AUTOCHORD: AUTOMATIC CHORD RECOGNITION LIBRARY AND CHORD VISUALIZATION APP

Christopher John Bayron
Independent
Manila, Philippines
cjdbayron@gmail.com

ABSTRACT

In this paper I present *autochord*, a bundle of tools for automatic chord recognition, comprising of 1) a Python library that performs Audio Chord Estimation (ACE), and 2) a JavaScript app for visualizing and comparing chord labels, all open-source and freely available online.[1] The Python library (hereinafter referred to as *autochord.py*) can generate MIREX-style chord labels[2] which can be interpreted and visualized by the app (hereinafter referred to as *autochord.js*). Used together, this toolset functions as a full chord recognition app.

1. INTRODUCTION

This project was started with two primary goals in mind: to build an ACE model using machine learning (ML), and to deploy it in an easy-to-use chord recognition web app for free public consumption. Furthermore, the app shall run purely on client-side i.e., all processing happens only in the user's web browser, hence, no song data or any derived features are sent over the Internet. This eliminates any concerns with the privacy of the song being processed.

One tool that proved useful for client-side ML is *TensorFlow.js* (TFJS), a JavaScript port of *TensorFlow*.[3] The rich stack of TensorFlow tools allow for conversion of Python models to be runnable in a TFJS-based web app, leading to decide to use TensorFlow for the ACE model.

The best trained model, however, while working well in Python, contains modules that are not portable to TFJS as of writing. Hence, the decision to package the model in *autochord.py*, keeping all chord recognition processing in Python, and building a simpler app, *autochord.js*, which can display the generated chord labels from *autochord.py*.

2. MODEL DEVELOPMENT

2.1 Training Data

The chord recognition model was trained on The McGill Billboard Project dataset[4] [1], particularly using its chroma vectors feature set and MIREX-style chord annotations. This was deemed the favorable dataset due to containing popular songs, likely to be what the public would use chord recognition apps for, and more importantly, due to accessibility of the features to train on, as most song datasets online do not release the raw audio (and rightly so, due to copyright) or any of its derived features.

A few checks were done before proceeding to training. The dataset features are chroma vectors generated from *NNLS-Chroma* VAMP plugin [2], while the expected input to *autochord.py* are raw audio files. There was a need to ensure that the plugin can be run from Python, and that the features can be closely reproduced from raw audio to refute poor model performance arising from training and inference feature mismatch. Fortunately, there exists a library for running VAMP plugins from Python.[5] To investigate the potential feature mismatch, five songs were arbitrarily selected from the dataset. Each of the songs' raw audio was acquired and processed on NNLS-Chroma to generate new chroma vectors. *Dynamic time warping* (DTW) [3] was then employed on the normalized vectors in round-robin fashion to check similarities between the songs. Table 1 shows the total alignment cost for each DTW operation. For comparison, each cost was also normalized by the length of the warping path.

Song ID	0018	0270	0637	0736	1289
0018	0.080	0.368	0.280	0.265	0.264
0270	0.366	0.042	0.334	0.283	0.342
0637	0.280	0.338	0.031	0.300	0.278
0736	0.267	0.280	0.300	0.048	0.281
1289	0.260	0.345	0.274	0.290	0.039

Table 1. Normalized DTW alignment cost for five Billboard songs. Row headers signify *new* chroma vectors for a song ID. Column header signify vectors from dataset.

Alignment cost for vectors in same song index is consistently low, 0.048 on average, while costs for different song indices are consistently much higher. This was deemed enough to show that feature mismatch will not significantly affect inference performance.

2.2 Chord Recognition Model

Initially, a multi-layer feedforward neural network was trained, which takes each 24-element NNLS-Chroma

[1] https://github.com/cjbayron/autochord

[2] https://www.music-ir.org/mirex/wiki/2021:Audio_Chord_Estimation

[3] https://www.tensorflow.org

[4] https://ddmal.music.mcgill.ca/research

[5] https://github.com/c4dm/vampy-host

vector as its input and predicts the chord among 25 classes: major/minor triads and no-chord. This simple architecture was only able to reach around 50% chord accuracy, after which no further tuning was explored.

To take advantage of the temporal relationships in the chroma features, a Bidirectional LSTM-CRF (Bi-LSTM-CRF) [4] was then used. The final trained model contains an LSTM layer with 128 units for *each* direction and takes chroma vector sequences of fixed length 128 as its input, pre-padded with zero-valued vectors as needed, and outputs for *each* vector in the sequence, a chord among the 25 classes previously mentioned. During training, a dropout rate of 0.1 was used for the LSTM layer, batch size of 64 for the inputs, and Adam [5] was used as optimizer with learning rate of 0.001. A randomly sampled set of 100 songs was used as test set, while the remaining 600+ songs was used for training. Chord accuracy was used for model selection, while accuracy excluding no-chord labels ("non-no-chord accuracy"), as well as *weighted chord symbol recall* (WCSR) on root notes and major/minor triads were also measured, as shown in Table 2. On a mid-range CPU, average inference speed is 0.11s for a 4-minute song. The Bi-LSTM-CRF model is uploaded to a public cloud service for usage in *autochord.py* library.

