A Sep2Tree Model for Recognizing Synthetic Bach Chorales

First Author
Affiliation1
author1@smcnetwork.org

Second Author
Affiliation2
author2@smcnetwork.org

Third Author
Affiliation3
author3@smcnetwork.org

Fourth Author
Affiliation4
author4@smcnetwork.org

ABSTRACT

Relation Join has been applied to generate music harmonic sequences in a given composer or genre style. Compared to other music generation method, Relation Join does not require expert knowledge or estimation of any probabilities while generating a massive number of sequences. However, whether the generated compositions are distinguishable from the original ones is still a question that has not been explored. The evaluation of synthetic music sequences is considered subjective and hard to quantify. In this paper, we formed the evaluation problem as a classification task where the original and synthetic music sequences are assigned with two different labels. The evaluation of synthetic music sequences can then be quantified using the accuracy of classification experiment. In this paper, besides the traditional state-of-art classifiers, we propose a Seq2Tree network based deep learning model to classify synthetic and original music sequences. Our model extends the Long Short Term Memory network (LSTM) to take advantage of the characteristics of music data. The results show that our methods successfully classify 98.75% of the corpus that contain both original and synthetic Bach sequences, while the best performance of other machine learning classifiers is 81.63%.

1. INTRODUCTION

Relation Join, as a very new music generation method, has many advantages over classical generation methods [1], because all the previous methods either depend on music knowledge, or use machine learning techniques that have to estimate the probability of a music sequence. For example, Experiments in Music Intelligence (EMI), a music generation system proposed by Cope (1992) [2] uses a pattern matcher combined with Augmented Transition Network (ATN) , Hidden Markov Model (HMM) [3] requires the estimation of the transition matrix and the probability distribution of the hidden states [4, 5], or Genetic Algorithms (GAs) [6] requires a fitness function as evaluation that must be trained. Ni et al. [1] displays part of the synthetic Bach sequences generated by Relation Join, all of which sound very like Bach original work for general human listeners. However, despite the success and novelty of Relation Join, they did not provide any quantitative evaluation for the synthetic music sequences.

In this paper, we form the evaluation problem as a classification problem. Classification in machine learning is the problem of identifying the labels of a new observation based on a training set of observations whose labels are known. If we assign original and synthetic music sequences with different labels, the question of how distinguishable are the computer compositions from the human compositions becomes a classification problem, in which accuracy of the classification result can be used as the criteria.

Various classification methods have been used in recognizing music with different genres, music from different countries, and music composed by different composers etc., which can be grouped into two groups. The first group of methods calculates the statistics from music data, like count of notes, pitches, etc. [7, 8, 9, 10, 11, 12], and feeds the statistics into some classifiers to predict the labels of the test set. The classical classifiers that have been used to classify music sequences include: the k-Nearest Neighbour Classifier (kNN) [13], Support Vector Machine Classifier (SVM) with kernel [14, 15], Multi-layer Perceptron Classifier (MLP) [16, 12], and so on. The other group uses a sequential model like HMM [17, 18] on certain representations of music sequences. As an example, Chai et al.[18] estimate HMM on Helmut Schaffrath's Essen Folksong Collection using four different representations of melody including: absolute pitch, absolute pitch with rhythm, interval and contour. They achieve 77% accuracy when distinguishing between Irish and Austrian music.

By assigning different ground truth labels to human and computer compositions, the classification accuracy can then be used as a criteria to evaluate whether the synthetic music sequences are distinguishable from the original work. If the classification accuracy is high, it means the synthetic work is easily distinguishable. Instead, if the accuracy is relatively low, it means the synthetic work has higher similarity to the original work. The generation process can then be adjusted and improved based on the results. Our method can be combined with the music generation methods to help produce music sequences that are as pleasurable to human listeners as music composed by human composers.

