

2. PREVIOUS WORK

An increasing amount of work is being conducted on automatic playlist generation, with considerable focus being placed on the creation of playlists by means of acoustic or meta-data similarity [10–14].

More recently, connectivity graphs derived from music social networks are being used to measure similarity. For example, [5] uses network flow analysis to generate playlists from a friendship graph for MySpace [3] artists. In [4], the authors use Last.fm collaborative filtering data

[1] http://www.spotify.com
[2] http://www.last.fm

[3] http://www.myspace.com

to create a similarity graph considering songs to be similar if they have been listened to by similar users. They then embed the graph into a Euclidean space using LMDS, where similar artists would appear near one another.

Another approach [3] uses a case-based reasoning system. From its pool of real human-compiled playlists, the system selects the most relevant ones in regards to the user's one song query and mixes them together, creating a new playlist.

We are aware of only one other attempt to use radio station playlists as a source of data. In [1] radio station playlists are used to construct a weighted graph where each node represents a song and each arc's weight is the number of times the two songs are observed one after the other. From the graph, the authors are able to infer transition probabilities between songs by creating a Markov random field. Our approach is similar, with the advantage that we can generalize to songs not observed in the training data.

3. CONSTRUCTING THE DATA SET

Our model is trained on professional radio station playlists. For this experiment, we consider a playlist to be a sequence of 2 or more consecutive plays uninterrupted by a commercial break. Suppose a radio station plays the tracks t_a, t_b and t_c one after the other, we will consider $\{t_a, t_b\}$ and $\{t_b, t_c\}$ as two 2-song sequences $\in \mathbb{S}^2$, and $\{t_a, t_b, t_c\}$ as one 3-song sequence $\in \mathbb{S}^3$. We consider the sequences $\{t_a, t_b\}$ and $\{t_b, t_a\}$ as two distinct sequences. The model's output will thus be non-symmetric in regards to the order in which the songs are presented.

The playlist data we used came from two sources which we will cover in section 3.1.

3.1 Playlist sources

3.1.1 Radio Paradise

Radio Paradise [4] (RP) is a free Internet-streamed radio station that defines its format as "eclectic online radio." RP provided us with playlist data including every play from January 1st 2007 to July 28th 2008 (575 days). The data consists of 195,692 plays, 6,328 unique songs and 1,972 unique artists.

3.1.2 Yes.com

Yes.com is a music community web site that provides, among other things, the playlists for thousands of radio stations in the United States. Developers are able to access the playlist data via a free web based API [5] that returns the data in JSON format. One API call allows the developer to get a list of radio stations, either by searching by genre, by name or even by proximity to a given ZIP code. Then, for each retrieved station, the API provides access to that station's play history for the last 7 days. The self-assigned and non-exclusive genres of the available radio stations cover all major musical styles. The stations we used to build our own dataset were not chosen for any particular reason. Rather, we made a few searches with the API by genre until we obtained enough data for our work, that is 449 stations. The proportion of stations' non-exclusive association with the different genres is detailed in Table 1.

Table 1. Proportion of the 449 Yes radio stations associated with each genre. Because the genres are non-exclusive the sum of the percentages is > 100.

Latin	11.2%	Christian	11.4%
Country	20.6%	Hip-Hop	17.2%
Jazz	4.3%	Metal	14.1%
Pop	23.3%	Punk	1.6%
Rock	39.4%	R&B/Soul	13.6%

We used data mined from the Yes API from November 13th 2008 to January 9th 2009 (57 days), totaling 449 stations, 6,706,830 plays, 42,027 unique songs and 9,990 unique artists.

Unlike RP, Yes did not provide any indication of where the commercial breaks were located in the list of plays. We inferred where they were by looking at the interval between the start time of every pair of consecutive songs. As we will explain in section 4.1, we only used sequences made of tracks for which we had the audio files. This allowed us to calculate song length and to infer when commercials were inserted. Specifically, if the second of the two songs did not start within ± 20 seconds of the end of the first one, we assumed that a commercial break had been inserted and thus treated the two songs as non-sequential. This approach is more precise than the method used in [1], where breaks were inserted if the elapsed time between two songs was greater than 5 minutes.

