THE NATURAL LANGUAGE OF PLAYLISTS

Brian McFee
Computer Science and Engineering
University of California, San Diego

Gert Lanckriet
Electrical and Computer Engineering
University of California, San Diego

ABSTRACT

We propose a simple, scalable, and objective evaluation procedure for playlist generation algorithms. Drawing on standard techniques for statistical natural language processing, we characterize playlist algorithms as generative models of strings of songs belonging to some unknown language. To demonstrate the procedure, we compare several playlist algorithms derived from content, semantics, and meta-data. We then develop an efficient algorithm to learn an optimal combination of simple playlist algorithms. Experiments on a large collection of naturally occurring playlists demonstrate the efficacy of the evaluation procedure and learning algorithm.

1. INTRODUCTION

Music listeners typically do not listen to a single song in isolation. Rather, listening sessions tend to persist over a sequence of songs: a *playlist*. The increasing quantity of readily available, digital music content has motivated the development of algorithms and services to automate search, recommendation, and discovery in large music databases. However, playlist generation is fundamental to how users interact with music delivery services, and is generally distinct from related topics, such as similarity and semantic search.

Although many automatic playlist generation algorithms have been proposed over the years, there is currently no standard evaluation procedure. As a result, it is difficult to quantitatively compare different algorithms and objectively determine if any progress is being made.

At present, the predominant approach to playlist algorithm evaluation is to conduct human opinion surveys, which can be expensive, time-consuming and difficult to reproduce. Alternatively, current automated evaluation schemes either reduce the problem to a (discriminative) information retrieval setting, or rely on simplifying assumptions that may not hold in practice.

In this work, we propose a simple, scalable, and objective evaluation procedure for playlist algorithms that avoids the pitfalls of previous approaches. Our approach is guided by the observation that playlist generation is not (only) an information retrieval problem, but a *language modeling* problem. The proposed method can be applied to a large class of playlist algorithms, and we provide several examples with experimental results. Finally, we propose an algorithm to learn an optimal ensemble algorithm from a collection of simple playlist generators.

2. A BRIEF HISTORY OF PLAYLIST EVALUATION

Although many algorithms for playlist generation have been proposed, evaluation procedures have received relatively little specific attention. Here, we briefly summarize previously proposed evaluation strategies, which can broadly be grouped into three categories: *human evaluation*, *semantic cohesion*, and *sequence prediction*. This section is not intended as a comprehensive survey of playlist *algorithms*, for which we refer the interested reader to [8, chapter 2].

2.1 Human evaluation

Since the eventual goal of playlist algorithms is to improve user experience, the ideal method of algorithm evaluation is to directly measure human response. Numerous studies have been conducted in which test subjects rate the quality of playlists generated by one or more algorithms. Pauws and Eggen [18] asked users to provide a query song with a particular context-of-use in mind (*e.g.*, *lively music*), which was used as a seed to generate a playlist. The user evaluated the resulting playlist on a scale of 1–10, and how many tracks in the playlist fit the user's intended use context. From these survey responses, the authors were able to derive various statistics to demonstrate that their proposed algorithm significantly outperforms randomly generated playlists. Similarly, Barrington, et al. [1] conducted experiments in which users were presented with two playlists (generated by obscured, competing systems) and asked to indicate which one was (subjectively) better, and why.

While direct human evaluation studies can provide evidence that one algorithm measurably outperforms another, they also have obvious practical limitations. This can be laborious, difficult to reproduce, and may require large numbers

of test subjects and example playlists to achieve statistically meaningful results and overcome the effects of subjectivity.

2.2 Semantic cohesion

The general impracticality of large-scale user studies has motivated the development of automated evaluation techniques. The most common approaches compute some easily measurable quantity from each song in a generated playlist (*e.g.*, artist, album, or genre), which is used to determine the *cohesion* of the playlist. Cohesion may be defined by frequency counts of meta-data co-occurrence (*e.g.*, songs by the same artist) [13, 14] or entropy of the distribution of genres within the playlist [7, 12]. In this framework, it is typically assumed that each song can be mapped to a unique semantic tag (*e.g.*, *blues*). This assumption is often unrealistic, as songs generally map to multiple tags. Assigning each song to exactly one semantic description may therefore discard a great deal of information, and obscure the semantic content of the playlist. A more general form of semantic summarization was developed by Fields, et al. [9], and used to derive a distance measure between latent topic models of playlists. However, it is not immediately clear how such a distance metric would facilitate algorithm evaluation.