Chord accuracy	67.33
Non-no-chord accuracy	70.03
WCSR (root)	74.77
WCSR (majmin)	70.62

Table 2. Metrics on 100 songs from Billboard dataset.

3. PYTHON LIBRARY

3.1 Setup

autochord.py is available in PyPI [6] and is installed by running: `pip install autochord`. It has been tested to work well on Python 3.6, on a Linux (Ubuntu) machine.

On first import, the NNLS-Chroma plugin packaged along with *autochord.py* is setup and the chord recognition model is downloaded on the target machine.

3.2 Features

autochord.py provides a simple API for chord recognition: the *recognize* function which takes in as arguments the filename of the song WAV file and an optional LAB file where the predicted chord labels are stored in MIREX format as shown in following code:

```
import autochord
autochord.recognize('song.wav',lab_fn='est.lab')
```

Under the hood, *recognize* resamples the input audio to 44100 Hz, runs NNLS-Chroma to extract chroma features, feeds the chroma to the trained Bi-LSTM-CRF model, and returns the predictions in a list of tuples, each in the format: *(start time, end time, chord label)*. Based on tests on a mid-range CPU, the full prediction time is projected to be around 7 seconds for a 4-minute song.

4. CHORD VISUALIZATION APP

4.1 Setup

The *autochord.js* app is deployed online via GitHub Pages[7] and works as expected on recent versions of Chrome and Firefox.

4.2 Features

Any MIREX-format LAB file can be used to visualize chords in *autochord.js*. The app requires to load a song before a LAB file can be loaded. It displays the song in a "seekable" waveform, overlayed with color-coded chord labels. Optionally, a second LAB file can be loaded which adds another instance of a labelled waveform. This may be useful for visual comparison of different labels e.g., ground-truth vs. model predictions. Figure 1 shows a snippet of the app when two files are loaded.

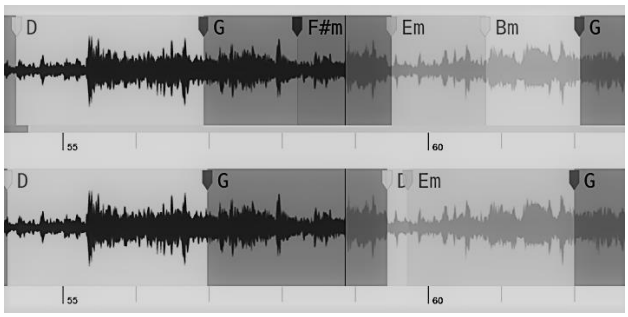

Figure 1. Waveform display of *autochord.js* app

5. FUTURE WORK

Integrating chord estimation and visualization in a single app will provide a more user-friendly experience, for which there is a need to port all Python processes into JavaScript. Primarily, the chord recognition model and NNLS-Chroma plugin must be considered. For the model, fixing the conversion issue[8] seems to be the low-hanging fruit, but one may also explore changing the architecture or even using tools other than TensorFlow. For the plugin, some open-source solutions[9] show promise.

For increasing the accuracy of chord recognition, alternative and supplementary techniques e.g., attention-based models, class weighting, and data augmentation may be explored. More significantly, getting access to raw audio of datasets and extracting more low-level features will likely lead to big improvements, as the NNLS-Chroma vectors alone may lack the more intricate audio features that describe the occurrence of each chord class.

6. REFERENCES

[1] J. Burgoyne, J. Wild, and I. Fujinaga, "An Expert Ground Truth Set for Audio Chord Recognition and

[6] https://pypi.org/project/autochord

[7] https://cjbayron.github.io/autochord

[8] https://github.com/tensorflow/tfjs/issues/5413

[9] https://github.com/piper-audio/piper-vamp-js

Music Analysis", in *Proc. of the 12th International Society for Music Information Retrieval Conference*, ed. A. Klapuri and C. Leider (Miami, FL, 2011), pp. 633–38

[2] M. Mauch and S. Dixon, "Approximate Note Transcription for the Improved Identification of Difficult Chords", in *Proc. Of the 11th International Society for Music Information Retrieval Conference*, ed. J. Downie and R. C. Veltkamp (Utrecht, the Netherlands, 2010), pp. 135–40

[3] M. Mueller, *Fundamentals of Music Processing — Audio, Analysis, Algorithms, Applications*. Springer Verlag, 2015

[4] Z. Huang, W. Xu, and K. Yu, "Bidirectional LSTM-CRF Models for Sequence Tagging", in *CoRR*, abs/1508.01991 (2015)

[5] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization", in *Proceedings of International Conference on Learning Representations*, (2015)