Musical Pitch Identification

Manoj Deshpande

Introduction

Musical pitch identification is the fundamental problem that serves as a building block for many music applications such as music transcription, music accompaniment, query-by-humming, instrument identification, source separation, etc. There is a need for robust and accurate techniques for pitch identification.

Pitch detection is an important and active research topic with deep history. The research community has pursued several techniques for fundamental frequency estimation. In spite of success of domain specific estimation techniques, no single general purpose technique exist that works well for music particularly in presence of multiple fundamental frequencies such as in polyphonic input, unpitched audio sources such as percussion instruments, and noise.

For CS229 class project, I employed the machine learning techniques learned in the class for musical pitch determination. My interest is guitar, and I aimed to apply what I learned in the class to guitar. In my approach, I leveraged general machine learning techniques to build pitch classifier without relying on any explicit domain knowledge based on the musical structure. In this report, I describe my use of Naïve Bayes, Logistic Regression, and Support Vector Machine variants. I also experimented with additional techniques such as multilayer perceptron, Radial Basis Function network using K-means, etc.; however, findings from these additional techniques are not described in the report due to space limitation.

Training Set Preparation

As a budding guitarist, I fancy use of machine learning techniques to develop a comprehensive guitar learning system. Such a system can listen to and understand a player playing a scale or practicing a score. This understanding can be used to track student's progress, lend advice, and accompany a user.

The guitar pitch detection is the first building block of such a system, and the guitar pitch forms the target variable or class variable of the machine learning system.

Target Variable

The musical pitch consists of mix of fundamental frequency and its harmonics. The fundamental frequency f Hz relates to musical pitch p by equation $p = 69 + 12\log_2(f/440)$. The resulting pitch values create a linear space with a semitone of size 1 and an octave of size 12. The range from $p = 0$ to $p = 127$forms the MIDI tuning standard. Pitch detection to this granularity is a great capability; however, I discovered that many machine learning algorithms face performance challenge in large 128-ways classification. To mitigate this challenge, I limited my classification to pitch class space rather than the pitch space. In contrast to 128 element pitch space, the pitch class space is a 12 element circular quotient space that groups pitch octaves together forming the following 12-chromas:

C	C#/Db	D	D#/Eb	E	F	F#/Gb	G	G#/Ab	A	A#/Bb	B
0	1	2	3	4	5	6	7	8	9	10	11

Pitch identification within a chroma is not a severe limitation for guitar learning system as the student is aware of his/her position of playing on the guitar fret board and will accordingly interpret any feedback from the system.

Training Data

For the end goal of building a guitar learning system, I selected 20 guitar instrumental music tracks with medium difficulty such as those in the guitar training methods textbooks. These tracks include a lead guitar melody accompanied by guitar chords for popular tunes such as Jingle Bells, Yankee Doodle, the Star Spangled Banner, etc. I sampled the tracks at 8 KHz to generate monophonic WAV encoded files.

Following the advice from the class in the learning theory lecture, for supervised learning algorithms the number of training examples needed to learn well is usually roughly linear in the number of parameters. Based on this advice and literature survey, I aimed for training set with ~40,000 training examples.

I used WaveSurfer, an open source tool, to estimate fundamental frequency for each 10 millisecond frame in 50 Hz to 2000 Hz range. The rational for selection of these parameters is explained in the Feature Variables section below. Subset of WaveSurfer output was manually checked against the musical scores of the selected songs for correctness. The erroneous samples corresponding to the unidentified pitch were eliminated from the training set. This set up resulted in ~40,000 frames with chroma labels.

The distribution of the training data is shown below in Figure. The beginning and intermediate guitar songs are often played either in the key of C major or A minor and avoid the use of accidentals. However, a guitar student in his practice may often make a mistake and play accidentals. The student may also practice music scales in keys other than C major or A minor. For this reason, it was important to generate training data with enough representation of accidentals. To remedy, I implemented a phase vocoder in Matlab that shifted the pitch of each 10 millisecond frame over $\pm 1, 2, 3, 4, 5, 6$ semitones. The phase vocoder implementation ensured that pitch shifting did not introduce any discontinuities and artifacts in the resampled song. The corresponding labels were also shifted to generate training data appropriately. The distribution of the second training data set with ~52,000 samples is also shown below in Figure.