Issues of semantic ambiguity aside, a more fundamental flaw lies in the assumption that cohesion accurately characterizes playlist quality. In reality, this assumption is rarely justified, and evidence suggests that users often prefer highly diverse playlists [20].

2.3 Sequence prediction

A more direct approach to automatic evaluation arises from formulating playlist generation as a prediction problem: given some contextual query (*e.g.*, a user's preferences, or a partial observation of songs in a playlist), the algorithm must predict which song to play next. The algorithm is then evaluated on the grounds of its prediction, under some notion of correctness. For example, Platt, et al. [19] observe a subset of songs in an existing playlist (the *query*), and the algorithm predicts a ranking of all songs. The quality of the algorithm is then determined by the position within the predicted ranking of the remaining, unobserved songs from the playlist. Maillet, et al. [15] similarly predict a ranking over songs from a contextual query — in this case, the preceding song or pair of songs — and evaluate by comparing the ranking to one derived from a large collection of existing playlists.

Essentially, both of the above approaches transform playlist evaluation into an information retrieval (IR) problem: songs observed to co-occur with the query are *relevant*, and all other songs as *irrelevant*. As noted by Platt, et al. [19], this notion of relevance may be exceedingly pessimistic in practice due to sparsity of observations. In even moderately large music databases (say, on the order of thousands of songs), the probability of observing any given pair of songs in a playlist becomes vanishingly small, and therefore, the overwhelming majority of song predictions are considered incorrect. In this framework, a prediction may disagree with observed co-occurrences, but still be equally pleasing to a user of the system, and therefore be unfairly penalized.

The IR approach — and more generally, any discriminative learning approach — is only applicable when one can obtain negative examples, *i.e.*, *bad* playlists. In reality, negative examples are difficult to define, let alone obtain, as users typically only share playlists that they like. [1] This suggests that discriminative evaluation may not be the most natural fit for playlist generation.

3. A NATURAL LANGUAGE APPROACH

In contrast to discriminative approaches to playlist evaluation, we advocate the *generative* perspective when modeling playlist composition. Rather than attempting to objectively score playlists as *good* or *bad*, which generally depends on user taste and unobservable contextual factors, we instead focus on modeling the distribution of naturally occurring playlists.

Formally, let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ denote a library of songs. We define a *playlist* as an ordered finite sequence of elements of $\mathcal{X}$. Any procedure which constructs such ordered sequences is a *playlist algorithm* (or *playlister*). In general, we consider randomized algorithms, which can be used to generate multiple unique playlists from a single query. Each playlister, be it randomized or deterministic, induces a probability distribution over song sequences, and may therefore be treated as a probabilistic generative model.

This leads to our central question: how should generative models of song sequences be evaluated? Here, we take inspiration from the literature of statistical natural language processing [16], in which statistical models are fit to a sample of strings in the language (*e.g.*, grammatically valid sentences in English). A language model determines a probability distribution $\mathbf{P}$ over strings, which can be evaluated objectively by how well $\mathbf{P}$ matches the true distribution $\mathbf{P}_*$. Since $\mathbf{P}_*$ is unknown, this evaluation is approximated by drawing a sample $\mathcal{S} \sim \mathbf{P}_*$ of naturally occurring strings, and then computing the *likelihood* of the sample under the model $\mathbf{P}$.

Returning to the context of playlist generation, in place of vocabulary words, we have songs; rather than sentences, we have playlists. The universe of human-generated playlists therefore constitutes a *natural language*, and playlisters are models of the language of playlists. While this observation is not itself novel — it appears to be folklore among music researchers — its implications for algorithm evaluation have not yet been fully realized. We note that recent work by Zheleva, et al. [21] evaluated playlisters in terms of perplexity

[1] A notable exception is the work of Bosteels, et al. [4], in which explicit negative feedback was inferred from skip behavior of Last.fm users. As noted by the authors, skip behavior can be notoriously difficult to interpret.

(exponentiated log-likelihood) of the genre distribution in a playlist, rather than the song selection itself.