3.2 Putting the data together

Combining all the data yielded 6,902,522 plays, with an average of 15,338 plays per station. As we will explain in 4.1, the features we used as input to our model required us to have access to each of the songs' audio file. Of the 47,044 total songs played in the playlists we used, we were able to obtain the audio files for 7,127 tracks. This reduced the number of distinct usable song sequences to 180,232 and 84,668 for the 2 and 3-song sequence case respectively. The sequences for which we had all the audio files were combinations from 5,562 tracks.

Finally, we did not possess a set of explicit negative examples (i.e. two-song sequences that a radio station would never play). In order to perform classification we needed examples from both the positive and negative class. To address this, we considered any song sequence that was never observed in the playlist as being a negative example. During training, at each new epoch, we randomly sampled a new set of negative examples matched in size to our positive example set. With this strategy it is possible that we generated false-negative training examples (i.e. two-song sequences that we didn't see as positive examples in our data set but that in fact a radio station would play). How-

[4] `http://www.radioparadise.com`
[5] `http://api.yes.com`

ever, since we resample new negative examples after every training epoch, we do not repeatedly show the model the same false-negative pair, thus minimizing potential impact on model performance.

4. SONG SIMILARITY MODEL

4.1 Features

We use audio-based features as input to our model. First, we compute 176 frame-level autocorrelation coefficients for lags spanning from 250ms to 2000ms at 10ms intervals. These are aggregated by simply taking their mean. We then down sample the values by a factor of two, yielding 88 values. We then take the first 12 Mel-frequency cepstral coefficients (MFCC), calculated over short windows of audio (100ms with 25ms overlaps), and model them with a single Gaussian (G1) with full covariance [16]. We unwrap the values into a vector, which yields 78 values.

We then compute two song-level features, danceability [9] and long-term loudness level (LLML) [8]. Danceability is a variation of detrended fluctuation analysis, which indicates if a strong and steady beat is present in the track, while the LLML gives an indication of the perceived loudness of the track. Both of these features yield a single numeric value per song.

These 4 audio features are concatenated to form an 180 dimensional vector for each track.

4.2 Learning models

We formulate our learning task as training a binary classifier to determine, given the features for a sequence of tracks, if they form a song sequence that has been observed in our radio station playlists. If a sequence has been observed at least once, it is considered a positive example. As mentioned above, the negative examples are randomly sampled from the pool of all unseen sequences.

We use three types of learning models in our experiments: logistic regression classifiers, multi-layer perceptrons (MLP) and stacked denoising auto-encoders (SdA). Logistic regression, the simplest model, predicts the probability of a song-sequence occurrence as a function of the distance to a linear classification boundary. The second model, a multi-layer perceptron, can also be interpreted probabilistically. It adds an extra "hidden" layer of non-linearity, allowing the classifier to learn a compact, nonlinear set of basis functions.

We also use a type of deep neural network called a stacked denoising auto-encoder (SdA) [17]. The SdA learns a hierarchical representation of the input data by successively initializing each of its layers according to an unsupervised criterion to form more complex and abstract features. The goal of this per-layer unsupervised learning is to extract an intermediate representation which preserves information content whilst being invariant to certain transformations in the input. SdAs are exactly like neural networks with the exception that they have multiple hidden layers that are initialized with unsupervised training.

In our experiments, the models operated directly on pairs (or 3-tuples in the case of predicting sequences of length 3) of audio features. The input x of our model is thus a vector of length $180 \cdot n$, with $n \in \{2, 3\}$, formed by concatenating the features of each track into a single vector.

We used 75% of our unique pairs/triplets for training, keeping 12.5% for validating the hyper-parameters and 12.5% for testing. We did not perform any cross-validation.

4.3 Similarity evaluation

Measuring the quality of the similarity space induced by the model is not easy and highly subjective. We will first look at its performance on the learning task (4.3.1), and then try to evaluate it in a more qualitative way (4.3.2).

4.3.1 Learning performance

Classification errors for the different models we trained are presented in Table 2. The errors represent the proportion of real sequences that were classified as false sequences by each model, or vice versa, on the test set, for the best combination of hyper-parameters.

