Spectral Transition-Based Playlist Prediction

Nipun Agarwala, Chris Billovits, Rahul Prabala {nipuna1, cjbillov, rprabala }@stanford.edu

December 11, 2015

Abstract

Since the advent of the radio, and in the last decade, Pandora and Apple Music, curated playlists have become a crucial player in the music industry. Current techniques for playlist recommendation rely on supervised learning or collaborative filtering on metadata. Though such techniques may work well sometimes, they do not capture the musical semantics which attracts listeners to one song or makes them desire a certain song after the current one plays in a playlist. The paper proposes a transition model on spectral data to learn musical tastes and patterns of playlists. We achieve 0.96 F1 using a small shallow neural network on supervised transition prediction task, formulated below.

1 Background

Playlist generation is motivated by several different lines of thought. Traditional radio stations seek to play popular songs. Other DJ's may select playlists in order to match prior interests with new, unheard artists [1]. Much of the real-world implementation of playlist recommendation systems are reliant on relevance feedback to mediate between this priorities. We seek to classify playlists which were built on transition-based criteria [1] which is a generally unexplored approach.

The majority of playlist recommendation schemes rely entirely on metadata, rather than audio. While some approaches have included temporal structure of songs, tempo, and other user-based "constraints" that may be imposed on a generated playlist [2], there have been relatively few investigations into factoring the spectral component into playlist generation. Those approaches that rely on spectral content more often than not use it solely as a similarity metric [3]. Chen et al. [4] have examined metric embeddings as a smoothing model

to reduce playlist perplexity, which becomes an issue because of data sparsity.

1.1 Motivation

The end goal of automated playlist generation can be derived directly from a binary transition-based classifier. Given n songs, the objective would be to craft a k-length playlist that maximizes the joint probability under Markov assumptions. This is a linear chain graphical model, upon which inference is tractable. The local probability distributions $p(s_i|s_{i-1})$ can then be derived from the label confidence of a binary transition classifier on the window (s_{i-1}, s_i) .

We present this binary transition-based classifier and analyze its direct performance, and briefly assess its suitability for the end-goal task.

2 Data Collection

In order to draw deep, high order connections on the relative ordering of songs within a playlist, we begin by selecting curated playlists employed by DJ's at EDM music festivals in order to ensure that playlists which emphasize good transitions between songs. We follow the general formulation of [5] in developing a semi-supervised binary classification algorithm, treating consecutive songs as positive examples. To generate benign negative examples, we crawl Soundcloud and took 358 popular EDM-genre music streams and select random transitions between these songs to be negative, with the assumption that random pairs of songs will not possess strong transitions. This allows simple binary classification without introducing a very large bias. We generated 845 examples, of which 31% were positive support, and stratified them into a 60 / 40 train / test set split.

3 Feature Selection

3.1 Fast Fourier Transforms (FFT)

The unsupervised methodology proposed to derive musical semantics of the inputs involves the Fast Fourier Transform(FFT), computed using the Discrete Time Fourier Transform by

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\omega n}$$

Each music input of total time t seconds is divided into $\frac{t}{\tau}$ strips, where τ is the configurable length of each musical strip. An FFT is performed for each strip, generating frequencies based on τ and the sample rate of the audio file. These frequencies are then binned according to the frequencies of notes for a standard piano tuning $A4=440 \mathrm{Hz}$. The separation in frequencies between the notes is divided equally and assigned to each note, respectively, so that the value of the FFT for a given frequency will be assigned to the bin with the least difference. The values (in Hz) for the bins are given by

$$f(n) = 440 \times 2^{\frac{n-49}{12}}$$

3.2 Mel Frequency Cephstral Coefficients (MFCC)

An additional featurization was selected using the Mel Frequency Cephstral Coefficients (MFCCs) and their Delta coefficients. The MFCCs more closely approximate the way that humans perceive audio by emphasizing the lower frequencies in the audible range [6]. Rather than simply capture the frequency component at a given time, these coefficients capture a scaled version of the energy present in each filterbank, which is a binned collection of frequencies. The filterbanks are calculated by the Mel scale, which is given by

$$M(f) = 1125 \ln(1 + f/700)$$

The MFCCs are computed in a similar fashion to the FFTs, in that the sample is first split into frames, upon which the mel-filterbanks are applied by using a triangular filter centered at each filterbank value. The filtered energies are then logscaled, and the coefficients are extracted from the log of the Discrete Cosine Transform of the logscaled filtered energies.

