FFT and Machine Learning Application on Major Chord Recognition

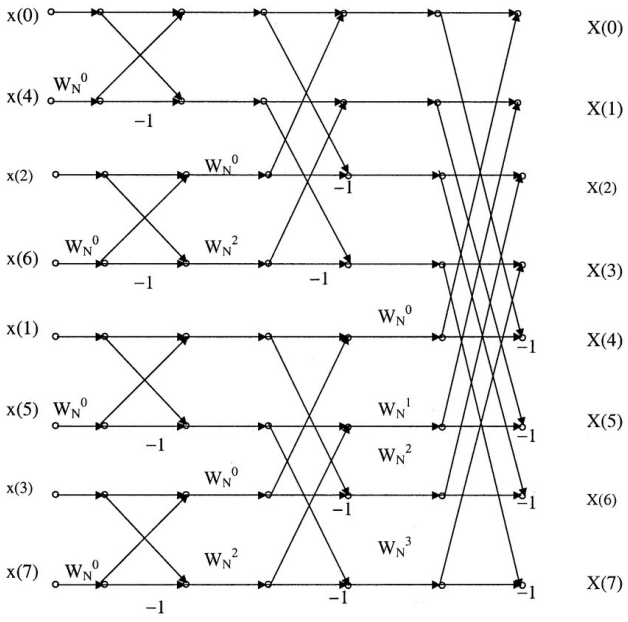
Nolan Monnier, Darien Ghali, and Sophie X. Liu
Engineering School, Oral Roberts University, Tulsa OK 74171 USA

Abstract — The purpose of this project was to use theory of Fast Fourier Transforms and a machine learning algorithm called Treebagger in order to process and recognize musical chords. Four musical chords were selected: C, D, G, and A. To be successful, the program should recognize chords in various octaves and musical inversions. Our goal is to use digital signal processing techniques such as sampling and frequency decomposition to preprocess audio files for input and train into the machine learning algorithms. For our application, we used MATLAB's machine learning tool Treebagger.

Keywords—FFT; machine learning; musical chord recognition; Treebagger (key words)

I. THEORY

Digital frequency analysis relies heavily on the Fast Fourier Transform (FFT). It is a specific form of the traditional Digital Fourier Transform (DFT), used to increase computational speed. It simplifies the signals overall DFT into a series multiplications and two point DFTs. It can be represent in the following butterfly signal graph.



$W_N^0 = 1$, $W_N^1 = (1-j)/\sqrt{2}$, $W_N^2 = -j$, $W_N^3 = -(1+j)/\sqrt{2}$

Fig. 1. A N = 8 FFT Signal Graph.

It 's important to note that, for situations its applicable, the result of the FFT is equal to the result of the DFT. Thus it follows the same rule for frequency resolution shown in equation 1.

$$Fs/N \quad (1)$$

Where Fs is sample frequency, N is number of data points used in the FFT. So if the goal of this project is to identify chords from a signals frequency, then some musical theory must covered to understand why the application is so difficult. A chord consists of three or more musical notes, played simultaneous. Because a chord is only defined by the notes that comprise it, are a near limitless array of keys and qualities that can make up a chord. The level of complexity is so vast, that it is nearly impossible for even most musicians to guess both a keys key (i.e C) and quality (Major) consistently. This complexity is further compounded by octaves and inversions. A chord can exist in any octave, or even between octaves. This means that no chord can be represent by a unique set of frequencies. Furthermore, the orientation of the three or more notes does not change a notes quality or key. The complexity is so great in fact, that simply frequency recognition software is insufficient to identify chords.

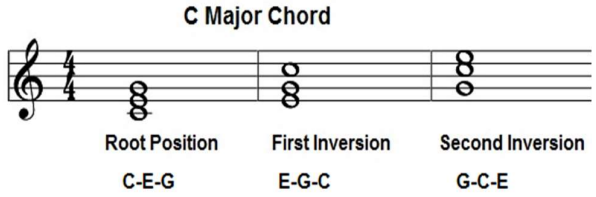


Fig. 2. C Major Chord with inversions

So in order to tackle the complexity of Chord recognition from audio samples, we will be using the Machine Learning techniques. Specifically, we used classification trees with Bootstrapping Aggregation, also known as bagging. Classification trees are decision trees that a computer can use to learn datasets. At the end of each branch of the tree, either a prediction or new decision is made. Normally however, a large classification tree has terrible predictive power due to overfitting. To solve this, bagging takes multiple smaller decision trees and takes the aggregate result.