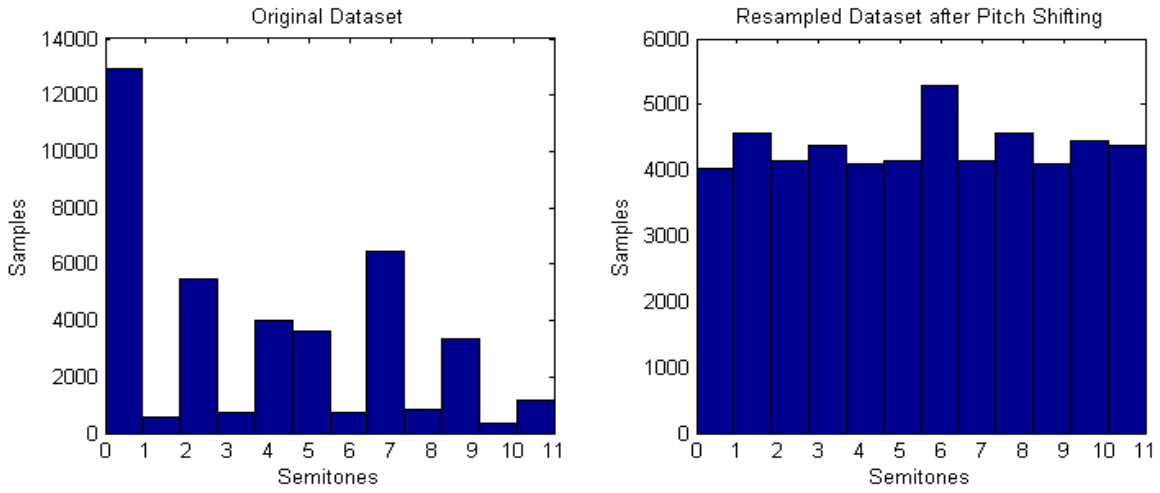

Figure 1 Training Data Distribution

As an alternative to resampling, the above problem can be solved by designing cost matrix with appropriate misclassification cost.

Feature Variables

I reviewed several time domain and frequency domain audio signal processing techniques for selection of appropriate feature variables for the pitch identification problem. The time domain techniques included zero crossing rate, autocorrelation, and YIN metrics where as the frequency domain techniques included the short

term Fourier Transform (STFT), spectrogram, and cepstrum metrics as feature variables. After some experimentation and literature research, I focused on the use of spectrogram metrics as feature variable.

As explained earlier, I sampled the guitar music tracks at 8 KHz in the monophonic WAV format, and I resampled to shift pitch to cover the pitch class scale uniformly. With these WAV data set as input, I used Matlab to calculate a 1024 point spectrogram with 1024 sample duration sliding Hanning window and 80 point overlap. The 80 point overlap at 8 KHz sampling frequency corresponds to 10 milliseconds. A guitar student may not exceed fast tempo of 120 beats per minute. At that tempo with 4/4 time signature, $1/16^{th}$ notes are sustained for 500 milliseconds. 125 millisecond windows with 10 millisecond overlap provide ample opportunity to capture guitar output with sufficient fidelity.

The guitar pitch is mainly determined by fundamental frequency, and the guitar produces fundamental frequencies in the range from 80Hz to 1500Hz. As such, I limit the spectrogram content to 2 KHz corresponding to first 256 bins. In addition, to account for timbres variation across guitars and additional context, I normalized the content of the first 256 frequency bins by subtracting the mean and dividing by the standard deviation. The normalized magnitudes of the first 256 frequency bins are the 256 feature variables.

Algorithms Selection and Experiments

Following the determination of feature variables and generation of training data, I researched possible supervised and unsupervised learning algorithms for musical pitch detection and shortlisted Naïve Bayes, Logistic regression, Multilayer Perceptron, and Support Vector Machine techniques. After some initial implementation efforts, I discovered Weka machine learning toolkit and used it for experimentation with the selected machine learning techniques. All the experimentation below used 3-fold cross validation for performance analysis.

Naïve Bayes

I first selected Naïve Bayes algorithm due to its simplicity, and its fast execution helped me run many quick experiments and develop better understanding of the data. I plot the performance curves below for the original dataset and the resampled data set. The resampled dataset exhibits marginally better learning than the original dataset while plateauing at ~64% accuracy. Resampled dataset represents each chroma more or less uniformly. However, the original dataset is by no means uniform. To understand individual chroma classification better, I plot the RoC and Precall characteristics for resampled dataset and chroma-0 (C) and chroma-1(C#) of the original dataset.