Additionally, the Delta coefficients can be computed on a given set of MFCCs that characterize

the change in MFCCs over the time scale given by each frame. This enhanced set of features allows for greater performance in many speech and music recognition tasks [6]. For a given frame t, the Delta coefficient d_t is given by

$$d_{t} = \frac{\sum_{i=1}^{N} i(c_{t+i} - c_{t-i})}{2\sum_{i=1}^{N} i^{2}}$$

where c_t is the MFCC at frame t. The data and feature selection comprises a live pipeline that will be used to continually refine negative example generation.

4 Models

We implement 3 primary binary classifiers from scratch using numpy and scipy, benchmarking them against scikit-learn when applicable.

4.1 Logistic Regression

A naive first algorithm used was Logistic Regression, where 0 was assigned for bad transitions of songs and 1 for good transitions. It runs the following optimization objective:

$$\min J(\theta) = \|y - h_{\theta}(x)\|^2 + \lambda \|\theta\|^2$$

where $h_{\theta}(x)$ is the sigmoid function. Since the model is trained on more adverserial than positive examples, the model is hopefully selective in its choosing of good transitions, as seen in the results. Since it can be noticed that the FFT amplitudes are generally very small (of the order 10^{-5}), L-2 regularization was added to keep the coefficients small. This addition also prevented overfitting of the model on the training set, considering that there were much more features (either 1000 or even 20,000) than training examples.

4.2 Hinge Loss Regression

A more complex model used the Hinge loss to allow for stronger classification than Logistic Regression through a max margin classifier. The optimization problem is:

$$\min J(\theta) = \max(0, 1 - y \cdot \theta^T x) + \lambda \|\theta\|^2$$

where $\theta^T x$ is our predictions. This loss function linearly increases with the absolute continuous value of $\theta^T x$. Another interesting note about this loss is the max operator. If y and $\theta^T x$ are of the same sign, then the max operator chooses 0 as

the loss, showing that the prediction is correct and no loss needs to be added to the objective. Like above, L-2 normalization is added to keep the θ values in check and prevent over-fitting due to a large feature size.

4.3 Neural Networks

We used shallow feedforward neural networks with small hidden layer sizes and tanh activation. The intuition was that shallow neural networks are sufficient to model arbitrarily complex functions. In effect, the neural network trains a $\tan(h)$ + affine kernel from the full input vectors to H features, under the top layer objective. We used a least squares shallow neural network with the objective function

$$J(W, U) = \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - U \cdot (f(W \cdot x^{(i)})) \right)^{2} + \frac{\lambda}{2} (||W||^{2} + ||U||^{2})$$

The activation function is f = tanh. In our notation here, we use the bias trick, so U has dimension $2 \times H + 1$ and W has dimension $H + 1 \times numfeat + 1$. Note that $y^{(i)}$ is a one-hot label vector.

Secondly, we implemented a sigmoid top layer, with which to get confidences on our classification decision:

$$J(W, U) = \sum_{i=1}^{m} y^{(i)T} \cdot \log \left(\sigma \left(U \cdot f(W \cdot x^{(i)}) \right) \right)$$
$$+ \frac{\lambda}{2} (||W||^2 + ||U||^2)$$

We used the latter network with sigmoid top layer to achieve our best results.

4.4 Optimization

We utilized two optimization techniques not mentioned in class, so we summarize them here.

Adagrad. For all three classifiers, we optimize the objective function with stochastic gradient descent and annealed learning rates. For logistic regression and neural networks, we use the diagonal Adagrad method [7]. Our Adagrad implementation provides a per-feature learning rate that monotonically decreases according to the l_2 norm of the gradient history. Adagrad works

well because it has tight regret bounds. Individual features which suffer high gradient losses become more cautious, but rare informative features are not penalized. Hence, Adagrad resists overtraining, especially on small datasets, by annealing the learning rate for common features. This makes it more robust to hyperparameters and achieve better convergence.