To use classification as evaluation method, we need to design a good classifier. We believe that the music data con-

tain sequential information that traditional classifiers like SVM can not catch. In addition, the segments of chords formed by nearby chords could provide useful information for the recognition task. Based on our assumption, we introduce a new classifier called Seq2Tree classifier based on Long Short-Term Memory networks. We compare our model with existing state-of-art classifiers on two different features, N-Tuples feature, in which each element in the feature vector is a segment of N successive chords, and the Forte Class Number feature. Our model is introduced in Section 2. In Section 3, we demonstrate how the two features are extracted, and the classification experiments on the features. Section 4 concludes our work and discusses future work.

2. TREE STRUCTURED LSTM MODEL

2.1 Recurrent Neural Network

The Recurrent Neural Network (RNN) multilayer deep learning model has been introduced for sequential data [19]. The sequential input is fed into the network, and then transformed into some output. The output will be a label in classification tasks. The architecture of RNN is as in Figure 1.

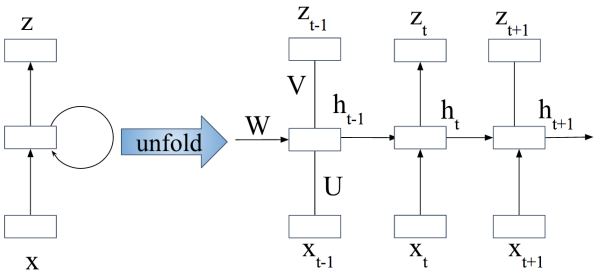

Figure 1. RNN architecture.

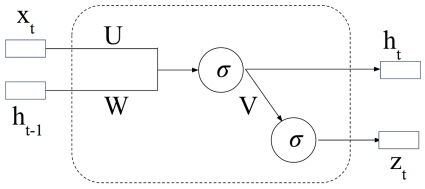

Figure 2. RNN unit.

Each RNN unit is shown as in Figure 2, where x_t is the input, h_t is the hidden state, σ is a sigmoid activation function ($\sigma(s) = \frac{1}{1+e^{-s}}$), and z_t is the output at time t. The RNNs map input sequences to hidden sequences via ($W = W_{hh}, U = W_{xh}, V = W_{hz}$):

$$h_t = \sigma\left(Ux_t + Wh_{t-1} + b_h\right)$$

and map hidden sequences to output sequences via:

$$z_t = \sigma\left(Vh_t + b_z\right)$$

where W_{ij}'s are the weight elements that are the trained parameters, σ can be replaced with other element-wise nonlinearity activation functions, such as a hyperbolic tangent, etc., $t \in \{1, 2, \ldots, T\}$.

2.2 Long Short-Term Memory Network

Long Short-Term Memory (LSTM) [20] is Recurrent Neural Network with long-term memory blocks. In RNN, back propagation flows through many layers. So the information flowing through neural net passes through many stages. During the process, gradients quickly vanish, and become too small to provide a learning signal for very deep layers [20], which makes the RNN not able to learn long-range dependencies well. LSTM addresses this problem by introducing memory blocks to RNN. Each block contains a recurrent memory cell and three multiplicative units: the input, output and forget gates. The memory blocks help preserve the signals and keep them large enough to be back-propagated through time and layers, thereby making learning long-term dependencies possible. The architecture of LSTM is as in Figure 3.

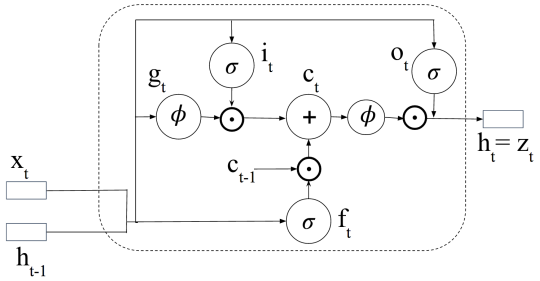

Figure 3. LSTM unit.

The LSTM can handle long-term temporal dynamics. It updates timestep t given x_t, h_{t-1} and c_{t-1} iteratively from $t = 1$ to T according to [21]:

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + b_i\right), \text{an input gate} \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right), \text{forget gate} \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + b_o\right), \text{output gate} \\
g_t &= \phi\left(W_{xc}x_t + W_{hc}h_{t-1} + b_g\right), \text{input modulation gate} \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t, \text{memory cell unit} \\
h_t &= o_t \odot \phi(c_t)
\end{aligned}
$$

where $\odot$ means element-wise multiplication. The input gate i_t controls the input flow to the memory cell. The block learns to forget previous memory and consider current input selectively through f_t and g_t. o_t then determines how much of the memory cell to be transferred to the hidden state.