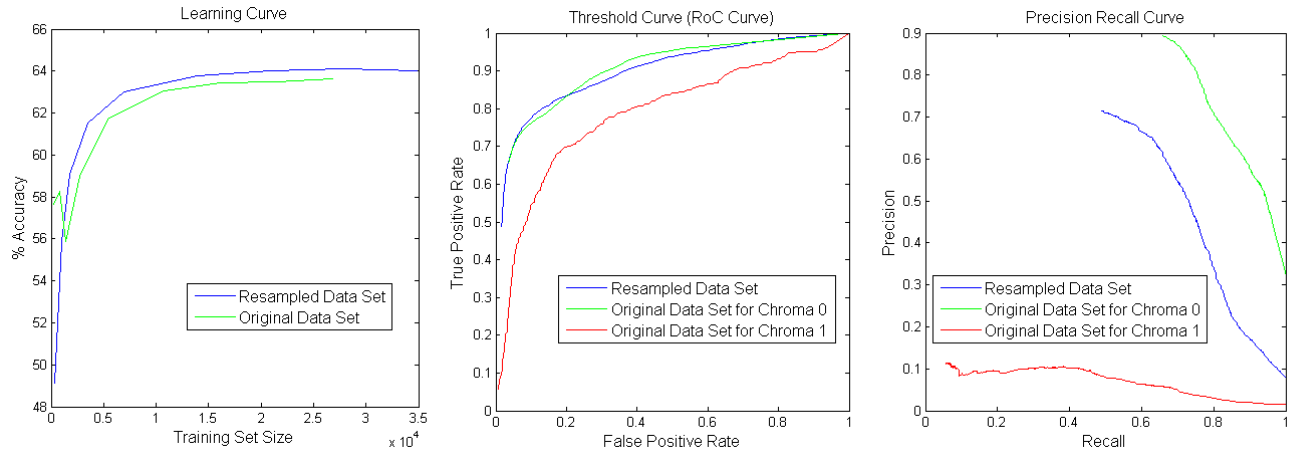

Figure 2 Naive Bayes Performance

With its larger representation in the original dataset, chroma-0 exhibits better RoC and Precall characteristics compared to those of the resampled dataset. However, the performance characteristics for less represented chroma-1 in the original dataset are significantly impaired compared to that of the resampled dataset. Similar observations were made for other learning algorithms as well. For reasons noted earlier, a guitar learning system may require uniform performance across all chroma, and learning from the resampled dataset is preferred to learning from the original dataset. As such, the remaining learning algorithms are compared using the resampled dataset baseline.

Naïve Bayes assumes that the feature variables describing the 256 frequency bin contents of the spectrogram are conditionally independent of each other given the chroma. This independent assumption is clearly not true! Additionally, the implementation of Naïve Bayes assumes that conditional distribution of the frequency bin content is normally distributed and estimates the distribution parameters using the training data. Both these assumptions likely limit the performance of the Naïve Bayes to at most 64% accuracy.

Logistic Regression

The logistic regression algorithm in Weka is a multinomial logistic regression model with a ridge estimator that stabilizes degenerate cases and guards against overfitting the parameter matrix $\mathrm{B}_{256\mathrm{x}11}$. The probability that the 256 length feature vector X_i corresponds to chroma $c = 0$ to 10 is given by the probability $P(c|X_i) = e^{X_i^T \beta_c} / (1 + \sum_{j=1}^{11} e^{X_i^T \beta_j})$. The corresponding value for c=11 is obtained by normalization. The parameter matrix $\mathrm{B}_{256\mathrm{x}11}$ is estimated by maximizing the multinomial log-likelihood function using Quasi-Newton method. Logistic regression performance is compared with Naïve Bayes performance below.

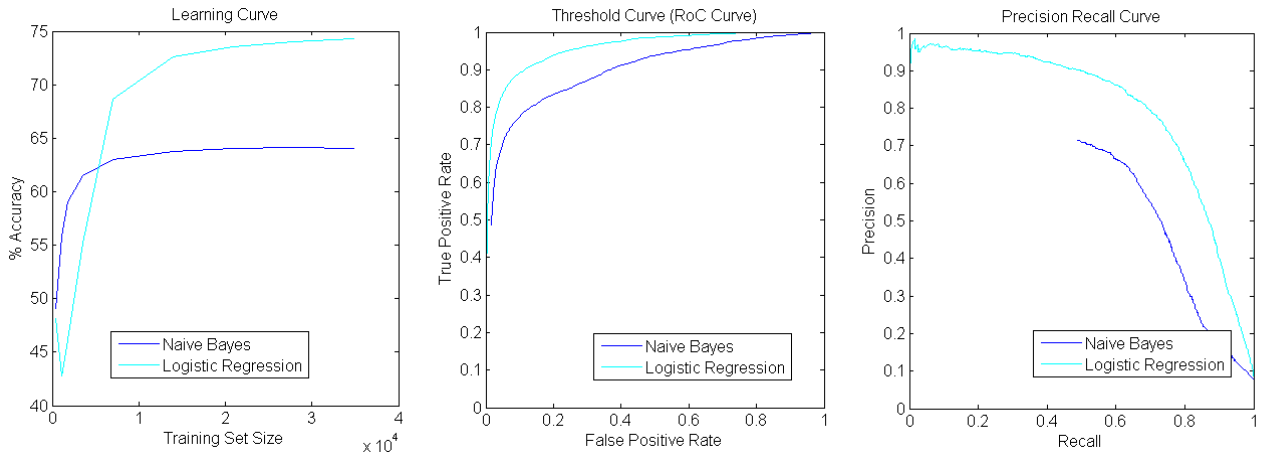

Figure 3 Logistic Regression and Naive Bayes Performance

Without any dependence and normality assumptions, Logistic Regression achieves the accuracy of ~74%, and its RoC and Precall performance well dominates that of Naïve Bayes.

