Automatic Chord Extractor and AI Composer with Neural Network

Changmin Lee
KAIST / 20190519

Jeonghyun Kim
KAIST / 20180147

Abstract

Ordinary people have difficulty composing and arranging chord-based music since music work requires harmonics knowledge. To solve this problem, we implemented automatic *Chord Extractor* and *Chord Generator*. Using the MLP model, Viterbi algorithm, and chord change sensing, we implemented various models for *Chord Extractor* and compared them. For *Chord Generator*, we used LSTM because of the similarity between subsequent chords generation and text generation.

1 Motivation

Music is a culture and hobby that can be easily encountered anywhere. Even when you walk on the street, you can hear music from restaurants or commercial BGM from shops. We encounter music even when we are not aware of it. Lots of songs are composed and arranged accordingly, and harmonics knowledge and musical experience are required for this music work.

One of the important harmonistic elements, the chord, harmonizes the melody. Therefore, using a chord sheet, the melody conceived when composing can be completed as a single song or the melody of the original song can be reinterpreted differently when arranging it.

However, ordinary people have difficulty composing and arranging chord-based music due to a lack of harmonics knowledge. To solve this problem, we conducted the AI Composer project that informs the progress of chords automatically.

2 Related Work

(Lee and Slaney, 2008) used chromagrams to recognize chords. A chroma vector is a twelve-dimensional vector, each dimension representing spectral energy in a pitch class in a chromatic scale. Thus, we decided to use this chroma feature for our project. Also, they used the hidden Markov model to learn a chord progression.

(Fujishima, 1999) used pattern matching for chord recognition. Also, they detected chord change using the direction of the chroma vector.

(Rao et al., 2016) combined SVM with the Viterbi algorithm. Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states that caused the sequence of events observed in hidden Markov models. Here, the hidden state corresponds to the chord type in our problem. Viterbi algorithm is used for the temporal correlation of chords.

3 Method

Figure 1 shows the overall pipeline of the proposed model. Our final goal is to generate the music file playing an obtained following chords sequence. We have to perform two tasks to reach our final goal. Our first task is to implement a *Chord Extractor* model that extracts chromagrams from the training music data and classifies them into a suitable chord using it. The second task is to implement a *Chord Generator* model that predicts the following chord sequence using some initial chord sequence of a given music file. *Chord Extractor* consists of three stages, MLP, Viterbi, and CCS (Chord Change Sensing), and *Chord Generator* consists of LSTM cells.

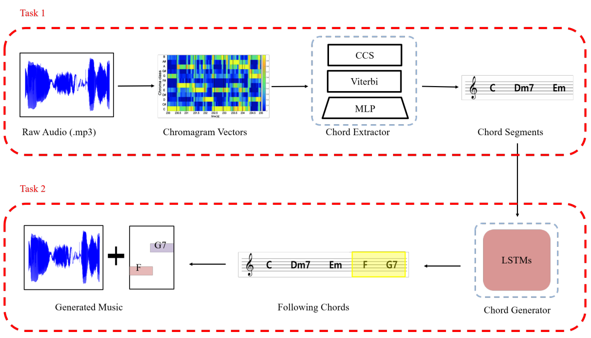


Figure 1: Overall pipeline of the proposed model

3.1 MLP

We have tried MLP classification to extract chords from chroma vectors. (Rao et al., 2016) introduced a temporal correlation support vector machine (TCSVM), which combined the SVM classification with Viterbi algorithm for temporal correlation of chords. However, since classical ML methods that do not use GPU require a lot of system RAM usage, the size of input data is limited. Also, while SVM has a fixed algorithm, MLP has many parameters that we can adjust such as the structure of hidden layers and hidden sizes. So MLP has more potential for performance improvement. Therefore, we decided to combine MLP and the Viterbi algorithm to

idx	chord	idx	chord	idx	chord
0	N	9	A♭:maj	18	F:min
1	C:maj	10	A:maj	19	F♯:min
2	C♯:maj	11	B♭:maj	20	G:min
3	D:maj	12	B:maj	21	A♭:min
4	E♭:maj	13	C:min	22	A:min
5	E:maj	14	C♯:min	23	B♭:min
6	F:maj	15	D:min	24	B:min
7	F♯:maj	16	E♭:min	25	X
8	G:maj	17	E:min		

Table 1: index2chord. idx indicates index corresponding to each chord.

proceed with the chord classification instead of SVM in this project.