2.3 Bidirectional LSTM Network

The basic idea of Bidirectional LSTM Network (BLSTM) [22, 23] is to incorporate future information into prediction. BLSTM processes the input data in both directions with two separate hidden layers (forward layer and backward layer), then feeds the two layers to the same output layer. The architecture of a bidirectional network is demonstrated in Figure 4.

2.4 Seq2Tree Classifier

For music data, we believe that the sequential information and potential tree structure of the input units are very

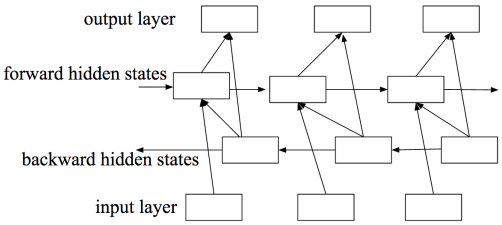

Figure 4. The architecture of a bidirectional network.

important for prediction tasks. For accurate predictions, we design a new LSTM network that reconstructs the tree structure from sequential inputs called Seq2Tree model. In the Seq2Tree model, we not only make use of the sequential music information, but also take advantage of the tree-structured dependency paths inherited in sequential inputs. Different from the original LSTM in which nodes always take their previous states as parent nodes, we select the parent node for each state from the previous state and its ancestors by adding a direction gate. Figure 5 shows a sample structure generated from our model.

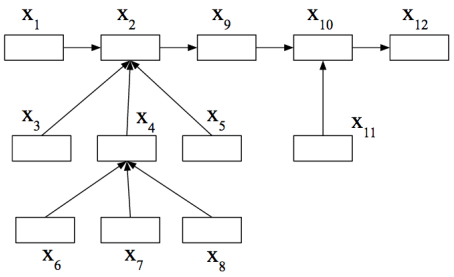

Figure 5. An sample structure generated by our model. x_2 is the ancestor node of x_4, x_5, x_6, x_7 and x_4 is the parent node of x_5, x_6, x_7. Root node of every subtree summarizes the output from its children nodes. Children nodes' hidden states are inherited from their parent nodes.

The Seq2Tree updates timestep t iteratively from $t = 1$ to T according to:

$$d_t^k = \theta(\sigma(W_{xd}x_t + W_{hd}h_k + b_d)), k \in \mathcal{K}, \text{direction gate}$$

$$h_{parent} = \prod_{d_t^k \neq 0} d_t^k h_k, k \in \mathcal{K}$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{parent} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{parent} + b_f)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{parent} + b_o)$$

$$g_t = tanh(W_{xg}x_t + W_{hg}h_{parent} + b_g)$$

$$c_t = i_t \odot g_t + f_t \odot c_{t-1}$$

$$h_t = g_t \odot tanh(c_t)$$

$$m_t = \sigma(W_{xm}x_t + W_{hm}h_t + b_m)$$

$$n_t = m_t \odot c_t,$$

$$c_t^l = c_t^l + n_t - \sum_l g_l n_t, l \in \mathcal{L}$$

$$h_t^l = o_t^l \odot tanh(c_t^l), l \in \mathcal{L}$$

where θ represents a binary thresholding function, $\odot$ means element-wise multiplication, $\mathcal{K}$ contains all indexes of the ancestor nodes of the previous state h_{t-1} and $\mathcal{L}$ contains the indexes of current node's ancestors. Note that the sizes of $\mathcal{K}$ and $\mathcal{L}$ do not necessarily match. h_{parent} stands for the parent selected for node h_t. For each node h_t, we put node h_{t-1} and the indexes of all the ancestor nodes of node h_{t-1} in a parent candidate list $\mathcal{K}$ in leaf-to-root order. For example, if h_2 is the parent node of h_4 and h_4 is the parent of x_6, when it comes to h_7, its candidate parent list will be $\mathcal{K} = \{h_6, h_4, h_2\}$. Then, calculate the decision gate for each node in the candidate list, until we find a positive gate, or we reach the root of the tree. If all the decision gates are negative, the node has no parent node.

