Determining musical genre of MIDI files with SVM

John Gardiner

December 12, 2017

1 Introduction

Classifying music has been a problem of interest in machine learning research for decades [2, 4, 5]. With the advent of streaming services with large selections of music, this problem has important applications. In particular, music recommendations based on the content of music files, rather than just usage data, allow streaming services to include new or less-well-known music in their recommendations [3]. In this project we attempt a simple version of this task, sorting MIDI files between two musical genres.

We use a MIDI file database with 10266 classical (i.e. Western art music) files and 976 EDM (including house, techno, "trance," etc.) files to create and test an SVM algorithm for distinguishing MIDI files in the two genres. While most attempts to classify music involve classifying waveforms, by using MIDI files we will be attempting to determine the genre from musical data rather than sound data. This both simplifies the problem, as MIDI files are smaller, and potentially makes the problem more interesting from a musical point of view, as recording or performance information is not present, just music. We are also not interested in "superficial" characteristics of the pieces like the length of the piece or the instrumentation, and choose our model to with this in mind, as we will explain in Section 2.

1.1 MIDI files

A MIDI file is organized as a sequence of "events." Most "events" are either the beginning or ending of a note. Along with these notes comes information about their pitch, volume, and the instrument playing them, etc. We used the MATLAB MIDI toolbox described in [6] to convert each MIDI file into MATLAB matrix, the rows designating individual notes. The eight columns of the matrix designate track numbers, channel number, pitch, volume, note start time, note end time, message number of note on, and message number of note off, respectively. The track and channel numbers are typically meant to express instrumentation, while the message numbers designate the order in which the beginnings and endings of the notes are encoded in the MIDI file.

2 Model

We use a soft-margin SVM to classify the files. A soft-margin SVM model seeks to separate the two classes of training points by a boundary with as wide a margin as possible and imposes a linear cost for points found close to or on the wrong side of the boundary. The position of the boundary depends only on the points nearest to it, making it robust to the presence of outliers, which may very well exist in our data. Also, we anticipate some overlap between the classical and EDM clusters. The choice of a "soft-margin" SVM allows for some number of points to be misclassified to prevent overfitting the boundary.

2.1 SVM models

The output of an SVM model is a function $y(x) = w^T\phi(x) + b$ where $y(x) > 0$ classifies a piece x as classical and $y(x) < 0$ classifies x as EDM. Here w is a matrix and ϕ is a nonlinear map. We denote the training data points by x_n and their true classifications, either 1 for classical or -1 for EDM, as t_n. Then solving the model involves minimizing

$$\frac{1}{2}\|w\|^2 + C\sum_n \xi_n \quad \text{where} \quad \xi_n = \begin{cases} \big(t_n - y(x_n)\big)\theta\big(t_n - y(x_n)\big) & \text{for} \quad t_n = 1 \\ \big(y(x_n) - t_n\big)\theta\big(y(x_n) - t_n\big) & \text{for} \quad t_n = -1 \end{cases}$$

with constraints $t_n(w^T\phi(x_n) + b) \geq 1$. The first term maximizes the distance from the boundary to the datapoints while the second term imposes a cost for points close to the boundary or on the wrong side of it. This problem is dual to the problem of minimizing

$$\sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_m t_m \langle\phi(x_m), \phi(x_n)\rangle t_n a_n$$

over the a_n given the constraints $\sum_n a_n t_n = 0$ and $0 \leq a_n \leq C$ [1]. Expressed in this form it is clear that we can generalize the model by replacing the maps $\langle\phi(x), \phi(y)\rangle$ with a kernel $K(x, y)$. So solving our model entails the quadratic programming problem of minimizing

$$\sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_m t_m K(x_m, x_n) t_n a_n \tag{$\star$}$$

with the above constraints, where K is an appropriate kernel. Once the optimal a_n values are found, the classifying function y is then

$$y(x) = \sum_n a_n t_n K(x, x_n) + \frac{1}{|M|}\sum_{n\in M}\left(t_n - \sum_m a_m t_m K(x_n, x_m)\right) \tag{$\star\star$}$$

where M are the indices of a_n with $0 < a_n < C$ [1].

2.2 Choice of kernel

With the above set up only the kernel remains to be specified. We try two different choices of kernel corresponding to different ways of organizing the MIDI information in vectors. As mentioned in 1.1 the output from reading a MIDI file in MATLAB is a matrix with eight columns where each row describes a note. We remove columns 7 and 8 which correspond to message number of notes on and off as that information has no musical meaning. We also remove columns 1 and 2 corresponding to track and channel, to avoid classifying based on instrumentation information. What remains are note begin times, end times, pitches, and volume. For our first choice of kernel we leave the matrices thus, with their rows corresponding to notes.

In an attempt to capture harmonic content of the music, for our second kernel, we rearrange the information into musical "chords." We take all the times corresponding to the beginning or ending of any note and for pairs of consecutive times we record which pitches are playing. We consider the pitch values modulo 12, as pitches 12 values apart are harmonically equivalent. For each pair of consecutive times we write a row with 12 entries, one for each of the 12 possible pitches (mod 12). The entries of the row are 1 if the corresponding pitch is playing and 0 otherwise if that pitch is not playing. As an examle a MIDI file in our database for Beethoven's Minuet in G begins

$$\begin{array}{cccccccccccc}
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}.$$

The matrices for different pieces have different amounts of rows. This presents two problems. First it raises the question of what an appropriate distance measure would be between two vectors of different sizes. Second we would like our model to not take the length of the piece, as measured in number of notes or number of chords, into account. So from the rows of each piece's matrix we select a set number D of rows using a process described in Section 3. The distance between two pieces is then the Euclidean distance between their corresponding resized matrices. We use a gaussian kernel $K_{nm} = K(x_n, x_m) =$

$\exp\left(-\frac{\|x_n - x_m\|^2}{\sigma_n \sigma_m}\right)$ where x_n are the matrices with D rows described above and σ_n is the distance from the n-th piece to its k-nearest neighbor in the training set.