3.2 LSTM

If we thought of one chord as a token, generating natural subsequent chords would be similar to natural text generation. Various types of recurrent neural networks are used in tasks that deal with sequences such as natural language processing and video processing. And one of them, LSTM, shows good performance in many situations. So we implemented Chord Generator using LSTM.

4 Experiments and Results

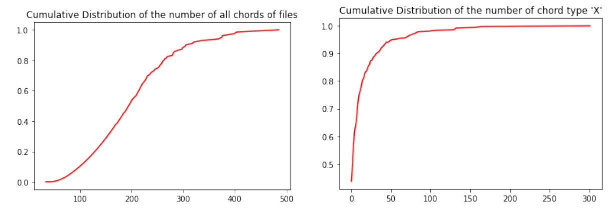
4.1 Dataset

Due to music copyright, we used the McGill Billboard Project dataset which consisted of 890 songs from (Burgoyne et al., 2011). This dataset contains timestamps, chords (major, minor), and non-negative-least-squares chroma vectors from each song. The chords information is the ground-truth set. The non-negative-least-squares chroma vector is 'Chromagram and Bass Chromagram' which is a 24-dimensional chromagram, consisting of both bass chromagram and chromagram. This chromagram is calculated with sampling rate = 44,100 Hz and hop length = 2,048.

4.2 Preprocessing

First, we extracted chord information from the dataset to make a chord dictionary. As a result, 44 chord types were found, but there were duplicates such as F♯ and G♭. Thus, We created the chord dictionary which maps chords to indices, which were integers between 0 and 25, and duplicate chords pointed to the same index like Table 1. Index 1 to 12 indicate major chords, index 13 to 24 indicate minor chords, and index 0 & 25 indicate non-chords.

In Figure 2a, the number of chords varies considerably from one music file to another. So, we chose music files containing chords less than 200. Also, there are 2 non-chord types, 'N' and 'X'. 'N' usually appears at the start and end point of the song, which has no melody. However, 'X' usually appears in the middle part of the song, which implies that part doesn't contain chord information such as the rap part. Thus, considering Figure 2b, we decided to choose music files containing 'X' less than 10. We then split these chord scores files into train files, validation files, and test files at a rate of 0.8, 0.1, and 0.1.



(a) Chords distribution (b) X chord type analysis

Figure 2: Chord analysis

Figure 3 shows the number of occurrences of each chord type converted into indices. We can see that the dataset has data bias. For instance, the G:maj chord occurred 6632 times, while A♭:min occurred 355 times. We thought that this data bias would affect the accuracy of *Chord Extractor*, so we sampled only a certain number of each chord type. We will compare the sampling result to the non-sampling result later.

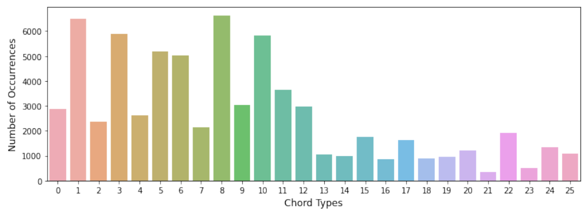


Figure 3: Imbalanced Data

4.3 Chord Extractor

4.3.1 MLP

We collected all NNLS chromagrams and chord types of 470 songs to obtain input data for MLP. Our neural network consists of some linear layers, Batch Normalization layers, ReLU activation, Dropout layers, and a final linear layer with softmax activation. Figure 4 shows the layers of the network. Since the size of the input chromagram is 24 (12 pitch classes for bass note and treble note) and the number of chord types is 26 (twelve base pitch classes for two chord templates - major, minor - and additional two types - N, X), input size and output size are 24 and 26, respectively.