While the logistic regression clearly lacks learning capacity to adequately model the data, the MLPs and SdAs have similar performance. SdAs have been shown to outperform MLPs in complex image classification tasks ([17]) but were unable to learn a significantly better representation of the features we are using for this task. This could mean that the feature set was not sufficiently rich or that the task was simply too difficult for the hierarchical model to find any kind of compositional solution to the problem.

Table 2. Classification errors on the test set for the different models we trained as well as a random baseline. SdA-n represents an SdA with n hidden layers.

Model	2-song seq.	3-song seq.
random	50.00%	50.00%
logistic regression	31.73%	21.08%
MLP	8.53%	5.79%
SdA-2	8.38%	5.58%
SdA-3	8.62%	5.78%

4.3.2 Retrieval evaluation

By using the original radio station playlists as the ground truth, we can evaluate the retrieval performance of our model. The evaluation is done using *TopBucket* (TB) [7], which is the proportion of common elements in the two top-N lists.

Constructing the ground truth from the playlists is done as follows. Each 2-song sequence $S_n^2 \in \mathbb{S}^2$ is made up of tracks $\{t_n^1, t_n^2\}$ and has been observed $|S_n^2|$ times. We construct one top list $\mathcal{L}_{t_i} \forall t_i \in \mathbb{T}$, as the set of all sequences S_n^2 for which $t_n^1 = t_i$, ordered by $|S_n^2|$. $\mathcal{L}_{t_i}$ essentially gives a list of all the tracks that have followed t_i ordered by their occurrence count. In the 3-song sequence case, we construct a top list $\mathcal{L}_{\{t_i,t_j\}}$ for pairs of tracks since in

Table 3. Retrieval performance based on the *TopBucket* (TB) measure of our models compared to random, popularity-biased random, acoustic similarity and autotags similarity. Each score represents the average percentage (and standard deviation) of songs in the ground truth that were returned by each model.

	2-song sequences		**3-song sequences**	
Model	TB10	TB20	TB5	TB10
random	0.25%±0.16%	0.58%±1.75%	0.11%±1.45%	0.25%±1.52%
popularity-biased random	1.01%±3.15%	2.19%±3.34%	0.51%±3.17%	0.96%±3.15%
acoustic (G1C)	1.37%±3.80%	2.37%±3.73%	0.63%±3.48%	1.61%±4.01%
autotags (Cosine distance)	1.43%±3.98%	2.34%±3.86%	0.58%±3.34%	2.26%±4.89%
logistic regression	2.47%±5.08%	6.41%±6.40%	0.20%±2.00%	1.16%±3.40%
MLP	16.61%±14.40%	23.48%±13.17%	7.72%±13.92%	20.26%±17.85%
SdA-2	13.11%±12.05%	19.13%±11.19%	7.25%±13.66%	21.74%±19.75%
SdA-3	13.17%±11.31%	18.22%±10.04%	9.74%±18.00%	26.39%±22.74%

practice, a playlist generation algorithm would know the last two songs that have played.

For our experiments, we used the top 10 and 20 elements and 5 and 10 elements for the 2 and 3-song sequence case respectively. The results, shown in Table 3, represent the average number of common elements in the ground truth's top list and each of the similarity models' for every song.

Because most sequences were only observed once ($|S_n^2| = 1$), we were often in the situation where all the sequences in $\mathcal{L}_{t_i}$ had an occurrence of 1 ($\forall S_n^2 \in \mathcal{L}_{t_i} : |S_n^2| = 1$) and the number of sequences in $\mathcal{L}_{t_i}$ was greater than N for a top-N list. Because in such a situation there was no way to determine which sequences should go in the top-list, we decided to extend the top-N list to all the sequences that had the same occurrence count as the N^{th} sequence. In the 2-song sequence case, we also ignored all sequences that had $|S_n^2| = 1$ to keep the top-N lists from growing a lot larger than N. Ignoring as well all the songs that did not have at least N tracks in their top-list, we were left, in the 2-song sequence case, with 834 songs that had an average of 14.7 songs in their top-10 list and 541 songs with an average of 28.8 songs in their top-20 list. In the 3-song sequence case, the top-5 list was made up of 1,181 songs with a 6.6 top songs average and the top-10 list was composed of 155 songs with an average of 13.8 top songs.

