Robo-~~hummer~~ re-installer

Zachariah J. DeCook, Micah Y. Ng

Vision and Overview

This senior design project aimed to write a program that searches for songs on Hymnary.org by a user's audio rendition of the melody (especially via humming). This project hoped to be an alternative to the existing melody search.

We planned to solve this by designing a system to recognize tunes based on individual rhythms and notes. In order to do so, we attempted to train a Neural Net to analyze audio signals for these rhythms/notes. We came up with a search algorithm to match the analyzed audio to actual song definitions. This software was cleaned up so that it is a somewhat usable product.

Research Review

Searching using the on-screen keyboard on hymnary.org is quite cumbersome. Various companies already produce software capable of recognizing songs from their actual recordings, however they necessitate a high degree of similarity; the audio fragment must essentially be from the same recording for their audio fingerprinting techniques to work.

We researched audio analysis libraries, and found Marsyas to be a good first step for getting quantitative feature data from audio files.

We decided to use Tensorflow to train our neural network.

Design

We designed a pipeline to bring audio from a web interface through our machine learning, and into the k-nearest neighbors search.

We collected audio samples to train the neural network.

In testing this, we discovered it to be unreliable, so we created a keyboard interface to make sure the searching works properly.

Our search algorithm takes in the music XML corpus and divides it up into a shifting window of the context length (we selected 4 notes). Each segment is converted to a feature vector comprised of the pitches and lengths of each note relative to the first note in the segment. The relative format allows us to ignore differences in key and tempo.

Example

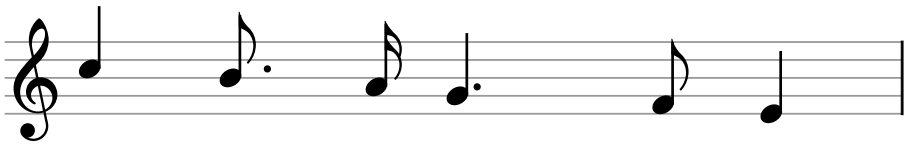

Figure 1: Full phrase.

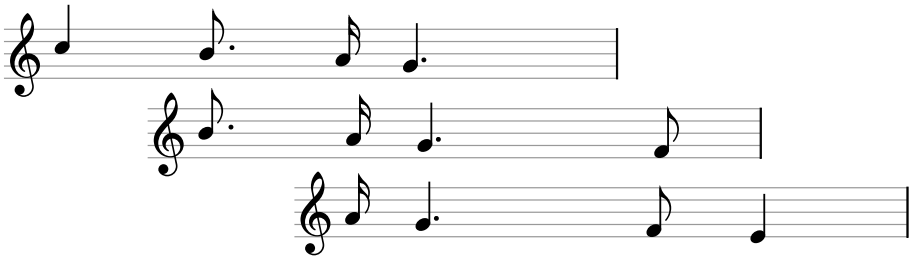

Figure 2: Segments.

Table 1: Relative values of the first segment.

MIDI	Length
72	500
-1 (71)	**0.75** (375)
-3 (69)	**0.25** (125)
-5 (67)	**1.5** (750)

Resulting feature vector: `< -1, 0.75, -3, 0.25, -5, 1.5 >`

The feature vectors are stored in an index for k-nearest neighbor queries. We selected the the HNSW approximate k-NN algorithm for its superior performance and accuracy.

At search time, the input music phrase is similarly divided into feature vectors. These vectors are run through k-NN. The resulting points are grouped by which song they are from, with the frequency becoming the match score.

Results

Our initial neural networks for predicting notes from audio data supported excessive learning capabilities. This resulted in the networks learning the exact training files. When fed new data, prediction performance was abysmal, often ranging +/- 1 octave.

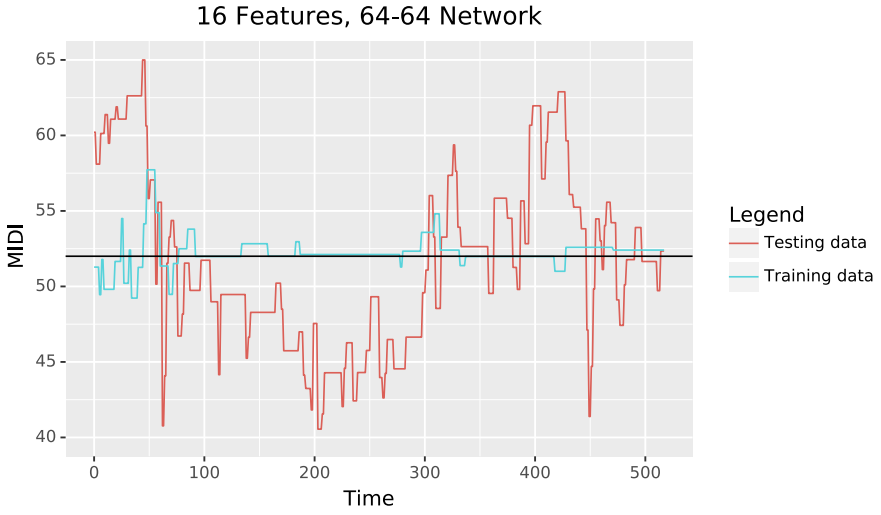

Figure 3: Over-training.