And we compared the performance by sampling the data at various sizes to check the effect of imbalance in the training data. As a result of chord prediction using MLP, there were some music files with high accuracy as shown in Figure 5, but there was a big difference in accuracy for each file. That's because it didn't detect the minor chord at all in some files. And sampling handled this problem quite effectively. File 0605, which consists

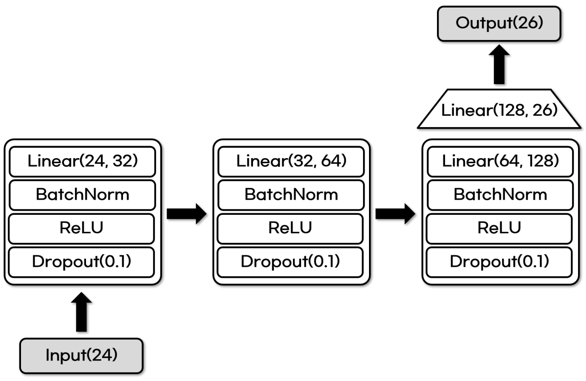


Figure 4: MLP layers for *Chord Extractor*

of many minor codes, showed a 47.41% increase in accuracy compared to before sampling as shown in Figure 6.

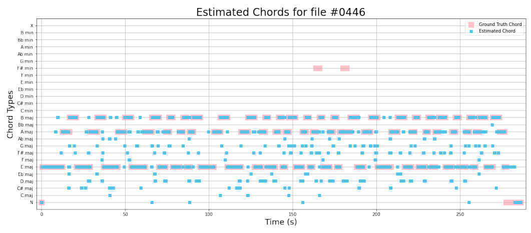


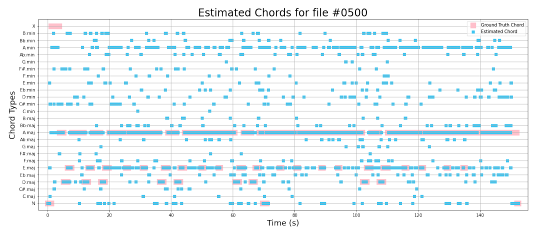
Figure 5: The chord prediction plot of the file with the highest accuracy (84.57%)

4.3.2 Viterbi

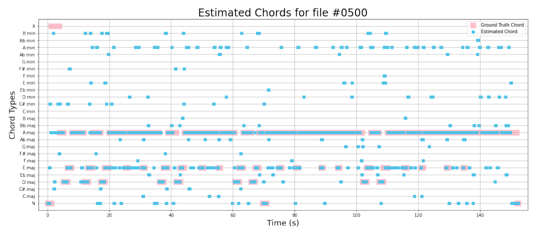
Currently, our *Chord Extractor* base model is implemented with MLP to categorize the chord types using NNLS chromagrams of an interval of about 46ms. Chromagrams are obtained from music files with sampling rate = 44,100Hz and hop_length = 2,048. Since chromagrams are measured at very short intervals, there is a very high probability that those measured in a similar time period have the same chord type, but MLP cannot use temporal correlation of chord sequence. So Viterbi algorithm was used to smooth it.

4.3.3 Chord Change Sensing

We checked the tendency of the incorrectly predicted chords after Viterbi and we found that the model changes the chord earlier than the ground truth as shown in Figure 9a. Figure 8 is a plot of the distance of two adjacent chroma vectors according to the chromagram index. We know that the chord change occurs at chromagram indices 144, 155, and 188. However, our model changes the chord earlier than the actual one. However, we found that in many cases the actual chord change occurs when the chroma distance is less than 1.0. So, even if the model predicts a chord change, we waited until it becomes less than a certain threshold before changing the chord. As a result, we were able to see that the timing of change became similar to ground truth in Figure 9b.