We compared our model's retrieval performance to two other similarity models. First, we computed the similarity in the space of autotags [7] from the cosine distance over song's tags vector [2]. The second comparison was performed by retrieving the most acoustically similar songs. Acoustic similarity was determined by using G1C [15] which is a weighted combination of spectral similarity and information about spectral patterns. We also compared our model to a popularity-biased random model that probabilistically chooses the top songs based on their popularity. Each song's popularity was determined by looking at the number of sequences it is part of.

In the 3-song sequence case, for the autotag and acoustic similarity, we represent the similarity $sim(\{t_1, t_2\}, t_3)$ as the mean of $sim(t_1, t_3)$ and $sim(t_2, t_3)$.

The results of Table 3 clearly show that there is more involved than simple audio similarity when it comes to reconstructing sequences from radio station playlists. The performance of the audio and autotag similarity are indeed significantly lower than models that were trained on actual playlists.

Furthermore, the TB scores of Table 3 are from the models that have the best classification error (see Table 2). It is interesting to note that some models with a worst classification error have better TB scores. While classification is done by thresholding a model's certainty at 50%, TB gives an indication of the songs for which a model has the highest certainty. Since these are the songs that will be used when generating a playlist, this metric seems more appropriate to judge the models. The relation between classification error and TB scores is a topic for further investigation.

5. STEERABLE PLAYLIST GENERATION

While the model presented above is able to build a similarity space in which nearby songs fit well together in a playlist, it does not provide a mechanism for allowing the user to personalize the sequence for a given context. To address this, final song selection was done using the *Aura* [6] [2] recommendation engine from Sun Microsystems Labs. Aura is able to generate transparent and steerable recommendations by working with a textual representation — a tag cloud — of the items it is recommending. Specifically it finds the most similar items to any other in its pool by computing the cosine distance on their respective tag clouds. It can also explain to the user why an item is recommended by showing the overlap between tag clouds.

We use Aura as a means to allow users to personalize ("steer") the playlist generation by allowing them to create a personal tag cloud that represents the music they wish to listen to. In order to generate tag clouds for our tracks, we used *Autotagger* [7], a content-based machine learning model. This model is designed to generate a tag cloud (specifically a weighted vector of 360 music-relevant words) from an audio track, thus allowing us to use Aura's cosine distance measure to compute the similarity between

[6] `http://www.tastekeeper.com`

each track and the user's personal cloud.

5.1 Steps for generating a steerable playlist

Our playlist generation algorithm works as follows :

1. A seed track $t_s \in \mathbb{T}$ is selected amongst all possible tracks.

2. The similarity model is used to compute transitional probabilities between the seed song and all other ones (with more similar songs having higher transition probabilities), keeping only the top φ, or thresholding at a certain transition probability ρ. Let $\mathcal{T}$ be the group of these top songs:

$$\mathcal{T} = \underset{t_i \in \mathbb{T} \setminus t_s}{\arg\max^{\varphi}} \mathcal{M}(t_s, t_i) \quad (1)$$

3. The user is then invited to create a tag cloud $\mathcal{C}_U$ by assigning weights to any of the 360 tags in the system. In this way the cloud is personalized to represent the mood or type of songs the user would like to hear. The higher the weight of a particular tag, the more impact it will have on the selection of the next song.

4. Autotagger is used to generate a tag cloud $\mathcal{C}_{t_j}$ for all tracks $t_j \in \mathcal{T}$. The cosine distance ($cd(\cdot)$) between these tag clouds and $\mathcal{C}_U$ is used to find the song that best matches the abstract musical context the user described with his or her cloud:

$$t_{min} = \underset{t_j \in \mathcal{T}}{\arg\min}\, cd(\mathcal{C}_U, \mathcal{C}_{t_j}) \quad (2)$$

5. The track t_{min} is selected to play next. Since the system is transparent, we can tell the user we chose the song t_{min} because it has a certain transition probability from the seed song but also because its tag cloud overlapped with $\mathcal{C}_U$ in a particular way. The user can then go back and modify the tag cloud $\mathcal{C}_U$ to influence how subsequent songs will be selected.