For our task, we add a root node to the tree structure generated by our model. More specifically, we add an extra node as the parent node for the highest level nodes generated by the model. For example, we will add a new node x_0 as the parent node of $x_1, x_2, x_9, x_{10}, x_{12}$ in Figure 5. The root node's state is updated through the root states of its non-empty subtrees. Then we build a softmax classifier on top of the root node. The output of the softmax function is a probability distribution over all possible classes:

$$p(y = j|h_{root}) = \frac{e^{h_{root}^T w_j}}{\sum_c e^{h_{root}^T w_c}}, c \in \mathcal{C}$$

$$\hat{y} = argmax_c\, p(y = c|h_{root})$$

where w_j is the weight vector for label j, h_{root} is the hidden states of the root node, and $\hat{y}$ is the predicted label of the input.

We use cross entropy loss of the predicted label $\hat{y}$ as our loss function:

$$J(\theta) = -\frac{1}{|\mathcal{C}|}\sum_{y=c} p(\hat{y}) \cdot log\, p_\theta(\hat{y}|h_{root}), c \in \mathcal{C}$$

where $\mathcal{C}$ is set that contains all possible labels, $|\mathcal{C}|$ is the size of the set, $p(\hat{y})$ represents the probability that the input music sequence actually belongs to the class $\hat{y}$, and $p_\theta(\hat{y}|h_{root})$ indicates the predicted probability that the input sequence falls in the class $\hat{y}$ with the parameter set θ.

3. EXPERIMENTS

In this section, we compare our methods to existing state-of-art classifiers in machine learning.

3.1 Data

We generate 400 synthetic chord sequences in Bach style with Relation Join [1], and put them together with 400 original Bach chorales. So we will have a corpus with 800 sequences. In the Relation Join process, the length of each chord tuple is set to 5, and the number of overlapping chords in successive tuples is set to 3.

3.2 Features

We use two sets of features in this paper.

3.2.1 N-Tuples

The first feature we used in this paper is called N-Tuples. There are five steps in the feature extraction process. First, we extract all the chords from the corpus. We ignore the octave designation for all the notes, replace each note with their pitch class, remove the duplicate notes, and sort all

the notes. Then we adjust each chord based on the key signature of the sequence the chord in. Third, we assign a unique index to all the chords extracted from the whole corpus, and replace the chords in each sequence with the unique index. Fourth, we extract all distinct N-Tuples of successive chords. Fifth, for each sequence in the corpus, we count each of the types of N-Tuple, and normalize the counts to be components of the feature vector. For details see `http://haralick.org/music/ntuples_xiuyan.pdf`

Example 3.1 *Suppose we have a corpus that contains two sequences, sequence 1 is the first 10 chords from Bach bwv26.6 (Figure 6) and sequence 2 is the first 10 chords from Bach 66.6 (Figure 7).*

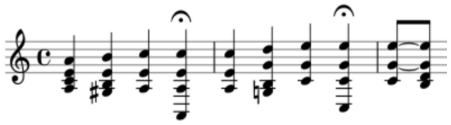

Figure 6. The first 10 chords of Bach bwv26.6.

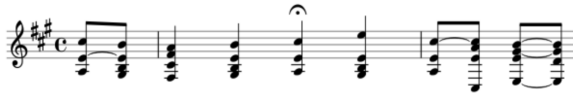

Figure 7. The first 10 chords of Bach bwv66.6.