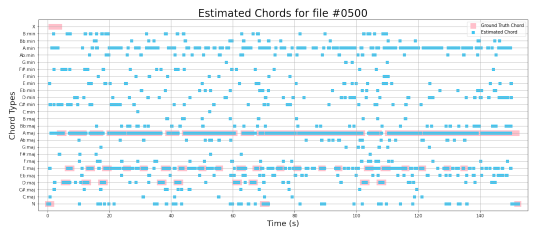


(a) No sampling (12.51%)

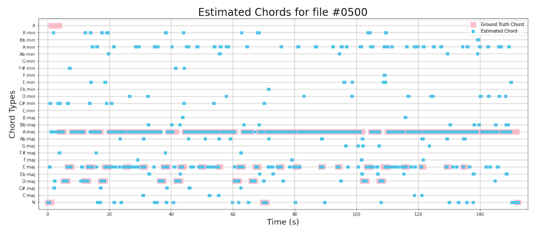


(b) Sampling Number 30,000 (59.92%)

Figure 6: Improved accuracy of File 0605 by sampling the training data



(a) Before Viterbi (52.80%)



(b) After Viterbi (65.90%)

Figure 7: Improved accuracy of File 0500 by applying Viterbi Algorithm (Sampling Number 10,000)

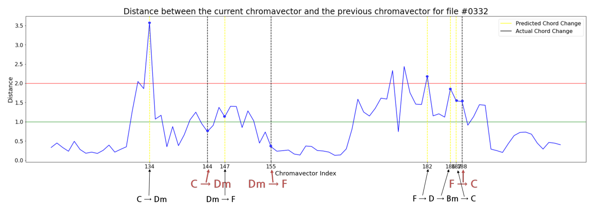
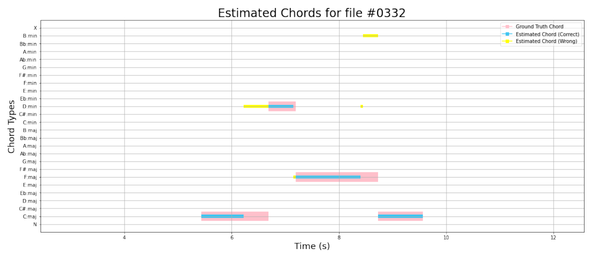
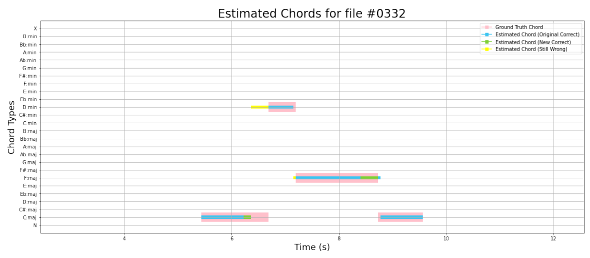


Figure 8: Distance between the current chroma vector and the previous chroma vector for file 0332



(a) Before CCS (64.55%)



(b) After CCS (70.06%)

Figure 9: Improved accuracy of File 0332 by applying Chord Change Sensing (Sampling Number 10,000)

model	sampling	accuracy (%)			
		M	MV	MC	MVC
ver 0	None	55.84	58.07	56.85	58.78
ver 1	10,000	57.80	62.45	59.24	63.48
ver 2	30,000	57.63	61.77	58.99	62.71
ver 3	50,000	56.93	60.51	58.31	61.50
ver 4	70,000	56.92	60.04	58.12	60.90
ver 5	90,000	**59.74**	**63.43**	**61.05**	**64.32**
ver 6	110000	57.03	59.83	58.07	60.64

Table 2: *Chord Extractor* accuracy result. We experimented MLP model for 7 versions according to the sampling number. Also, we checked accuracy when applying M, MV, MC, MVC for each model. M is MLP model, V indicates Viterbi algorithm, and C means chord change sensing.