3.1 Evaluation procedure

To evaluate a playlister A, we require the following:

1. a library of n songs $\mathcal{X}$,
2. a sample of playlists $\mathcal{S} \subseteq \mathcal{X}^*$,[2] and
3. the likelihood $\mathbf{P}_A[s]$ of any playlist $s \in \mathcal{X}^*$.

While the last requirement may seem like a tall order, we will demonstrate that for large classes of playlisters, the computation can be quite simple.

A playlister A can be evaluated by computing the average log-likelihood of the sample $\mathcal{S}$:

$$\mathcal{L}(\mathcal{S} \mid A) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \log \mathbf{P}_A[s]. \quad (1)$$

The average log-likelihood, on an absolute scale, is not directly interpretable — although it approximates the cross-entropy between $\mathbf{P}_A$ and the true, unknown distribution $\mathbf{P}_*$ [16] — but it is useful for performing relative comparisons between two playlisters. Given a competing playlister A', we can say that A is a better model of the data than A' if $\mathcal{L}(\mathcal{S} \mid A) > \mathcal{L}(\mathcal{S} \mid A')$.

There is a subtle, but important distinction between the proposed approach and previous approaches to playlist evaluation. Rather than evaluate the perceived quality of a generated, *synthetic* playlist, we instead evaluate the *algorithm* in terms of how likely it is to produce *naturally occurring* playlists.

4. PLAYLIST ALGORITHMS

To demonstrate the proposed evaluation approach, we will derive playlist probabilities for several generic playlisters. Although the method is fully general, we restrict attention to playlisters which satisfy the Markov property:

$$\mathbf{P}\left[(x_0, x_1, \ldots, x_k)\right] = \mathbf{P}[X = x_0] \prod_{i=1}^{k} \mathbf{P}[X_{t+1} = x_i \mid X_t = x_{i-1}]. \quad (2)$$

We assume that the first song is chosen uniformly at random, and therefore contributes a fixed constant $\log 1/n$ to the overall log-likelihood, which may be safely ignored. The likelihood of an arbitrary playlist under a Markov model can therefore be decomposed into the product of bigram likelihoods, so the log-likelihood is proportional to the sum:

$$\log \mathbf{P}\left[(x_0, \ldots, x_k)\right] \propto \sum_{i=1}^{k} \log \mathbf{P}[X_{t+1} = x_i \mid X_t = x_{i-1}].$$

[2] $\mathcal{X}^*$ denotes the Kleene-$*$ operation, and contains all sequences of any length of elements drawn from $\mathcal{X}$.

Note that this reasoning can be extended to higher order Markov models — *e.g.*, second order would decompose into trigrams — but to ease exposition, we focus on first-order models. For the remainder of this article, we will assume that $\mathcal{S}$ is a collection of bigrams.

4.1 Uniform shuffle

The simplest playlister selects each song uniformly at random from $\mathcal{X}$. This can be refined somewhat by disallowing consecutive repetitions, so that if the current song is x_t, then x_{t+1} is drawn uniformly at random from $\mathcal{X} \setminus \{x_t\}$. Since x_{t+1} depends only on x_t, it satisfies the Markov property, and the conditional bigram probability is

$$\mathbf{P}_U[X_{t+1} = x \mid X_t = x_t] = \begin{cases} 1/n-1 & x \neq x_t \\ 0 & x = x_t \end{cases}. \quad (3)$$

The uniform shuffle playlister provides an obvious baseline, and should be included in any comparative evaluation.

4.2 Weighted shuffle

A slight variation on the uniform shuffle is to draw the next song not from a uniform distribution, but a weighted distribution derived from a score function $F(x) > 0$, which may encode artist popularity, user preference, or any other song-level property. The resulting bigram probability is

$$\mathbf{P}_F[X_{t+1} = x \mid X_t = x_t] = \begin{cases} \frac{F(x)}{\sum_{x' \neq x_t} F(x')} & x \neq x_t \\ 0 & x = x_t \end{cases}. \quad (4)$$

In general, F may be dynamic and can be used to incorporate user feedback, thereby facilitating *steerability* [15]. Dynamic and interactive evaluation is beyond the scope of this article, and we focus on static score functions.