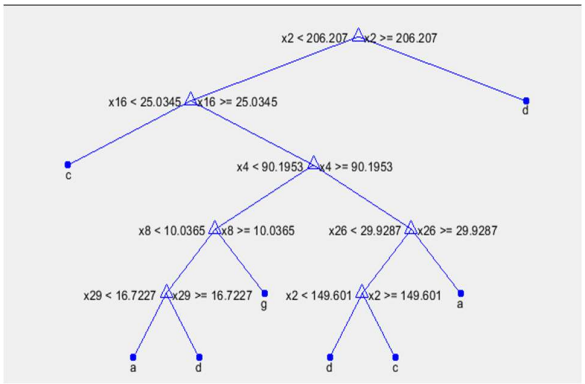


Fig. 3 A Sample MATLAB Classification Tree

These aggregate trees comprise a model, which can be used to predict new data. To test the predictive power of a new model, we cross validate. Cross validation means giving a model data it has not seen before, letting it predict the answer, and the comparing it with the correct result. In classification applications, this is typically done through dividing the number correct by the number predicted.

2. METHODOLOGY

The plan for our program was simple. First, we'd obtain a set of chord samples. Then, MATLAB would read in these files and store them. MATLAB would process this data for key frequency information, and use this frequency as data in a machine learning algorithm.

A digital music interface called MIDI was used to render the musical chords in Garage Band's recording studio. We recorded 36 audio files, containing the chord C, D, G, and A. We limited our recording to the middle three octaves, and the first three inversions (0,1,2). iTunes was then used to convert the files to wav, an uncompressed format acceptable to MATLAB. Next we determined a table of relevant musical note frequencies to encompass all possible musical notes played in our set of samples. The useful range of note spans from G2 to C5 (98 Hz to 523.3 Hz), so our table included every note in between. This table would prove useful for preprocessing our signal data in MATLAB.

Our MATLAB program takes the samples and stores them internally for quick reference. It will also store input information like the length of each sample N, the chord of each sample (i.e. C,D,G,A), and the sampling frequency Fs. The each recording sample is stored in a row of a matrix, corresponding with a chord identifier stored in a matrix. MATLAB then quickly extracts frequency data from each row using the FFT, and subsequently stores them in a new dataset. The amount of data retrieved by the FFT is excessive (Fig. 4), so we used our table of useful frequencies to preprocess the data (Table 1). The table is hardcoded into our process function, so MATLAB can index and extract the desired frequencies from our FFT results. This feature selection was done through dividing the frequency table by the frequency resolution (Equation 1) and then indexing the FFT data. This achieves as set condensed and useful data samples seen in Figure E. The output data is formatted such that each row is a data sample, and each column is a note's magnitude.

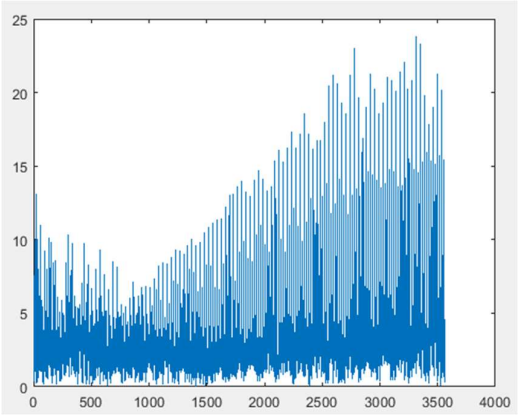


Fig.4. Output of FFT

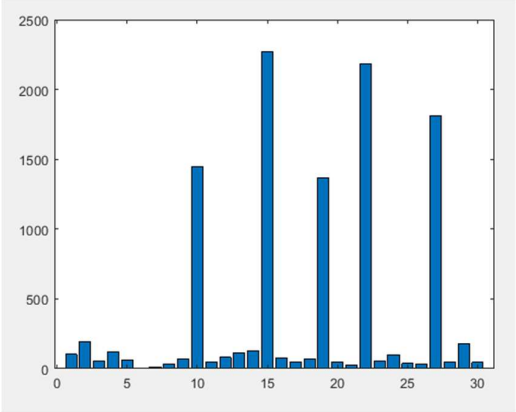


Fig. 5. Output of Process Function

In order to prepare the data for machine learning, we separated the whole set into train and test sets. In keeping with standard practices, 75% of the audio files were chosen as a training set. The frequency matrix as well as the corresponding chord vector are both separated in this manner. The train set matrix and vector are sent as arguments to the Treebagger function. The Treebagger function returns an machine learning model. The test set of the audio files are used for Cross Validation. We test to see how good our model's output is by giving it unfamiliar chord. So, we pass the model and our test set of the frequency matrix to Treebagger predict function. Finally, these predicted chords are compared with the test set of chords in a simple loop to count how many of the predictions were correct.