Adagrad did not perform markedly better with the max-margin classifier. We hypothesize this is because the gradient is linear, so over-confidence in the hypothesis function does not as readily induce large gradient errors.

L-BFGS. For logistic regression and the neural networks, we use the quasi-Newton method L-BFGS, which maintains a linear gradient history to approximate the inverse Hessian, and line search in order to approximate the function minimum. Because it uses storage space linear in the size of θ , it is tractable to use on nontrivial feature sets where the inverse Hessian is computationally intractable.

5 Feature Reduction

The initial feature vector size was on the order of 10,000 features. Since getting datasets was hard and ultimately only 845 song transitions were used, there was a tremendous need for regularization. On manual inspection of the weights on the features, it was observed that the weights on the features on either size of the middle 1000 features were considerably smaller, almost negligible, as compared to the middle 1000. This indicated that most of the information was encoded in the middle 1000 features, which corresponded to roughly 1 second in the transition. The rest of the features seemed to be highly correlated. Thus, the feature set was reduced to prevent over-fitting and allow regularization to be more effective.

6 Results

6.1 FFT - Quantitative

Our results can be summarized in figures 1 and 2.

The metric being used is the positive F1 scores, which gives a method of measuring the model's accuracy. It takes into account the precision of the model (i.e. the number of transitions correctly predicted out of the total number of songs predicted) and the recall (the number of transitions relevant that were correctly fetched out of all pos-

Algorithm\Results	Training F1	Test F1
Logistic Regression	0.65	0.62
(Reduced Features)[Baseline]		
Logistic Regression	0.85	0.82
(Full Features) [Baseline]		
Hinge Loss (Reduced Features)	0.91	8.0
Hinge Loss (All Features)	0.99	0.87
Shallow Neural Network	0.88	0.89
(Reduced Features)		
L-BFGS Neural Network	0.94	0.94
(Reduced Features)		

Figure 1: Results table using F1 Scores

sible correct transitions).

As seen from the table, the reduced feature model for Logistic regression gave the the least performance of all possible models. It is obvious that it gave less performance than the full model, which overfit due to the extremely large feature vector. Hinge loss, as expected, does better due to the max margin nature of the classifier. The reduced model again has lower performance. For both these cases, Adagrad allows for an appropriate learning rate to maximize performance, though not preventing overfitting as much.

The particularly interesting models were the Neural Networks. The results for Neural Networks for different hidden layers and iterations can be seen in Figure 2.

NN Features\Results	Training F1	Test F1
10 hidden layer, 150 iterations	0.86	0.85
25 hidden layer, 150 iterations	0.88	0.89
50 hidden layer, 150 iterations	0.88	0.9
5 hidden layer, L-BFGS	0.95	0.96
10 hidden layer, L-BFGS	0.99	0.95
20 hidden layer, L-BFGS	0.96	0.96

Figure 2: Neural Net results using F1 Scores

As seen from the Figure 2., we have that the L-BFGS NN with 20 hidden layers gives the best test and train error, in that order. 150 iterations, in combination with Adagrad, was seen to give the sweet spot in training. Adagrad additionally prevented overfitting the NN by choosing appropriate learning rates. We can also observe that as the number of hidden layers increases from 10 to 20, the F1 score growth decrease and converges. We take the score having the larger train F1 score in case the test f1 scores are the same.

We hypothesize that L-BFGS does better in training the same network architecture because 1) the dataset is small, so stochastic methods do not offer a large speedup, and 2) we run L-BFGS with a narrower convergence criteria. Adagrad

still does better than our grid-searched learning rates, but Adagrad is too conservative in its later iterations in the size of its updates compared to L-BFGS; A momentum-based subgradient method might not have this limitation.