After the first two steps, we will get

$$q_1 = [(1,6,9),(1,5,8),(1,6,9),(1,6,9),(1,6,9),\\ (4,8,11),(1,4,9),(1,4,9),(1,4,9),(1,4,8,11)]$$

from the sequence 1, and get

$$q_2 = [(3,7,10),(2,5,10),(0,3,7),(2,5,10),(3,7,10),\\ (2,5,10),(3,7,10),(3,7,10),(2,5,10),(2,5,8,10)]$$

from sequence 2. There are 8 different chords in the corpus. If we assign an unique index to each chord as in the following:

$$\{(0,3,7):0,(1,4,8,11):1,(1,4,9):2,(1,5,8):3,\\ (1,6,9):4,(2,5,8,10):5,(2,5,10):6,\\ (3,7,10):7,(4,8,11):8\}.$$

Let $N = 3$, in the fourth step, we extract all 3-Tuples from q_1 and q_2. Finally, the feature vector extracted from q_1 and q_2 are as following:

$$q_1 = [0,0.125,0.125,0.125,0.125,0.125,0.125,0.125,\\ 0,0,0,0,0,0,0,0.125]$$

$$q_2 = [0.125,0,0,0,0,0,0,0,0,0.125,0.125,0.125,0.125,\\ 0.125,0.125,0.125,0]$$

3.2.2 Forte Class Number

The second feature we use is the Forte Class Number of the chord. Forte number is a pair of numbers used by Allan Forte to present pitch class set [24]. The first number represents the number of different pitch classes in the pitch class set. The second number is the order number of the pitch class set in Forte's ordering of all pitch class sets with the same number of pitches. The Forte class number is the number of the Forte set class within the same set group. For example, the Forte class number of chord (C, E, G) is 11. Then we extract the first 30 chords from each sequence in the corpus.

Example 3.2 *Take the first line of bwv26.6 in Figure 8 as an example. The feature vector we extracted from the sequence using forte class number is:*

$$[11,11,11,11,11,11,11,11,11,26,11,20,27,10,2,\\ 11,27,11,16,11]$$

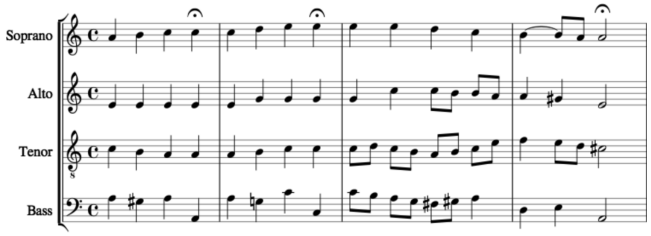

Figure 8. The first line of Bach bwv26.6.

3.3 Baseline Classifiers

We use 12 traditional state-of-art classifiers in machine learning as our baseline classifier, which include: kNN, MLP, SVM with Linear kernel (Linear SVM) [25], SVM with Radial Basis Function kernel (RBF SVM) [25], Decision Tree Classifier (DT) [26], Random Forest Classifier (RF) [27], AdaBoost Classifier [28, 29], Guassian Naive Bayes Classifier (NB) [30, 31, 32], Gaussian Process Classifier (GP) [33], Linear Discriminant Analysis Classifier (LDA) [34], the Quadratic Discriminant Analysis Classifier (QDA) [35], and BLSTM Classifier [36].

3.4 Results

We randomly partition the whole corpus into five groups, use each of them as the testing set while the remaining four groups are used as the training set. All the classification results are the average of the five experiments as the test sets are rotated. Let N_o and N_s be the true number of original and synthetic sequences respectively. For each experiment, we calculate the number of original sequences that are classified as synthetic (N_{os}), the number of synthetic sequences that are classified as original (N_{so}), and the number of sequences that are classified correctly (N_c). Then accuracy is $\text{Accuracy} = N_c/N$. False positive rate (FPR) is defined as $\text{FPR} = N_{so}/N_s$. False negative rate (FNR) is defined as $\text{FNR} = N_{os}/N_o$. The classification accuracy on the corpus with all the classifiers we mentioned are summarized in Table 1. We find that for the N-Tuples features, Adaboost achieve the highest accuracy among the classifiers. We did not use our model on N-Tuple data, because the feature vectors are not sequential