3. TESTING PROCEDURES

To review, the MATLAB code is divided into various sections. It includes inputting the chords, processing the chords, separating the chords into training and test sets, using the Treebagger to model the data, and cross-validating.

We initially loaded files one at a time. On each input, we'd print out the sampling rate to ensure the frequency was consistent. Eventually, we needed to load all the audio files at once. To facilitate this, we stored the .wav files and their chord names into respective columns in an excel file. The excel file was then used to identify to MATLAB and load each the entire set.

Though the focus of our project was on preprocessing, we also still had to optimize the performance of Treebagger in order to prove concept. For bagged classification forest, the number of trees often significantly impacts the performance of the learning model. So, part of the testing to improve predictive accuracy involved plotting an Error-vs-Tree Number Graph to help improve predictive accuracy:

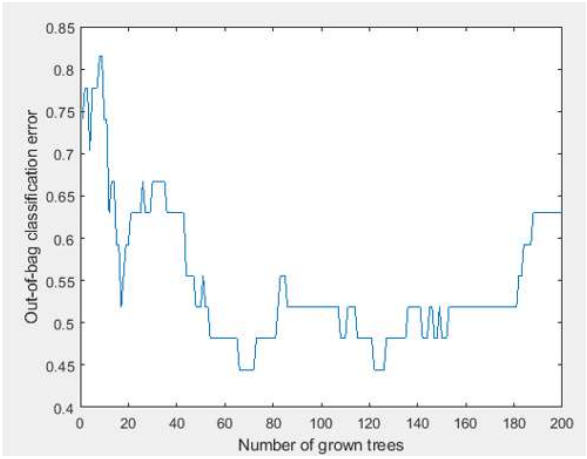


Fig.6. Out of Bag Error Predictions

4. EXPERIMENT RESULTS AND GRAPHS

We initially had low predictive accuracy in recognizing the musical chords. Percentages were typically around 20-30%. In other words, our algorithm was simply guessing. At one point, attempts to tune our Treebagger algorithm included additional parameters trying parameters such as 'MinLeafSize'. Eventually, we determined that the best performance occurred for a forest size 30-40 trees, and determined that no other parameters provided a sizeable improvement. Additional improvements were made concerning our musical note frequencies used for indexing and grabbing useful information from our samples. Originally we used math to determine the frequencies to process, going from 98 to 523.3Hz in steps of 4Hz (G2 to C5). As this proved problematic, we exchanged this for an exact hard coded vector of the musical note frequencies. In addition, we noticed an error in our attempt to take the magnitude of the FFT samples. Initially, the MATLAB code used the function real() for this task, but it became clear that the correct function was abs(). After these changes, the prediction results for the musical chord recognition increased to 67-89%, depending on computer seed.

5. CONCLUSIONS

Overall, our musical chord recognition using machine learning proved successful, as the goal was to score predictive accuracy of 75% or greater, as well as show that this type of chord identification could be done with the proper preprocessing. To this end, we were extremely successful. Increasing the number of recorded samples, further effort to fine tune the Treebagger algorithm, or further efforts to finetune feature selection.. In addition, bagged classification trees are not most reliable machine learning tool when compared with neural networks or Randomforest algorithms. Perhaps python could be used to utilize these machine learning tools. Finally, our task could be broadened to include more chords or even include identification of the musical instrument.

TABLE 1. TABLE OF RELEVANT NOTES AND THEIR FREQUENCIES.

Note	Freq (Hz)
G2	98.00
G#2	103.93
A2	110.00
A#2	116.54
B3	123.47
C3	130.81
C#3	138.59
D3	146.83
D#3	155.56
E3	164.81
F3	174.61
F#3	185.00
G3	196.00
G#3	207.65

A3	220.00
A#3	233.08
B3	246.94
C4	261.63
C#4	277.18
D4	293.66
D#4	311.13
E4	329.63
F4	348.23
F#4	369.99
G4	392.00
G#4	415.30
A4	440.00
A#4	466.16
B4	493.88
C5	523.25

6. REFERENCES

[1] Joseph Hoffbeck, "Enhance your DSP course with these interesting projects". ASEE annual conference & exposition, 2012.

.