Computer-executed Genre Classification of Music
Computer Systems Lab 2009-2010

Alex Stabile

May 31, 2010

Abstract

The goal of this project is to write code that can accurately group given pieces of music into appropriate genres. Genres are often generalities that may not perfectly fit a given piece of music, but by analyzing different musical qualities we may determine what genre best describes it. Computer code that can accomplish this task could have applications in sorting large libraries of music or suggesting music to individuals based on musical qualities, rather than comparing with common likes and dislikes. For ease of analyzing notes, midi format music files are used as input for the program. Python was used to write classes that can read and store the information contained in midi files, such as note value, duration, and tempo. These data are organized by grouping notes into their appropriate beats. Organizing them in such a manner allows for harmonic analysis, which provides insight into how a piece is written and what it sounds like. We theorize that this information would be enough to distinguish among basic genres of music. For this project, music by Mozart was compared to music by Rachmaninoff, two great composers with very different compositional styles. A neural network is trained by analyzing harmonic data from a set of music by each composer, learning how to distinguish between the two even with music it has not seen before. For future research, analyses of other low-level musical qualities may be implemented to provide a fuller picture of a given piece.

Keywords: music, genre, naive, bayes

1 Introduction and Background

Current research often uses statistical models to determine how a given piece of music should be categorized. One approach has been to use Inter-Genre Similarity modeling [2]. This project attempted to categorize music by analyzing the timbral textures of a short sample of music. These textures differ due to differences in instrumentation and rhythm. Gaussian Mixture Models were used to create the statistical model, and IGS to

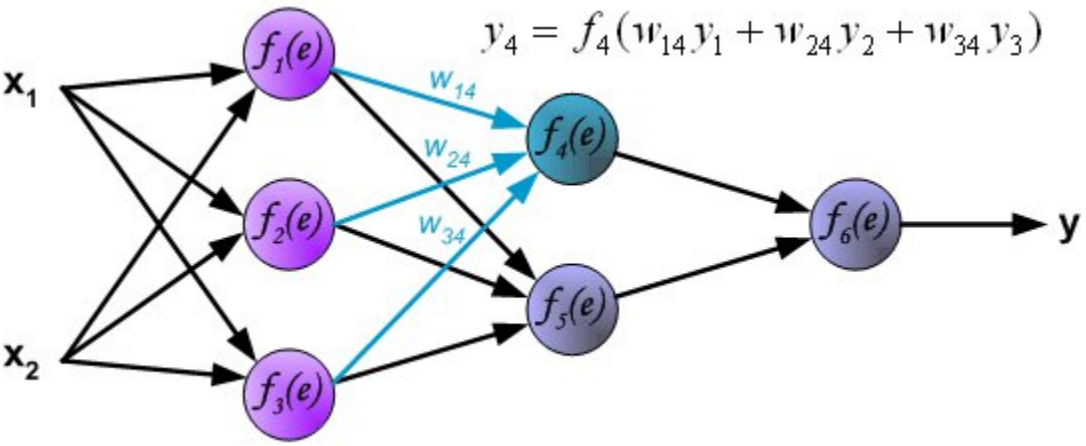
cluster similar groups together. Their algorithms grouped a 0.5 to 30.0 second sample of music into one of nine genres. Longer samples yielded better results, up to 64 percent correct. This somewhat low success rate is attributed to the difficulty of machine analysis of sound samples. Music has also been represented through a model of rhythmic complexity [6]. With this method, rhythm was represented in a tree structure and its overall complexity was determined. Results were moderately successful, but the method of organizin notes could be useful in other attempts at classifying music. Another approach attempted an automatic method of classification that is completely general [3]. This was done by looking for mathematical similarity, rather than features specific to music. Midi files were used for musical analysis, and very successful results were shown when grouping music into rock, jazz, or classical genres. Results were moderately successful when attempting to group pieces by composer, but got worse as sample size increased. Interestingly, the algorithm could even cluster like file types together (sorting out java class files, gene sequences from different species, and widely different styles of music). One especially successful experiment was "On musical stylometry: a pattern recognition approach" [1]. Different musical aspects of pieces were analyzed, and a statistical model was created to group new pieces into their appropriate period. By analyzing more musical characteristics, their model became more fine-tuned. These characteristics included harmonies, dissonance, note entropy, and types of intervals. No method has yielded or should be expected to yield perfectly successful resultseven many people cannot successfully place music into its correct genre.