Classifier	Accuracy	
	N-Tuples(%)	FCN(%)
kNN	66.00	63.75
NB	71.63	62.38
Linear SVM	47.50	58.63
RBF SVM	74.25	49.88
MLP	49.50	59.38
DT	74.13	68.13
RF	52.38	62.13
GP	67.25	63.75
LDA	66.50	59.25
QDA	52.13	64.50
AdaBoost	**81.63**	67.38
BLSTM	–	62.75
Seq2Tree	–	**98.75**

Table 1. Results of all classifiers on the corpus that contain 400 original Bach chorales and 400 computer compositions with Forte Class Number feature (FCN) and N-Tuples feature, the numbers in the table are based on the average of the five experiments.

data. For the Forte Class Number feature, our model outperform all the existing state-of-art classifiers with 98.75% accuracy.

To further identify the mistakes made by the classifiers on the Forte Class Number feature. We calculate the FPR, the percentage of the number of synthetic sequences that are identified as original over the true number of synthetic sequences, and FNR, the percentage of the number of original sequences that are identified as synthetic over the true number of original sequences. The results are summarized in Table 2.

Classifier	FPR (%)	FNR (%)
kNN	**36.75**	35.75
NB	**64.50**	10.75
Linear SVM	**52.25**	30.50
RBF SVM	22.25	**78.00**
MLP	**42.25**	39.50
DT	**48.50**	15.50
RF	38.50	**40.25**
GP	**36.50**	36.00
LDA	**66.50**	59.25
QDA	**44.00**	27.00
AdaBoost	**35.25**	30.00
BLSTM	36.25	**37.75**
Seq2Tree	**1.25**	**1.25**

Table 2. The FPR and FNR of all classifiers over Forte Class Number feature (FCN). the numbers in the table are based on the average of the five experiments.

We can see from Table 2 that, our classifier has the the lowest FPR and FNR. For other classifiers, FPR is generally higher than FNR, which means most classifiers tend to classify a synthetic sequences as original work, which is consistent with our impression that the synthetic sequences are very much like Bach original work for human listeners.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we described a method to distinguish between human composed music and computer generated music. Our classifier is based on the LSTM network. Our model reconstructs the tree structure from the input sequences. It can incorporate the sequential information in music data into the prediction. Furthermore, because of the tree structure of our model, it also takes advantage of the fact that the segments of chords also carry the information helpful for the classification task. Our model is the first one that distinguishes between human compositions and computer compositions using a tree structured LSTM. We tested our method on a corpus consisting of 400 original Bach chorales and 400 synthetic Bach-style sequences using the N-tuples features and the FCN features. We compared our method with existing state-of-art classifiers. For the N-tuples features that carry only segments information, the best performance is 81.63% achieved by the Adaboost classifier. For the FCN features that carry the sequential information, we achieve 98.25% accuracy, which means our model can successfully discriminate between most original and synthetic music sequences. Our model significantly outperforms the existing state-of-art machine learning classifiers, which makes it possible to be used to evaluate any music generation method to help improve the quality of computer compositions. In addition, it probably means our model is able to catch the features in music sequences which are important in classification tasks. In other words, our model has potential use in other music classification tasks, such as music style recognition, or composer recognition. For future work, we will further explore the features in the music sequences that make the computer compositions distinguishable from the human compositions when the synthetic sequences are already very "Bach" for general human listeners.

5. REFERENCES

[1] X. Ni, L. Liu, and R. Haralick, "Music Generation with Relation Join," in *CMMR*, 2016, pp. 286–297.

[2] D. Cope, "Computer Modeling of Musical Intelligence in EMI," *Computer Music Journal*, vol. 16, no. 2, pp. 69–83, 1992.

[3] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[4] A. Van Der Merwe and W. Schulze, "Music Generation with Markov Models," *IEEE MultiMedia*, vol. 18, no. 3, pp. 78–85, 2011.

[5] D. Conklin, "Music Generation from Statistical Models," in *Proceedings of the AISB Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, 2003, pp. 30–35.

[6] N. Tokui, H. Iba *et al.*, "Music Composition with Interactive Evolutionary Computation," in *Proceedings of the 3rd International Conference on Generative Art*, vol. 17, no. 2, 2000, pp. 215–226.