3 Algorithm

From the original data we randomly select and set aside 200 classical and 200 EDM points for testing. On the principle that the absolute pitch of a piece is unimportant to its genre, we shift the pitch values of the remaining datapoints by a constant value to get additional datapoints. From these we choose 4500 classical datapoints and 4500 EDM datapoints as training.

Our kernel function takes as input matrices with D rows, corresponding to either notes or chords. For pieces with more than D notes (or chords) we simply randomly sample D rows without replacement. For pieces with less than D rows, we repeat the piece as many times as we can without getting more than D rows, and then we randomly sample from the piece without replacement until we have exactly D rows. The notes are then randomly permuted. This process gives a matrix with the necessary D rows while ensuring maximally equal representation of each note.

We set the model parameters $D = 400$, $k = 10$, and $C = 10$. We solve the quadratic programming problem $(\star)$, with the kernels described in 2, using MATLAB's `quadprog` function. The function returns approximately optimal values for the a_n. With the a_n we use $(\star\star)$ to calculate the classifying function $y(x)$. The values returned by `quadprog` are never exactly 0, so when calculating $y(x)$ we set a cutoff of $\epsilon = 10^{-6}$ and exclude any a_n less than ϵ from the set M. Finally, for each test point we compare its genre with the sign of $y(x)$.

4 Results

Several trials give an average accuracy of approximately 85% for both kernels. So we see that organizing the note information in chords does not seem to improve the accuracy. We find that most of the a_n are nonzero, i.e. most of the training data points are support vectors. This negates one advantage of SVM, which is that the classifier is a function of a usually small number of the support vectors.

Projecting the datapoints onto the two principle components shows that the clusters for classical and EDM pieces overlap greatly (see figure 1). This suggests that the ideal feature map ϕ that would separate our data is very complicated. The EDM pieces are more closely clustered together likely allowing for the accuracy of our model, such as it is.

In a piece of music, musical meaning is not local in time, but spread out. Our kernels only took into consideration notes or chords and not any temporal relations between them, and thus miss most of the musical structure. This is a difficulty of using SVM or other kernel methods for our particular problem. Most choices of kernel would miss important

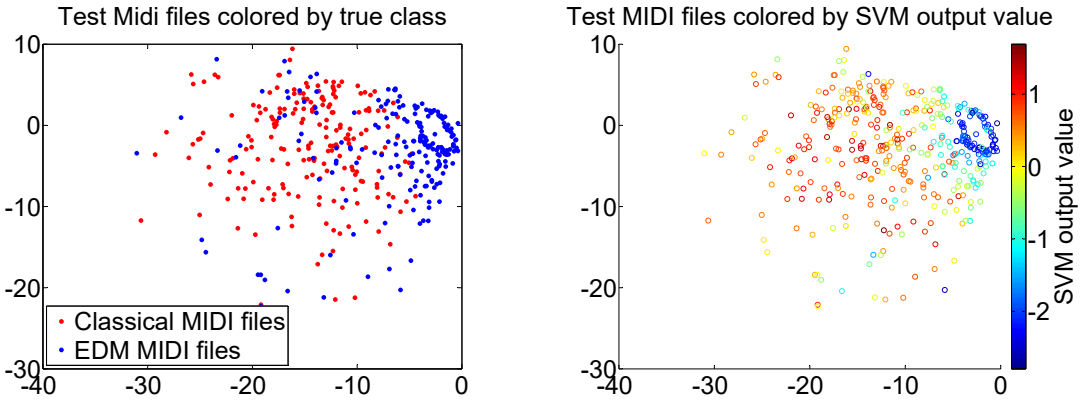

Figure 1: Test points projected onto the two principle components of the training data. These are datapoints with rows corresponding to chords.

structure. Constructing a good kernel for the problem might require a sense of "distance" between datapoints, and it is not clear what an appropriate measure of "distance" between musical pieces would be. Perhaps having a full answer to that question would obviate the need for something like an SVM classifier in the first place.

Because a kernel method requires too much understanding of the structure of music up front when choosing a kernel, a better choice of model might be one which could potentially "discover" musical structure on its own, say a convolutional neural network.

One difficulty that any model would encounter when faced with our chosen task is that it is sometimes difficult even for a human. In practice, to decide the genre of a piece, people use cues like the length and, importantly, the instrumentation, things we explicitly did not consider as part of our problem. It seems possible for the genre of a piece to be underdetermined from just the notes alone.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[2] Roger B. Dannenberg, Belinda Thom, and David Watson. "A Machine Learning Approach to Musical Style Recognition". In: *1997 International Computer Music Conference* (1997), pp. 344–347.

[3] Sander Dieleman. *Recommending music on Spotify with deep learning*. 2014. URL: `http://benanne.github.io/2014/08/05/spotify-cnns.html` (visited on 12/11/2017).

[4] A. R. Rajanna et al. “Deep Neural Networks: A Case Study for Music Genre Classification”. In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. 2015, pp. 655–660. DOI: 10.1109/ICMLA.2015.160.

[5] N. Scaringella, G. Zoia, and D. Mlynek. “Automatic genre classification of music content: a survey”. In: *IEEE Signal Processing Magazine* 23.2 (2006), pp. 133–141. ISSN: 1053-5888. DOI: 10.1109/MSP.2006.1598089.

[6] Ken Schutte. *MATLAB and MIDI*. 2012. URL: http://kenschutte.com/midi (visited on 12/10/2017).