2 Project Design

The first step of this project was to write code for reading and organizing the information stored in a midi file. A midi file is a sequence of commands preceded by how long to wait until they are executed (the delta-time value). For the purposes of this project, only the note-on and note-off commands and their delta-time values are needed. This allows code to determine which notes are playing at the same time and which beat they are a part of. This information is organized by storing it in a Beat class. Each instance of a Beat knows which notes sound on its downbeat, and which notes sound off the beat. It also contains which beat it represents from the piece (beat 1, beat 2, etc.), and can be sorted according to this number. To begin obtaining the necessary data, code iterates through a list of Beats and passes the notes found within each one to a chord identification method. This method returns what chord most accurately represents that combination of notes, roughly emulating how harmonic analysis is performed by a human, with some exceptions. In reality, harmony may change more than once within a beat. Also, notes that are part of a melody may be classified as non-chord tones, and computer code has no way of picking these out. However, groups of notes that do not form a chord can

still be processed: the code simply tries different combinations of different amounts of notes, looking for the most probable solution. The chord identification code accurately returns the key, quality, and inversion of a given group of notes (e.g., B Flat Dominant Seventh, Third Inversion). This information is then written to an output file. When the entire piece of music has been read, the output file will contain every chord found on every beat of that musical sample. An example line of this file might read: Beat 12: C Major, first inversion. Once these output files are finished, the actual data to be analyzed can be obtained. Though the inversion and quality of every chord is determined, this information was not used in the final version of the experiment. Instead, code reads the output file to count how many chords are in each relationship to the key of the piece. For example, a piece in C Major would likely contain many C, G, and F chords (the root, fifth, and fourth, respectively). The code treats a G chord in C Major the same as a D chord in G Major because both chords represent the fifth of each musical scale, and therefore perform the same role in their respective compositions. When every chord has been counted, each count is divided by the number of the beats in the piece, giving a ratio between 0 and 1 that represents how frequently that chord was found in the music. Machine learning methods have been shown to be useful in analyzing music due to their flexibility, so a neural network was constructed to analyze the data. Neural networks contain different layers of "perceptrons:" nodes that contain a value and that are associated with a weight (see figure 2.1). The input layer of the network is at the left side, and consists only of the different inputs the network will receive. In this experiment, these inputs are the ratios of different types of chords found in a piece. Because there are 12 such ratios, the neural network in this experiment has an input level containing 12 perceptrons. The values of the input layer are then "fed" through the network, assigning values to all the other nodes (one step of this process is illustrated in figure 2.1). The output layer consists of just one node, whose final value will represent what the network has determined to be the style of the music.

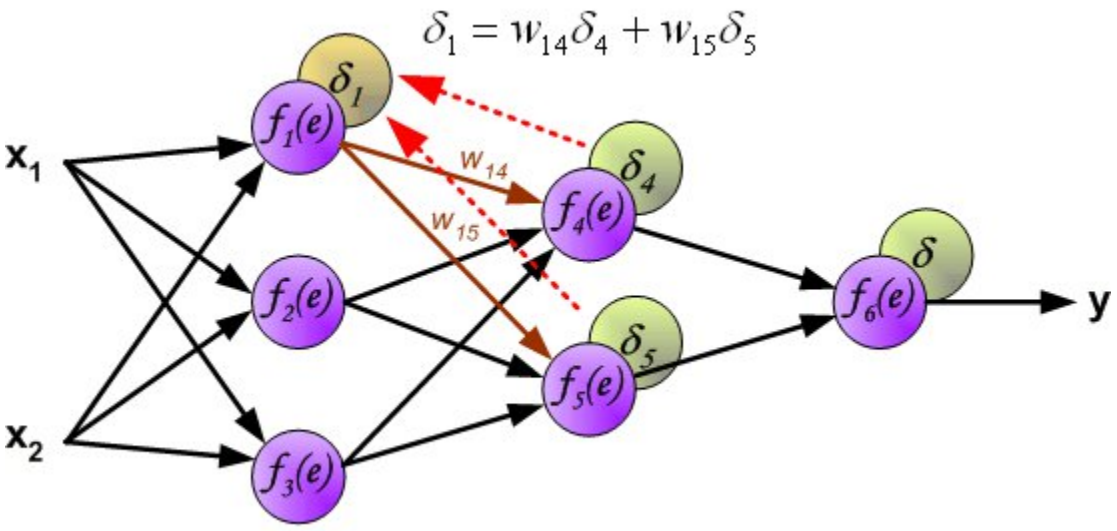
2.1 A Simple Neural Network



Source: http://galaxy.agh.edu.pl/ vlsi/AI /backp_t_en/backprop.html

When the network is created, weights are generated randomly for all the nodes. The network is then "trained" by giving it a data set containing inputs and their target outputs. In this experiment, a piece by Mozart had a target output of 0 while a piece by Rachmaninoff had a target output of 1. When the network produces a result, its weights are adjusted to lower its error. To determine how the weights should be adjusted, the learning algorithm known as back-propagation was used. This is a fast learning algorithm, an important quality due to the number of inputs the neural network has to handle. Back-propagation starts by calculating the error of the output node: the difference between the actual output and the target output. This is error is then propagated backwards through the network to each perceptron (as illustrated in figure 2.2). Weights are then adjusted based on the error assigned to each perceptron, lowering the total error of the network. After thousands of iterations, this process should yield a neural network that closely fits the training data. Theoretically, analyzing enough data will yield a network consistent not just with the training data, but also any inputs that may be thrown at it. The success of this project will be evaluated by how accurately the network produces output for music it did not see in training: compositions by Mozart should yield an output close to 0, and compositions by Rachmaninoff should yield an output close to 1.