4.3 K-Nearest neighbor and random walks

Another simple strategy for playlist generation is to construct a k-nearest-neighbor (kNN) graph over the song set by using some previously constructed distance metric (*e.g.*, acoustic, semantic, or social similarity), and form playlists by a random walk process on the graph. If the next song x_{t+1} is chosen uniformly at random from the neighbors $\eta(x_t)$ of the current song x_t, then the bigram probability is

$$\mathbf{P}_{k\text{NN}}[X_{t+1} = x \mid X_t = x_t] = \begin{cases} 1/k & x \in \eta(x_t) \\ 0 & x \notin \eta(x_t) \end{cases}. \quad (5)$$

One shortcoming of this approach — as well as any deterministic playlister — is that it assigns 0 probability to some transitions, in this case, those spanning non-adjacent nodes. Any such transition would be infinitely unlikely under the model; however, it seems unreasonable to expect that every observed bigram coincides with an edge in the graph (unless

the graph is complete). This can be remedied by smoothing with the uniform distribution (weighted by a constant μ):

$$\widehat{\mathbf{P}}_{k\text{NN}} = (1-\mu)\mathbf{P}_{k\text{NN}} + \mu\mathbf{P}_U \qquad \mu \in (0,1]\,. \quad (6)$$

Since probability distributions are closed under convex combinations, Eqn. (6) describes a valid distribution. Equivalently, this models a process which flips a μ-biased coin to decide whether to jump to an adjacent song in the graph, or a random song in the library (adjacent or not). This modification to the algorithm increases diversity and flexibility, and ensures that log-likelihood computations remain finite.

4.4 Markov chain mixtures

Any non-trivial playlister requires some tuning of parameters. For example, to implement kNN, one must select the underlying features and similarity metric, the neighborhood size k, and the smoothing parameter μ. This leads to an obvious question: can these parameters be optimized automatically? More generally, if we start with a collection of playlisters A_i (say, derived from different features [10, 12], values of k, *etc.*), is it possible to intelligently integrate them to into a single playlister?

Eqn. (6) exploits the fact that distributions are closed under convex combinations to combine two distributions (uniform and kNN) with fixed proportion μ. This can be generalized to combine m distributions as follows:

$$\mathbf{P}_\mu = \sum_{i=1}^{m} \mu_i \mathbf{P}_i \qquad \forall i:\ \mu_i \geq 0, \quad \sum_{i=1}^{m} \mu_i = 1. \quad (7)$$

Rather than using a fixed weighting $\mu = (\mu_1, \mu_2, \ldots, \mu_m)$, we can instead optimize μ by maximizing the likelihood of a collection of training examples under the mixture model. This can be accomplished by solving the optimization problem listed as Algorithm 1. Because the objective function (log-likelihood) is concave in μ, and the constraints are linear, this problem can be solved efficiently [5].

After computing the maximum likelihood estimate μ, playlists can be generated by sampling from the weighted ensemble distribution $\mathbf{P}_\mu$. The distribution described by Eqn. (7) characterizes the ensemble playlist algorithm listed as Algorithm 2, which, given the current song x_t, simply selects a playlister A_i at random according to the discrete distribution characterized by μ and returns a sample from the selected distribution $\mathbf{P}_i[X \mid X_t = x_t]$.

5. EXPERIMENTS

To demonstrate the proposed evaluation approach, we implemented several playlisters on a large song library, using acoustic-, semantic-, and popularity-based descriptors. The simple playlisters described here are merely intended to demonstrate plausible baselines against which more sophisticated algorithms may be compared in future work.

Algorithm 1 Markov chain mixture optimization

Input: Training bigrams
$\mathcal{S}' = \{(x_1, x'_1), \ldots, (x_{|\mathcal{S}'|}, x'_{|\mathcal{S}'|})\}$
Markov chains $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_m$
Output: Combination weights $\mu_1, \mu_2, \ldots, \mu_m$

$$\max_{\mu} \ \frac{1}{|\mathcal{S}'|} \sum_{(x,x')\in\mathcal{S}'} \log\left(\sum_{i=1}^{m} \mu_i \mathbf{P}_i\left[X_{t+1} = x' \mid X_t = x\right]\right)$$

$$\text{s.t. } \forall i\ :\ \mu_i \geq 0, \quad \sum_{i=1}^{m} \mu_i = 1$$

Algorithm 2 Ensemble playlist generation

Input: Current song x_t, playlisters $(A_i, \mathbf{P}_i)$, weights μ_i
Output: Next song x_{t+1}

1: Sample $i \sim \text{DISCRETE}(\mu)$ {Choose A_i}
2: **return** $x_{t+1} \sim \mathbf{P}_i[X_{t+1} \mid X_t = x_t]$ {Run $A_i(x_t)$}

5.1 Song data: Million Song Dataset

Our song data was taken from the Million Song Dataset (MSD) [3], upon which we constructed models based on artist terms (tags), familiarity, and audio content.