Support Vector Machines

Following the use of Naïve Bayes and Logistic Regression, I used the Support Vector Machine with Sequential Minimal Optimization implementation in Weka. I selected the linear kernel that takes majority vote from the output of all pairs $C(C - 1)/2$ discriminant functions.

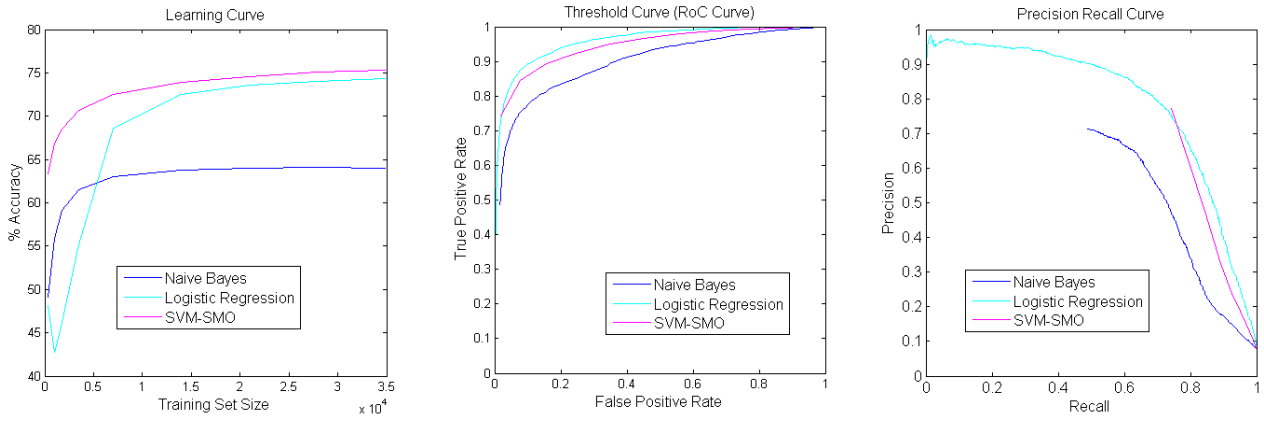

Figure 4 SVM, Logistic Regression, and Naive Bayes Performance

SVM not only exhibits higher accuracy than both logistic regression and Naïve Bayes but also achieves this accuracy on a smaller training set! However, for a full training set of 52K samples, logistic regression has slightly better RoC and Precall characteristics. SVM determines the maximal margin hyperplane separating the training set. Apparently, 10-15K training examples are enough for the SVM to discover the support vectors defining the near optimal separating hyperplane. The additional training examples do not affect the hyperplane orientation significantly. On the other hand, logistic regression determines the parameters by maximizing log likelihood, and additional training set examples help it improve the probability estimates further. This better probability estimate also helps logistic regression to improve the quality of its decision as depicted by RoC and Precall characteristics albeit at significantly higher training set size. The ability of SVM to train comparably at much smaller training set is a significant advantage due to slower execution of Weka toolkit implemented in Java.[1]

Epilogue

CS229 project offered me an excellent opportunity to understand machine learning algorithms by experimenting with them. In addition, I gained valuable knowledge regarding music, signal processing, and tools of the trade. I aim to continue tinkering with use of machine learning for developing guitar learning application and I appreciate any feedback you may have on improving my understanding.

Thank you.

References:

1. G. Poliner, D. Ellis, A. Ehmann, E. Gómez, S. Streich, B. Ong (2007), “Melody Transcription from Music Audio: Approaches and Evaluation”, IEEE Tr. Audio, Speech, Lang. Proc., vol. 14 no. 4, pp. 1247-1256, May 2007,.
2. D. Ellis and G. Poliner (2006), “Classification-Based Melody Transcription”, Machine Learning, special issue on Machine Learning In and For Music, vol. 65, no. 2-3, pp. 439-456, Dec 2006.
3. Andrew Ng, “Advice for applying machine learning”
4. Weka - Data Mining Java Software
5. Additional references not listed due to space limitation and are available on request

[1] In spite of several attempts to speed up Weka by parallelizing and compiling Weka, I was not able to implement SVM with Radial Base Function kernel with full training set due to execution and memory size constraints,. However, with 20K sized training set, RBF SVM did not show significant improvement over linear SVM potentially indicating feature set saturation.