Naturally, a lot of extra factors can be used when determining which song to play in step 4. For instance, we could consider the user's taste profile to take into account what types of songs he normally likes, mixing his current steerable cloud to the one representing his musical tastes. We could also include a discovery heuristic to balance the number of novel songs selected as opposed to ones the user already knows.

5.2 Example playlists

To illustrate the effect of the steerable tag cloud, we generate two playlists seeded with the same song but with very different steerable clouds. The first 9 iterations of both playlists are shown in Table 4. The effect of the cloud is clearly visible by the different direction each playlist takes. In our view, this transition is done smoothly because it is constrained by the underlying similarity model.

To visualize the similarity space and the playlist generating algorithm, we compute a full track-to-track similarity matrix and reduce its dimensionally using the t-SNEE [18] algorithm (see Figure 1). We chose t-SNEE because it tends to retain local distances while sacrificing global distances, yielding an appropriate two-dimensional visualization for this task (i.e. the distance between very similar songs is more important to us than the relative global placement of, e.g., jazz with respect to classical). We have overlaid the trajectory of the two playlists in Table 4 to illustrate their divergence.

Table 4. Both the following playlists are seeded with the song *Clumsy* by *Our Lady Peace*. To give a clear point of reference, we use the tag clouds of actual songs as the steerable cloud. The *soft* tag cloud is made up of the tags for *Imagine* by *John Lennon* and the *hard* tag cloud with the tags for *Hypnotize* by *System of a Down*.

Soft tag cloud
Viva la Vida by Coldplay
Wish You Were Here by Pink Floyd
Peaceful, Easy Feeling by Eagles
With or Without You by U2
One by U2
Fields Of Gold by Sting
Every Breath You Take by The Police
Gold Dust Woman by Fleetwood Mac
Enjoy The Silence by Depeche Mode
Hard tag cloud
All I Want by Staind
Re-Education (Through Labor) by Rise Against
Hammerhead by The Offspring
The Kill by 30 Seconds To Mars
When You Were Young by The Killers
Hypnotize by System of a Down
Breath by Breaking Benjamin
My Hero by Foo Fighters
Turn The Page by Metallica

6. CONCLUSIONS

We have demonstrated a method for learning song similarity based on radio station playlists. The learnt model induces a new space in which similar songs fit well when played successively in a playlist. Several classifiers were evaluated on a retrieval task, with SdAs and MLPs performing better than other similarity models in reconstructing song sequences from professional playlists. Though we were unable to show that SdAs outperform MLPs, we did show much better performance than logistic regression and measures such as G1C over standard audio features. Furthermore we argue that our model learns a direct similarity measure in the space of short song sequences rather than audio or meta-data based similarity. Finally, we showed a way of doing steerable playlist generation by using our similarity model in conjunction with a tag-based distance measure.

Though this model is only a first step, its power and simplicity lie in the fact that its two components play very different but complementary roles. First, the similarity model does the *grunt work* by getting rid of all unlikely candidates, as it was trained specifically for that task. This then greatly facilitates the steerable and fine-tuned selection of the subsequent track based on the textual aura, as most of

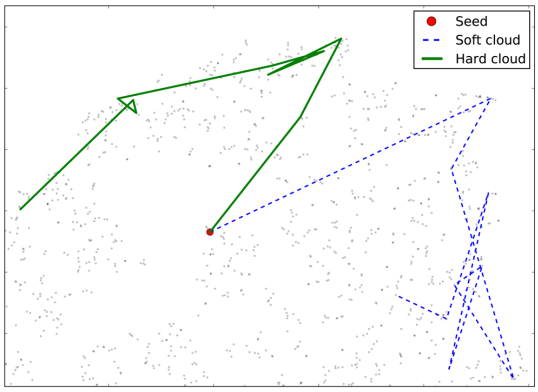

Figure 1. Part of the 2-d representation of the track-to-track similarity matrix generated by a 2-song sequence SdA model. The trajectories of the two playlists described in Table 4 are overlaid over the tracks. Both playlists are seeded with the same song, which is represented by the bigger dot. Each playlist diverges because of the steerable tag cloud that is guiding its generation.

the obvious bad picks have already been removed.