2.2 Propagation of error



Source: http://galaxy.agh.edu.pl/ vlsi/AI /backp_t_en/backprop.html

3 Results

In this experiment, 28 compositions were analyzed: 14 by Mozart and 14 by Rachmaninoff. Each set of pieces was then split in two, with one half forming the training data and the other half forming the testing data. The network was trained and then tested a number of times, with varying amounts of learning iterations. It seemed that after 10,000 iterations, results showed roughly the same degree of accuracy. However, all results were surprisingly accurate: the average output from the testing data always rounded to the correct number, and only differed from the target output by +/- 0.2. Pieces by Mozart should have an output close to 0, while pieces by Rachmaninoff should have an output close to 1. The exact results are in figures 3.1-3.4, where each row is the output for a given piece of music by the listed composer.

3.1 4,000 Iterations:

Mozart	Rachmaninoff
0.155536572	0.731279433
0.041554391	0.749991791
0.412595589	0.992937952
0.006189397	0.995647308
0.071807822	0.96666898
0.018199551	0.839796233
0.244445383	0.015525945
Average: 0.135761243723	Average: 0.75597823448

3.2 10,000 Iterations:

Mozart	Rachmaninoff
0.229537293	0.956960927
0.073226477	0.860681685
0.492563182	0.998574173
0.014352747	0.998577581
0.157468297	0.991409143
0.034547724	0.899635762
0.418378482	0.020019487
Average: 0.202867743222	Average: 0.817979822574

3.3 14,000 Iterations:

Mozart	Rachmaninoff
0.155292166	0.926540944
0.045178567	0.775823792
0.355731672	0.996341613
0.007536147	0.997391511
0.087238548	0.984396867
0.033244141	0.921939723
0.289612683	0.018089673
Average: 0.139119131994	Average: 0.802932017607

3.4 20,000 Iterations:

Mozart	Rachmaninoff
0.246987378	0.910667436
0.076599769	0.78254246
0.499717472	0.997315343
0.017284228	0.997505054
0.166669487	0.987666405
0.04632631	0.927365885
0.428092399	0.034919879
Average: 0.211668148902	Average: 0.805426066072

4 Discussion

The results of this experiment suggest a relationship between the composer of a piece of music and the types of harmonies found in the piece. Music by Rachmaninoff and Mozart was chosen to be analyzed because of their very different styles, and by using a neural network the computer could learn to reliably differentiate between these two composers. It seems that harmonies are one of the key reasons that music from different time periods and styles can sound so different. This experiment could be extended and improved by having access to more data, more composers, and more musical analysis. While harmonies seem to be important in defining an artists compositional style, factors such as note entropy, note density, and rhythmic complexity could certainly play a role as well. With more advanced networks and machine learning techniques, perhaps this greater amount of data could be used to obtain even more accurate results. Nevertheless, this project has suggested that harmony is a key difference between music of different styles, a difference that humans may be trained to pick up subconsciously but that a computer can also learn to analyze and interpret.

References

[1] Backer, Eric, and Peter van Kranenburg. *On musical stylometry - a pattern recognition approach*. N. pag. Delft University of Technology, 21 Jul. 2004. Web. 19 Jan. 2010

[2] Bagci, Ulas, and Engin Erzin. *Inter Genre Similarity Modelling for Automatic Music Genre Classification*. N. pag. N.p., 18 July 2009. Web. 24 Oct. 2009.

[3] Cilibrasi, Rudi, Paul Vitanyi, and Ronald De Wolf. *Algorithmic Clustering of Music*. N. pag. University of Amsterdam, 24 Mar. 2003. Web. 24 Oct. 2009.

[4] Dannenburg, Roger B., Thom, Belinda, and David Watson. *A Machine Learning Approach to Musical Style Recognition*. 344-347. pag. Carnegie Mellon University, Sep. 1997. Web. 19 Jan. 2010

[5] Kozbelt, Aaron, and Zahava Burger-Pianko. *Words, Music and Other Measures: Predicting the Repertoire Popularity of 597 Schubert Lieder*. 191-203. pag. Psychology of Aesthetics, Creativity, and the Arts, Vol 1 Issue 4, Nov. 2007. Web. 25 Apr. 2010

[6] Liou, Cheng-Yuan, Tai-Hei Wu, and Chia-Ying Lee. *Modelling Complexity in Musical Rhythm*. N. pag. National Taiwan University, 26 Mar. 2007. Web. 24 Oct. 2009

[7] Lu, Cheng-Che, and Vincent Tseng. *A Novel Method for Personalized Music Recommendation*. N. pag. National Cheng Kung University, 2009. Web. 6 Apr. 2010