6.2 FFT - Qualitative



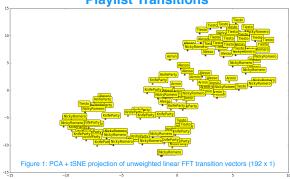


Figure 3: 2-D Projection of Feature Vectors using PCA + tSNE

Figure 3 shows an embedding of the FFT features into \mathbb{R}^2 for positive transitions of a few artists. When we added the artist name as a label, we discover that waveforms for song transitions are very close for Tiesto and KnifeParty, but a minority of DJ's (Nicky Romero) have wide-spanning embeddings, meaning that the transition methodologies are *not* consistent between artists. Despite this, we seem to predict each artist's transitions fingerprint precisely.

Listening to the mispredicted transitions, it appears that the transitions from progressive high frequencies to a series of low frequencies (i.e. a "drop") are the main sources of error. This is likely due to many curated playlists incorporating several of these types of transitions into their setlists, which in turn complicates the classification of these transitions. The characteristics of mispredicted transitions were entirely composed of second songs that contained a strong percussive element within the first few seconds.

Despite the great results and relatively few predictions, the p-values indicate a lot of relatively insignificant features. The null hypothesis was that the features are insignificant. Accordingly, there were 200 significant features, as measured by p-values < 0.7. They may be statistically insignificant, but that does not imply that there is no predictive effect of those features. For the remain-

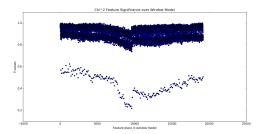


Figure 4: Feature Significance using p-values

ing, there is only a mild predictive effect. Thus, our hypothesis of using only the middle 1000 features was justified due to the highest significance for those values.

6.3 MFCC - Quantitative

Algorithm\Results	Training F1	Test F1
Logistic Regression (Full Features)	0.92	0.76
Hinge Loss (Full Features)	0.98	0.86
Neural Network (Full Features), L-BFGS, 5 Hidden Layers	0.98	0.84
Neural Network (Full Features), L-BFGS, 50 Hidden Layers	1.00	0.87

Figure 5: MFCC Transition Features Results using F1 scores

The MFCC feature transitions did not give results even close to those from the FFT. The best result came from the Neural Network with L-BFGS with 50 hidden layers, which had a score of 0.87. Despite supposedly being a better distance metric for music in general, it failed to be a better predictor than the FFT for good playlists transitions. To get an idea of the feature significance, p-values were tried to be found for the MFCC features. But their varied nature on either side of the y-axis (MFCC's can be negative) made it impossible to use the χ^2 distribution for p-values. Nevertheless, it can be deduced that fewer features were important and the significance of those features was lower than those of the FFTs.

7 Conclusion

The models and results presented in this paper prove that transition-based features are strongly predictive of curated music playlists and hence can be used for the same, at least for EDM music. It is also shown that transition based MFCC features are not as good as their FFT counterparts for strong predictive schemes. Finally, based on the p-value statistics of the features, it can be deduced

that there are certain frequency bins in every sample that are heavily predictive. Other features in conjuncture with these frequency bins can be used to make better predictive models.

References

- G. Bonnin and D. Jannach, "Automated generation of music playlists: Survey and experiments," ACM Comput. Surv., vol. 47, pp. 26:1–26:35, Nov. 2014.
- [2] J.-J. Aucouturier and F. Pachet, "Scaling up music playlist generation," in *Multimedia* and *Expo*, 2002. *ICME '02. Proceedings. 2002 IEEE International Conference on*, vol. 1, pp. 105–108 vol.1, 2002.
- [3] A. Flexer, D. Schnitzer, M. Gasser, and G. Widmer, "Playlist generation using start and end songs.," in *ISMIR* (J. P. Bello, E. Chew, and D. Turnbull, eds.), pp. 173–178, 2008.
- [4] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims, "Playlist prediction via metric embedding," in *Proceedings of the 18th ACM* SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, (New York, NY, USA), pp. 714–722, ACM, 2012.
- [5] F. Maillet, D. Eck, G. Desjardins, and P. Lamere, "Steerable playlist generation by learning song similarity from radio station playlists," in *Proceedings of the 10th Interna*tional Conference on Music Information Retrieval. 2009.
- [6] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," Acoustics, Speech and Signal Processing, IEEE Transactions on, vol. 28, no. 4, pp. 357–366, 1980.
- [7] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Tech. Rep. UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.