4.3.4 Result

We measured validation accuracy by changing the sampling number and model stage. Table 2 and Figure 10 shows the result. The MVC model with sampling number 90,000 showed the highest accuracy. Overall, using Viterbi algorithm and CCS together with MLP was the most effective. When used alone in MLP, Viterbi algorithm was more effective than CCS. Also, using sampling number 90,000 was the most effective overall.

Using the MVC model with sampling number 90,000, the validation accuracy was measured while changing the CCS threshold. Table 3 and Figure 11 shows the result. The CCS threshold showed the highest accuracy around 1.0.

As a result of chord prediction on 59 test files, the average accuracy was 63.69%. Figure 12 shows the kernel density of the test accuracy.

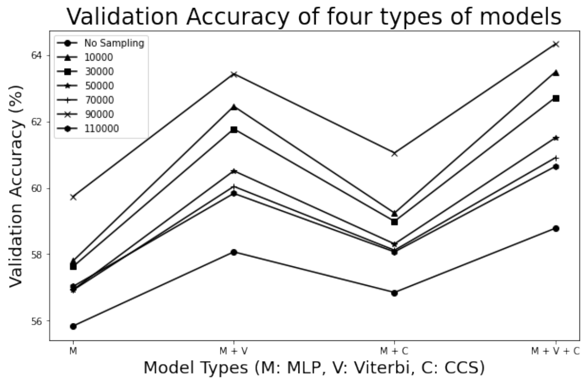


Figure 10: Validation accuracy of four types of models

threshold	acc (%)	threshold	acc (%)
0.5	57.86	1.5	64.26
0.6	61.66	1.6	64.17
0.7	63.35	1.7	64.08
0.8	64.18	1.8	64.02
0.9	64.41	1.9	63.93
1.0	64.32	2.0	63.86
1.1	64.46	2.1	63.78
1.2	64.45	2.2	63.73
1.3	64.36	2.3	63.67
1.4	64.28	2.4	63.62

Table 3: Adjusting CCS Threshold. acc means accuracy.

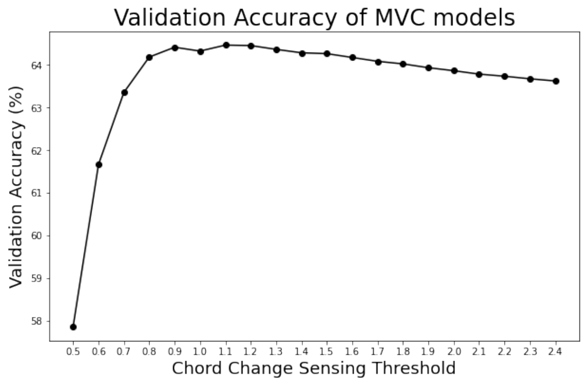


Figure 11: Validation accuracy of MVC model

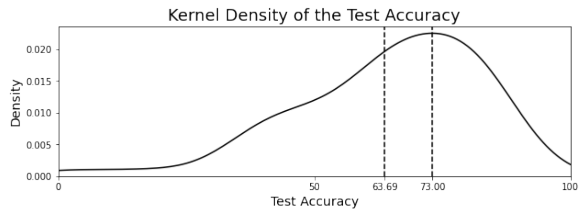


Figure 12: Kernel Density of the Test Accuracy

4.4 Chord Generator

We collected all chord sequences of 470 songs to obtain input data for LSTM. Our neural network consists of an embedding layer, two LSTM layers, and the final linear layer with softmax activation. Figure 13 shows the layers of the network. *Chord Generator* returns probabilities for 26 types of subsequent chords for a given chord.