Future work should attempt to use the number of occurrences of each sequence to give more importance to more reliable sequences. Also, the model might learn a better similarity space by being trained with richer features as input. For example, adding meta-data such as tags, a measure of popularity, the year the song was recorded, etc., might prove helpful. Such a richer input space is likely necessary to show a performance gain for SdAs over competing probabilistic classifiers. Our experiments also led us to believe that an increase in the quality of the learnt similarity could probably be attained by simply adding more training data, something that can be easily accomplished as thousands of songs are played on the radio everyday.

7. REFERENCES

[1] R. Ragno, CJC Burges, C. Herley: "Inferring similarity between music objects with application to playlist generation," *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pp. 73–80, New York, USA, 2005

[2] S. Green, P. Lamere, J. Alexander, F. Maillet: 'Generating Transparent, Steerable Recommendations from Textual Descriptions of Items," *Proceedings of the 3rd ACM Conference on Recommender Systems*, New York, USA, 2009

[3] C. Baccigalupo, E. Plaza: "Case-based sequential ordering of songs for playlist recommendation," *Lecture Notes in Computer Science*, Vol. 4106, pp. 286–300, 2006.

[4] O. Goussevskaia, M. Kuhn, M. Lorenzi, R. Wattenhofer: "From Web to Map: Exploring the World of Music," *IEEE/WIC/ACM International Conference on Web Intelligence*, Sydney, Australia, 2008.

[5] B. Fields, C. Rhodes, M. Casey, K. Jacobson: "Social Playlists and Bottleneck Measurements: Exploiting Musician Social Graphs Using Content-Based Dissimilarity and Pairwise Maximum Flow Values," *Proceedings of the 9th International Conference on Music Information Retrieval*, Philadelphia, USA, 2008.

[6] P. Lamere: "Social Tagging And Music Information Retrieval," *Journal of New Music Research*, Vol. 37, No. 2, 2008.

[7] T. Bertin-Mahieux, D. Eck, F. Maillet, P. Lamere: "Autotagger: a model for predicting social tags from acoustic features on large music databases," *Journal of New Music Research*, Vol. 37, No. 2, pp. 115–135, 2008.

[8] E. Vickers: "Automatic long-term loudness and dynamics matching," *Proceedings of the AES 111th Convention*, New York, USA, 2001.

[9] S. Streich: "Music Complexity: a multi-faceted description of audio content," Ph.D. Dissertation, UPF, Barcelona, 2007.

[10] J.J. Aucouturier, F. Pachet: "Scaling up Music Playlist Generation," *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, Vol. 1, pp. 105–108, Lausanne, Switzerland, 2002.

[11] B. Logan: "Content-based playlist generation: Exploratory experiments," *Proceedings of the 3rd International Conference on Music Information Retrieval*, 2002

[12] B. Logan: "Music Recommendation from Song Sets," *Proceedings of the 5th International Conference on Music Information Retrieval*, Barcelona, Spain, 2004.

[13] E. Pampalk, T. Pohle, G. Widmer: "Dynamic Playlist Generation Based on Skipping Behaviour," *Proceedings of the 6th International Conference on Music Information Retrieval*, London, UK, 2005.

[14] A. Flexer, D. Schnitzer, M. Gasser, G. Widmer: "Playlist Generation Using Start and End Songs," *Proceedings of the 9th International Conference on Music Information Retrieval*, Philadelphia, USA, 2008.

[15] E. Pampalk: "Computational Models of Music Similarity and their Application in Music Information Retrieval, Ph.D. dissertation, Vienna University of Technology, 2006.

[16] M. Mandel, D. Ellis: "Song-Level Features and Support Vector Machines for Music Classication," *Proceedings of the 6th International Conference on Music Information Retrieval*, London, UK, 2005.

[17] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol: "Extracting and Composing Robust Features with Denoising Autoencoders," *International Conference on Machine Learning*, 2008

[18] L.J.P. van der Maaten, G.E. Hinton: "Visualizing High-Dimensional Data Using t-SNE," *Journal of Machine Learning Research*, 9(Nov):2579-2605, 2008.