[7] C. Pérez-Sancho, P. J. P. De León, and J. M. I. Quereda, "A Comparison of Statistical Approaches to Symbolic Genre Recognition." in *ICMC*, 2006.

[8] P. J. Ponce de León Amador, J. M. Iñesta Quereda, and D. Rizo Valero, *Mining Digital Music Score Collections: Melody Extraction and Genre Recognition*. Intech, 2008.

[9] C. McKay and I. Fujinaga, "Automatic Genre Classification Using Large High-Level Musical Feature Sets." in *ISMIR*, 2004, pp. 525–530.

[10] B. Thom, "Unsupervised Learning and Interactive Jazz/Blues Improvisation," in *AAAI/IAAI*, 2000, pp. 652–657.

[11] P. Toiviainen and T. Eerola, "A Method for Comparative Analysis of Folk Music Based on Musical Feature Extraction and Neural Networks," in *International Conference on Cognitive Musicology*, 2001, pp. 41–45.

[12] G. Buzzanca, "A Supervised Learning Approach to Musical Style Recognition," in *Music and Artificial Intelligence. Additional Proceedings of the Second International Conference, ICMAI*, vol. 2002, 2002, p. 167.

[13] P. Cunningham and S. J. Delany, "K-Nearest Neighbour Classifiers," *Multiple Classifier Systems*, vol. 34, pp. 1–17, 2007.

[14] F. R. Bach, G. R. Lanckriet, and M. I. Jordan, "Multiple Kernel Learning, Conic Duality, and the SMO Algorithm," in *Proceedings of the 21st International Conference on Machine learning*. ACM, 2004, p. 6.

[15] M. Gönen and E. Alpaydin, "Localized Multiple Kernel Learning," in *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 352–359.

[16] X. Glorot and Y. Bengio, "Understanding the Difficulty of Training Deep Feedforward Neural Networks," in *Aistats*, vol. 9, 2010, pp. 249–256.

[17] O. A. S. Carpinteiro, "A Self-Organizing Map Model for Analysis of Musical Time Series," in *Proceedings of 5th Brazilian Symposium on Neural Networks*. IEEE, 1998, pp. 140–145.

[18] W. Chai and B. Vercoe, "Folk Music Classification Using Hidden Markov Models," in *Proceedings of International Conference on Artificial Intelligence*, vol. 6, no. 6.4, 2001.

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[20] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2625–2634.

[22] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[23] A. Graves, N. Jaitly, and A.-R. Mohamed, "Hybrid Speech Recognition with Deep Bidirectional LSTM," in *Automatic Speech Recognition and Understanding (ASRU) IEEE Workshop*. IEEE, 2013, pp. 273–278.

[24] A. Forte, *The Structure of Atonal Music*. Yale University Press, 1973, vol. 304.

[25] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.* (2003) A Practical Guide to Support Vector Classification. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[26] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. CRC press, 1984.

[27] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[28] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of Online Learning and an Application to Boosting," in *European Conference on Computational Learning Theory*. Springer, 1995, pp. 23–37.

[29] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-Class Adaboost," *Statistics and Its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[30] I. Rish, "An Empirical Study of the Naive Bayes Classifier," in *IJCAI Workshop on Empirical Methods in Artificial Intelligence*, vol. 3, no. 22. IBM New York, 2001, pp. 41–46.

[31] G. H. John and P. Langley, "Estimating Continuous Distributions in Bayesian Classifiers," in *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[32] A. Y. Ng and M. I. Jordan, "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes," *Advances In Neural Information Processing Systems*, vol. 2, pp. 841–848, 2002.

[33] C. E. Rasmussen, "Gaussian Processes for Machine Learning." MIT Press, 2006.

[34] M. Welling, "Fisher Linear Discriminant Analysis," *Department of Computer Science, University of Toronto*, vol. 3, pp. 1–4, 2005.

[35] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Mullers, "Fisher Discriminant Analysis with Kernels," in *IEEE Signal Processing Society Workshop on Neural Networks Signal Processing*. IEEE, 1999, pp. 41–48.

[36] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.