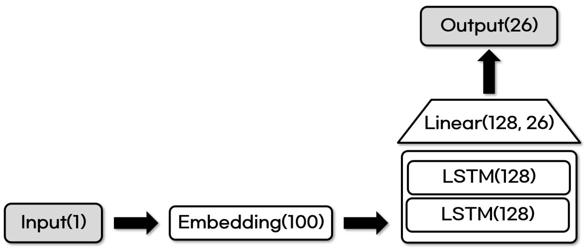


Figure 13: LSTM layers for *Chord Generator*

We generated the second half chords using the first half chord sequence of file 1007, which showed high accuracy in Task 1. For the first half sequence of file 1007 to be fed as input data of *Chord Generator*, both (1) *Ground Truth* sequence and (2) sequence *Predicted* by *Chord Extractor* were used. And for the prediction policy that selects the following chord, we tried both (1) *Argmax*, which selects the most probable one, and (2) *Random*, which predicts by generating a categorical random variable using a probability distribution.

4.4.1 Result

Figure 14 shows the results generated by *Chord Generator* with two different types of input data: *Ground Truth*, *Predicted*, and prediction policy: *Argmax*, *Random*. Because of the high temporal correlation of the chord sequence, the chords generated by the *Argmax* policy tended to continue with the previous chord. We could see that the chords generated from both types of input data by the *Random* policy were musically natural.

5 Conclusion

This project implemented an automatic chord extractor and AI composer. The automatic chord extractor used MLP with the Viterbi algorithm and chord change sensing technique. We used LSTM to implement AI composer.

Further research should reinforce the following points. First, our project detected the chord change only. In future work, we will normalize the tempo for various songs and separate bars. Also, we only considered major and minor chords. we will add various kinds of chords such as seventh, augmented, or diminished chords, and normalize tonality.

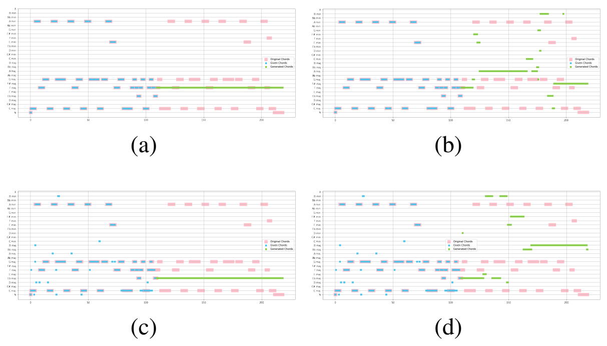


Figure 14: Results generated by *Chord Generator* with two different types of input data and prediction policy. (a) Input: *Ground Truth*, Prediction Policy: *Argmax*, (b) Input: *Ground Truth*, Prediction Policy: *Random*, (c) Input: *Predicted*, Prediction Policy: *Argmax*, (d) Input: *Predicted*, Prediction Policy: *Random*

6 Contribution

6.1 Changmin

Changmin implemented MLP base model for *Chord Extractor* and LSTM model for *Chord Generator*. He wrote final report section 3, 4.3, and 4.4.

6.2 Jeonghyun

Jeonghyun conducted data preprocessing and analysis. Jeonghyun implemented Viterbi algorithm and Chord Change Sensing for *Chord Extractor*. She wrote final report section 1, 2, 4.1, 4.2, and 5.

References

John Burgoyne, Jonathan Wild, and Ichiro Fujinaga. 2011. An expert ground truth set for audio chord recognition and music analysis. pages 633–638.

Takuya Fujishima. 1999. Realtime chord recognition of musical sound: a system using common lisp music. In *ICMC*.

Kyogu Lee and Malcolm Slaney. 2008. Acoustic chord transcription and key extraction from audio using key-dependent hmms trained on synthesized audio. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):291–301.

Zhongyang Rao, Xin Guan, and Jianfu Teng. 2016. Chord recognition based on temporal correlation support vector machine. *Applied Sciences*, 6:157.