Tag representations were derived from the vocabulary of 7643 artist terms provided with MSD. Each song is represented as a binary vector indicating whether each term was applied to the corresponding artist, and nearest neighbors are determined by cosine-similarity between tag vectors.

The Echo Nest [3] artist familiarity is used to define a static score function F over songs, which may be interpreted as a surrogate for (average) user preference.

The audio content model was developed on the 1% Million Song Subset (MSS), and is similar to the model proposed in [17]. From each MSS song, we extracted the time series of Echo Nest timbre descriptors (ENTs). This results in a sample of approximately 8.5 million 12-dimensional ENTs, which were normalized by z-scoring according to the estimated mean and variance of the sample, randomly permuted, and then clustered by online k-means to yield 512 acoustic codewords. Each song was summarized by quantizing each of its (normalized) ENTs and counting the frequency of each codeword, resulting in a 512-dimensional histogram vector. Each codeword histogram was mapped into a probability product kernel (PPK) space [11] by square-rooting its entries, which has been demonstrated to be effective on similar audio representations [17]. Finally, we appended the song's tempo, loudness, and key confidence, resulting in a vector $v_i \in \mathbb{R}^{515}$ for each song x_i.

Next, we trained an optimized similarity metric over audio descriptors. We computed target similarity for each pair of MSS artists by the Jaccard index between their user sets in

[3] `http://developer.echonest.com`

a sample of Last.fm [4] collaborative filter data [6, chapter 3]. Tracks by artists with fewer than 30 listeners were discarded. The remaining artists were partitioned 80/20 into a training and a validation set, and for each artist, we computed its top 10 most similar training artists. The distance metric was subsequently optimized by applying the metric learning to rank (MLR) algorithm on the training set of 4455 songs, and tuning parameters $C \in \{10^5, 10^6, \ldots, 10^9\}$ and $\Delta \in \{\text{AUC, MRR, MAP, Prec@10}\}$ to maximize AUC score on the validation set of 1110 songs. Finally, the resulting metric W was factored by PCA (retaining 95% of spectral mass) to yield a linear projection $L \in \mathbb{R}^{222\times 515}$ which maps each v_i into a Euclidean space in which nearest neighbor is optimized to retrieve songs by similar artists.

5.2 Playlists: Art of the Mix

Playlist data was taken from the Art of the Mix [5] (AotM) corpus collected by Berenzweig, et al. [2]. We chose this corpus primarily for two reasons. First, it is the largest publicly available set that we know of. Second, each playlist was (ostensibly) generated by a user — not a recommendation service or commercial radio DJ — so the corpus is an accurate sample of real playlists that occur in the wild. [6]

The AotM data consists of approximately 29K playlists over 218K unique songs by 48K unique artists, which we cleaned with a two-step procedure. First, artist names were resolved to identifiers by the Echo Nest artist search API. Second, we matched each song's artist identifier to the MSD index, and if the artist was found, we matched the title against all MSD song titles by the artist. A match was accepted if either title was contained in the other, or the edit distance was less than half the (AotM) title length. This was found by informal inspection to yield fewer false matches than a direct (*artist*, *title*) query to the Echo Nest API.

Having resolved songs to MSD identifiers, we then filtered the playlist set down to bigrams in which both consecutive songs were contained in MSD. This results in a collection $\mathcal{S}$ of 66250 bigrams over a library $\mathcal{X}$ of 26752 unique songs by 5629 unique artists. [7]

5.3 Experimental procedure

For each song $x_i \in \mathcal{X}$, we computed an optimized acoustic descriptor $v_i \in \mathbb{R}^{222}$, tag vector $w_i \in \{0,1\}^{7643}$, and artist familiarity score $F(x_i) \in [0,1]$. The familiarity score was used to construct a weighted shuffle Markov chain (Eqn. (4)). The audio and tag spaces were used to generate kNN Markov chains (Eqn. (5)) for $k \in \{2^4, 2^5, \ldots, 2^{12}\}$. This results in a collection of 9 audio-based Markov chains, 9 tag-based, and one familiarity-based. Including the uniform shuffle model, we have a total $m = 20$ simple playlisters.