To improve the model, we reduced the size of the neural network while increasing the number of input features. The prediction range narrowed to approximately +/- 3 tones. A significant improvement, but insufficient accuracy to make search via humming practical.

Our alternate method of predicting notes was based on the YIN algorithm. Although this algorithm is commonly used in tuners, we were unable to get useful results from it for hummed input. (Input from musical instruments tended to be highly accurate, but was outside of the scope of our project.) Pitch detection algorithms may warrant further investigation.

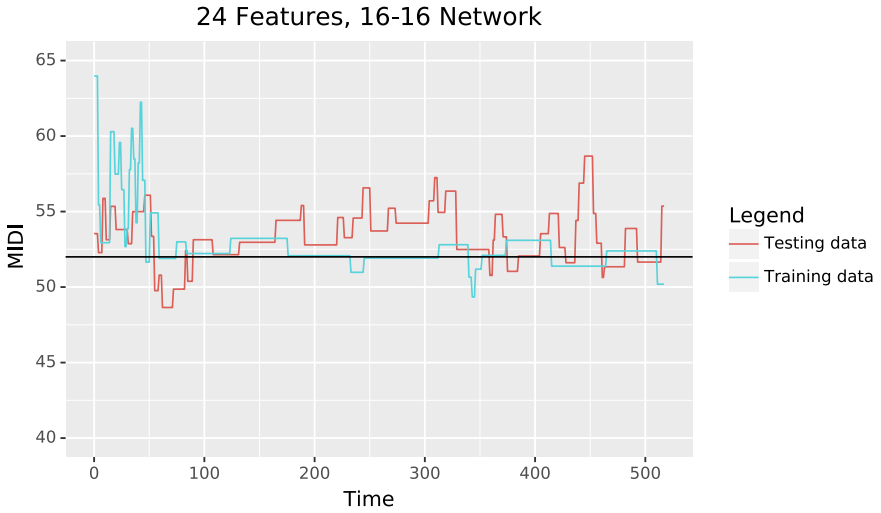

Figure 4: Smaller network.

The second component of our system, search, yielded more successful results. Even with mild inaccuracies in pitch and timing, our nearest-neighbor based scoring algorithm generally ranked the correct tune as the top, or close to the top result. Incorporating note length (in addition to just pitch) enabled our relatively short context length of four notes to remain a useful distinguishing factor.

However, the search was designed with audio input in mind. While humming, getting note length correct is easier than with our fallback keyboard input. As such, we may be able to provide more stable results by placing a lesser weight on the note length components of the score. Also, our search algorithm has increased difficulty in handling insertion and deletion mistakes. It requires sufficient input to take search contexts from around, but not including, this type of error. To make search more tolerant, we could include in both our index and search contexts, all combinations of single-deletion mistakes. This requires evaluation to prevent accurate searches from degrading.

Conclusions

Neither the YIN algorithm nor the neural network based approaches worked well for vocal input, so, we removed the humming-based search page in favor of the keyboard interface for getting notes. Searches come back with great results. Since the search index data came from Hymnary.org, each search will return a link to the relevant hymn instance on Hymnary.org.

Future Work

The neural net could be trained more to recognize humming.

We could experiment with LSTM - Long Short-Term Memory neural networks.

The new search with the keyboard interface could possibly be used as a replacement for the old Hymnary melodic search feature. More improvements to the interface - such as editing the sample, may be wanted. Ability to import music from MIDI files is a critical feature for this to be used as a replacement for the former melody search.

The audio analyzer is implemented as a REST endpoint. However, the search script must reinitialize indicies on each run, adding inefficiency. Converting to also be a micro server (ideally via AIOHTTP) could improve performance.

Further exploration of pitch detection algorithms could provide a viable replacement for neural network based note recognition. Our experiments with YIN (and related techniques) were minimal, and so could see improvements.

Acknowledgements

We acknowledge Dr. W. Harry Plantinga, for provision of MusicXML files for the search index, as well as a script to assist in importing data from these.

Will Groenendyk, for helping to set up LDAP logins on the system a few of the times that we re-installed the system.

Nathanael Dick, for starting work on the project, and giving motivation and encouragement for the work being done.

References

For our project, we used Marsyas and Tensorflow for the audio analysis section.

For the search backend, we used NMSLib, the Non-Metric Space Library, as well as a few scripts to parse musicXML into a simpler format for searching.

The web framework used for the site was laravel.

For the keyboard frontend, we used MIDI.js for audio feedback and VexFlow for visual musical notation feedback.

Appendices

- robo-hummer on Github