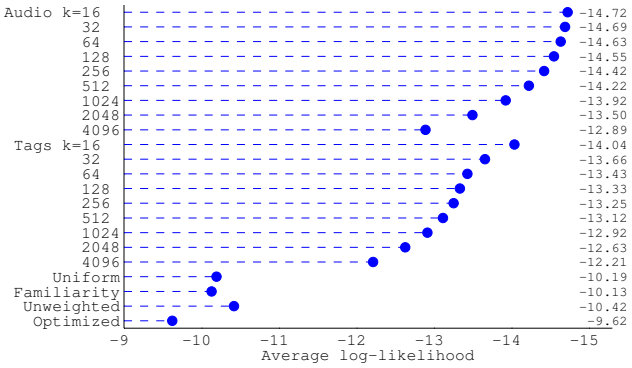

Figure 1. Average log-likelihood of test bigrams for each model under comparison. Scores are averaged across ten random training/test splits.

The playlist set $\mathcal{S}$ was randomly partitioned 10 times into 10%-train, 90%-test sets; each split was performed over the first element of the bigram so that for each song x_i, all bigrams $(x_i, \cdot)$ belong to either the training or test set. On average, this yields 6670.9 training and 59597.1 test bigrams.

Each simple playlister was evaluated by computing the average log-likelihood of test bigrams (x, x') (Eqn. (1)). All playlisters were smoothed by Eqn. (6) with $\mu = 0.01$.

We then ran Algorithm 1 on the training set, and evaluated the resulting playlister on the test set. Our implementation of Algorithm 1 is written in NumPy, [8] and on average, converges to the global optimum in under 20 seconds on standard hardware. Since the ensemble includes the uniform model, no additional smoothing is necessary. Finally, for comparison purposes, we also compared to the unweighted combination by fixing each $\mu_i = 1/m$.

5.4 Results

Figure 1 lists the average log-likelihood of each model under comparison. Although the audio- and tag-based models tend to generate playlists which are acoustically or semantically consistent, [7] they do not accurately model naturally occurring playlists. As illustrated in Figure 2, the majority of bigrams disagree with adjacencies in the kNN graphs, so kNN methods are outperformed by uniform shuffle. While the features described here do not suffice to model naturally occurring playlists, a richer feature set including lyrical or social information may significantly improve performance, and will be the subject of future research.

For small values of k, the tag playlister is forced to select

[4] `http://last.fm`

[5] `http://www.artofthemix.org/`

[6] One could of course model playlists derived from alternative sources, but be aware that such playlists may have different characteristics than user-generated playlists: *e.g.*, terrestrial radio playlists may be constrained by broadcast regulations or commercial factors.

[7] The bigram data and example playlists for each algorithm can be downloaded from `http://www-cse.ucsd.edu/~bmcfee/playlists/`.

[8] `http://numpy.scipy.org`

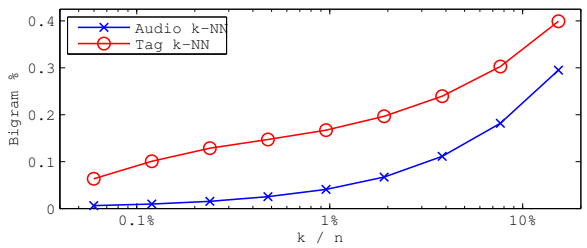

Figure 2. Fraction of bigrams $(x, x') \in \mathcal{S}$ where x' is a k-nearest neighbor of x (as a function of k/n).

	Audio	Tags	Familiarity	Uniform
μ	9%	27%	36%	28%

Table 1. Average weight assigned to each model when optimized by Algorithm 1. *Audio* and *Tag* weights are aggregated across all values of $k \in \{2^4, 2^5, \ldots, 2^{12}\}$.

among songs with highly similar tag vectors. Tag-based playlisters, therefore, tend to maximize semantic cohesion. The relatively low performance of the tag playlister indicates that semantic cohesion does not adequately describe naturally occurring playlists.

The *familiarity* model performs slightly better than uniform, and significantly better than the audio and tag playlisters. This suggests that popularity and social factors play significant roles in playlist composition; while not surprising, this should be taken into account when designing a playlister.

The optimized model produced by Algorithm 2 substantially outperforms all other models, even when only exposed to an extremely small training set (10%). Note that the unweighted combination degrades performance.

To help understand contributions of different components in the optimized model, we list the average weight assigned to each model by Algorithm 1 in Table 1, grouped by feature type. The content-based models receive a significant amount of weight, suggesting that the models contain some amount of predictive power. The large weight assigned to the uniform model may be interpreted as the proportion of information not modeled by content or familiarity, and thus constitutes a secondary measure of the (lack of) quality of the other models in the ensemble.

6. CONCLUSION

We have presented a simple, automatic evaluation procedure for playlist algorithms. To demonstrate the technique, we developed a suite of simple baseline playlisters, and evaluated their performance on naturally occurring playlists.

7. ACKNOWLEDGMENTS

The authors thank Benjamin Fields and Matthew Hoffman for many helpful conversations, and acknowledge support from Qualcomm, Inc., Yahoo! Inc., the Hellman Fellowship Program, and NSF Grants CCF-0830535 and IIS-1054960.

8. REFERENCES

[1] Luke Barrington, Reid Oda, and G.R.G. Lanckriet. Smarter than genius? Human evaluation of music recommender systems. In *ISMIR*, 2009.

[2] A. Berenzweig, B. Logan, D.P.W. Ellis, and B. Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *CMJ*, 28(2):63–76, 2004.

[3] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *ISMIR*, 2011.

[4] K. Bosteels, E. Pampalk, and E.E. Kerre. Evaluating and analysing dynamic playlist generation heuristics using radio logs and fuzzy set theory. In *International Conference on Music Information Retrieval*, 2009.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[6] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[7] M. Dopler, M. Schedl, T. Pohle, and P. Knees. Accessing music collections via representative cluster prototypes in a hierarchical organization scheme. In *ISMIR*, 2008.

[8] B. Fields. *Contextualize Your Listening: The Playlist as Recommendation Engine*. PhD thesis, Goldsmiths, University of London, April 2011.

[9] B. Fields, C. Rhodes, and M. d'Inverno. Using song social tags and topic models to describe and compare playlists. *Workshop on Music Recommendation and Discovery*, 2010.

[10] Ben Fields, Christophe Rhodes, Michael Casey, and Kurt Jacobsen. Social playlists and bottleneck measurements: Exploiting musician social graphs using content-based dissimilarity and pairwise maximum flow values. In *ISMIR*, 2008.

[11] Tony Jebara, Risi Kondor, and Andrew Howard. Probability product kernels. *JMLR*, 5:819–844, Dec 2004.

[12] P. Knees, T. Pohle, M. Schedl, and G. Widmer. Combining audio-based similarity with web-based data to accelerate automatic music playlist generation. In *ACM international workshop on multimedia information retrieval*, 2006.

[13] B. Logan. Content-based playlist generation: exploratory experiments. In *ISMIR*, 2002.

[14] B. Logan. Music recommendation from song sets. In *ISMIR*, 2004.

[15] F. Maillet, D. Eck, G. Desjardins, and P. Lamere. Steerable playlist generation by learning song similarity from radio station playlists. In *ISMIR*, 2009.

[16] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[17] B. McFee, L. Barrington, and G.R.G. Lanckriet. Learning content similarity for music recommendation, 2011. http://arxiv.org/1105.2344.

[18] S. Pauws and B. Eggen. PATS: Realization and user evaluation of an automatic playlist generator. In *ISMIR*, 2002.

[19] J.C. Platt, C.J.C. Burges, S. Swenson, C. Weare, and A. Zheng. Learning a gaussian process prior for automatically generating music playlists. In *NIPS*. MIT Press, 2002.

[20] M. Slaney and W. White. Measuring playlist diversity for recommendation systems. In *1st ACM workshop on Audio and music computing multimedia*, AMCMM '06, pages 77–82, New York, NY, USA, 2006. ACM.

[21] E. Zheleva, J. Guiver, E. Mendes Rodrigues, and N. Milić-Frayling. Statistical models of music-listening sessions in social media. In *WWW*, 2010.